Zero Knowledge Proof
Fundamentals and Applications

Nicolò Zarulli
matricola 296235

a.a. 2021-2022 - Università di Parma

Indice

Capitolo 1

Introduzione

Questo documento vuole essere un contenitore di informazioni relative all'approccio crittografico *Zero Knowledge Proof*. Nasce come mia personale necessità di raggruppare un insieme di ottime fonti ed informazioni, all'interno di uno stesso documento leggibile e coerente.

Conseguentemente la struttura dello stesso sarà molto semplice ed intuitiva: vi saranno dei *capitoli* e delle *sezioni* interne per approfondire determinati argomenti e dettagli dello stesso.

Per ora, data inizio Marzo 2022, seguirò una stesura che va di pari passo al mio lavoro di ricerca svolto nel *Dipartimento di Ingegneria dell'Informazione* presso l'Università di Parma, con un attento riguardo verso una traduzione in tesi di laurea triennale.

Capitolo 2

Zero Knowledge Proof

2.1 ZKP: Uno sguardo introduttivo

La *Dimostrazione a Conoscenza Zero* o *Zero Knowledge Proof* è un approccio/sistema crittografico attraverso il quale un entità detta **Prover** è in grado di dimostrare di essere in possesso di un ***Informazione*** o ***Witness*** ad un altra entità detta **Verifier**, senza però rivelarne il contenuto e senza rivelarne alcuna conoscenza a riguardo.

Figura 2.1: Zero Knowledge Proof

E' generalmente considerato un **Interactive Verification Protocol** in quanto richiede un interazione diretta tra i due enti comunicanti; si vedrà più avanti che in certi ambiti questo tipo di implementazione può risultare poco conveniente, indirizzando lo sguardo e l'interesse verso un implementazione più asincrona e *Non Interactive*.

2.2 ZKP: La Struttura

Come ogni protocollo richiede, delle regole devono essere stabilite e rispettate da chiunque abbia la necessità di utilizzare lo stesso. Nessuna eccezione viene fatta per i protocolli a Zero Knowledge.

Prima di introdurre determinate *regole* si può vedere un generico protocollo ZKP come un algoritmo che segue l'implementazione di 3 fasi sequenziali; esse saranno necessarie per stabilire un terreno di *gioco* equo e coerente sia rispetto al Prover che rispetto al Verifier.

La Dimostrazione a Conoscenza Zero cerca quindi di risolvere un certo problema che si pone tra due entità comunicanti:

- il Prover conosce un segreto, per il quale vi è una ricompensa in gioco. Egli, prima di rivelare il segreto, vuole ottenere la ricompensa.
- il Verifier, colui che ha messo in palio la ricompensa, vuole accertarsi che il Prover conosca effettivamente il segreto. Prima di pagare la ricompensa, vuole vedere il segreto stesso.

E' facile intuire come entrambe le entità non intendono mettere a rischio la loro integrità prima di aver ricevuto una prova concreta, sia essa il segreto o la ricompensa. E' in questo caso che entra in gioco la Dimostrazione a Conoscenza Zero: il Prover è in grado di dimostrare al Verifier di possedere un certo segreto, senza però rivelargli nulla di esso.

Per poter arrivare a questo grado di certezza, dove il Prover riesce a convincere il Verifier di essere in possesso di un informazione/segreto, detto *Witness* grazie alla computazione di un certo *Proof*, entrambe le entità dovranno seguire un algoritmo, costituito da tre fasi:

1. **Witness Phase**: Il Prover computa una dimostrazione, detta **Proof**, sulla base di un **segreto** e contenente uno *Statement*. Questa viene inviata al Verifier, che potrà analizzarla per capire la validità del Prover.

2. **Challenge Phase**: Ottenuto il Proof, il Verifier inizia a fare delle domande al Prover. Permettono al Verifier di capire se il Prover sia onesto o malizioso.

3. **Response Phase**: Il Prover riceve le *domande* dal Verifier; per ognuna di queste ne formula una risposta. Il Verifier valuta le risposte per poi prendere una decisione definitiva, accettando o rifiutando il Proof computato nella prima fase dal Prover.

E' importante notare delle dirette conseguenze:

- Se il Prover conosce veramente il segreto, riuscirà sempre a fornire risposte convincenti al Verifier. Altrimenti avrà, nel breve termine, una certa probabilità di riuscire ad ingannarlo ma a lungo andare non riuscirà a convincerlo.

- Se dopo una prima esecuzione dell'algoritmo il Verifier non risulta convinto, può ritornare alla Challenge Phase.

- Il lavoro del Prover potrebbe essere computazionalmente oneroso, dipende dal meccanismo utilizzato nella Response Phase.

- Nelle fasi descritte **nessun tipo di informazione privata verrà condivisa**.

2.2.1 Un esempio astratto

Per comprendere al meglio questo tipo di approccio crittografico, è molto efficace presentare al lettore una storia molto conosciuta che evidenzia ogni singola fase della crittografia a Zero Knowledge.
I due interlocutori saranno:

- **Bob**: colui che ha scoperto un certo segreto. Bob sarà il Prover.
- **Alice**: colei che garantisce una ricompensa per la rivelazione del segreto. Alice sarà il Verifier.

"Bob ha scoperto la parola segreta per aprire la porta in una caverna. La caverna ha una forma circolare con l'entrata da un lato e la porta che blocca l'altro lato. Alice dice a Bob che lo pagherà per il segreto, ma non prima di essere sicura che lui lo conosca davvero. Bob si dice d'accordo a rivelargli il segreto, ma non prima di aver ricevuto i soldi. Pianificano quindi uno schema con il quale Bob può dare prova ad Alice di conoscere la parola ma senza rivelargliela:

1. *Alice aspetta fuori all'entrata mentre Bob entra nella caverna; Bob sceglie uno dei due sentieri da percorrere tra A e B per raggiungere la porta. [**Witness Phase**]*
2. *Alice entra nella caverna e grida a Bob il nome del sentiero con il quale dovrà tornare all'entrata. [**Challenge Phase**]*
3. *Se Bob conosce la parola segreta, riuscirà sempre ad aprire la porta e ritornarne da Alice con il sentiero richiesto. [**Response Phase**]*

***Se Bob appare in modo affidabile probabilmente conosce veramente il segreto.**"*

Se si ipotizza che **Bob conosca veramente la parola magica**, è facile: se necessario apre la porta e ritorna attraverso il sentiero desiderato. È da notare che Alice non conosce il sentiero con il quale Bob ha raggiunto la porta.

Se si ipotizza però che **Bob non conosca il segreto**, egli avrebbe il 50% di possibilità di ritornarne con il sentiero richiesto da Alice, per ogni prova effettuata. Seguendo il *principio della probabilità ad eventi indipendenti*, la possibilità che Bob riesca ad adempire correttamente a tutte le richieste di Alice **senza conoscere il segreto** diventa statisticamente molto piccola (circa 0.01%). Al contrario, se Bob risponde in modo affidabile ad ogni evento imposto da Alice, lei potrà essere statisticamente convinta che Bob sia effettivamente in possesso del segreto (circa 99.99%).

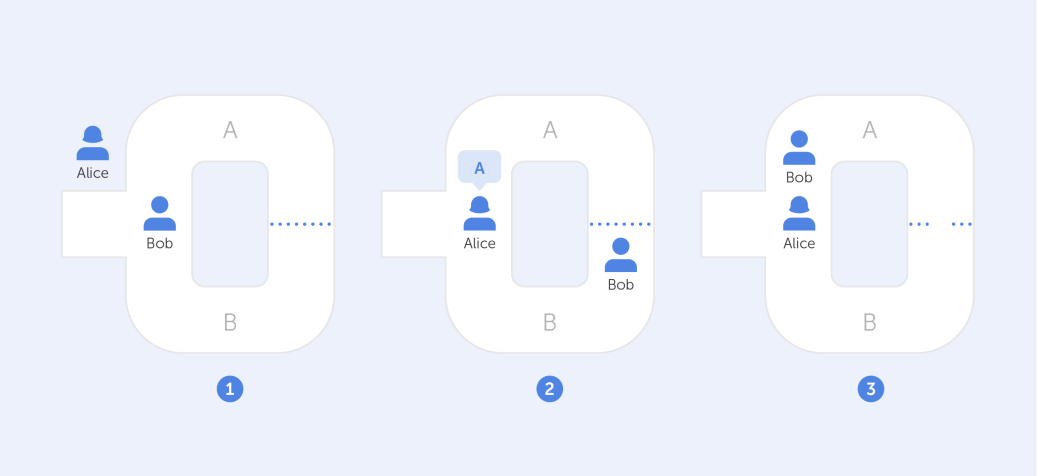


Figura 2.2: ZKP, un esempio attraverso una storia astratta

2.3 ZKP: Le Proprietà

La nozione di *Zero Knowledge* fu introdotta negli anni '80 da un gruppo di ricercatori MIT (*Goldwasser, Micali, Rackoff*) e descrive in modo molto più approfondito i concetti introdotti nelle sezioni precedenti. Attraverso un **Interactive Proof System** un Provider scambia dei messaggi con un Verifier per convincerlo che un certo *Mathematical Statement* in suo possesso sia vero.

Una naturale conseguenza di questa introduzione pone un certo riguardo alla protezione del segreto del Verifier, mettendo in conto un possibile comportamento **malizioso** da parte del Prover (è un ente in una posizione

avvantaggiata, *ha tutto da guadagnare e nulla da perdere*): *"un Prover malizioso tenta di ingannare un Verifier onesto nel credere ad uno statement falso"*. Questo è un punto fondamentale nello studio di questo determinato approccio crittografico, a riguardo anche ad una possibile implementazione su sistemi informatici veri e propri.

I ricercatori MIT sollecitarono però una problematica opposta e assolutamente **non trascurabile**: *"E se un Prover onesto non si potesse fidare di un Verifier, in questo caso, malizioso?"*. Questa situazione concerne il problema dell'**information leakage**: di quanto dettaglio/informazioni-extra ha effettivamente bisogno il Verifier durante la fase di valutazione del Proof fornito da un Prover? Il Verifier si basa sulla esclusiva fiducia che il Proof fornito dal Prover sia attendibile e non malizioso, ossia associato ad un segreto veritiero. Di conseguenza, entrambi i lati del protocollo (a.k.a. Prover & Verifier) necessitano di **sicurezze** sulle intenzioni dell'altra entità in comunicazione.

Vi è la necessità di introdurre delle **proprietà fondamentali** che ogni *Zero Knowledge Protocol* deve possedere ed implementare:

Completeness (completezza): Se lo *statement* del Proof è veritiero, un Prover onesto riuscirà sempre a convincere un Verifier altrettanto onesto. La probabilità non corrisponderà mai al 100%, ma l'obiettivo è quello di avvicinarsi il più possibile.

Soundness (solidità/correttezza): Se lo *statement* del Proof è falso, nessun Prover malizioso potrà convincere un Verifier onesto che l'affermazione sia vera; la probabilità di riuscire di convincerlo è resa il più bassa possibile, tenendo conto dei meccanismi utilizzati per implementare il protocollo.

Zero Knowledge (conoscenza zero): E' la proprietà che da il nome alla tecnica crittografica e garantisce che il Prover non condivida troppe informazioni, anche sensibili, ad un Verifier (onesto o disonesto che sia). Se lo statement risulta veritiero, nessun Verifier disonesto potrà sapere altro che tale informazione.

2.3.1 Un esempio più tecnico

In questo caso l'impostazione del modello è la seguente:

- Un Prover conosce un certo numero segreto S e vuole dimostrarlo ad un Verifier.

- Alla base del meccanismo di generazione del Proof, della computazione di una risposta da parte del Prover e del meccanismo di validazione degli statement da parte del Verifier, vi è l'**aritmetica modulare**.

L'esecuzione del protocollo vedrà le fasi introdotte in precedenza, con l'aggiunta di una quarta fase (*Verification Phase*) a carico del Verifier.

Esecuzione

1. **Witness Phase**: Il Prover computa

   $$v = s^2(modn) \quad \text{con} \quad n = pq \quad \text{e} \quad \sqrt{n} \leq s \leq n-1 \quad \text{e} \quad s \neq (p \wedge q)$$

   L'avversario non può estrarre il segreto s dal numero computato v.
   I numeri p e q sono due *large private primes*, ossia il risultato della moltiplicazione di due numeri primi.
   Il Prover sceglie un intero randomico r in $\big(1 \leq r \leq n-1\big)$ e computa

   $$x = r^2(modn) \qquad \text{dove x rappresenta uno } \textbf{statement}$$

   ed infine invia x al Verifier.

2. **Challenge Phase**: Il Verifier sceglie

   $$\text{un bit } \alpha \in \big\{0, 1\big\}$$

   e lo invia al Prover.
   Ogni bit descrive una challenge differente da proporre al Prover.

3. **Response Phase**: In base al bit α ricevuto, il prover svolgerà una challenge differente:

Se $\alpha = 0$: il Prover imposta il Proof come

$$\textbf{Proof} \quad \varphi = r$$

Se $\alpha = 1$: il Prover computa il Proof come

$$\textbf{Proof} \quad \varphi = rs(modn)$$

Successivamente invia il Proof φ al Verifier.

4. **Verification Phase**: Il Verifier valida il Proof φ ricevuto dal Prover. Se vale

$$\varphi^2 = x(v^\alpha)(modn)$$

allora accetta il φ ricevuto.
Decide poi se inoltrare una nuova *challenge* al Prover o confermare che quest'ultimo conosce effettivamente il segreto.

Questa implementazione verifica le proprietà chiave di ogni ZKP:

Completeness : Presupponendo che il *segreto s* conosciuto dal Prover sia *vero*, egli riesce sempre ad inoltrare un Proof ($\varphi = r$ o $\varphi = rs(modn)$) corretto al Verifier, dato che può computare senza problemi lo *statement* x.

Soundness : Se il Prover non conosce il *segreto s* reale, ad ogni *challenge* ricevuta dal Verifier avrà il 50% di probabilità di inviargli un Proof φ corretto, ossia che riesca a passare la *Validation Phase*. Al contrario, il Verifier rifiuterà il Proof φ fornito dal Prover sempre con una probabilità del 50%, per ogni evento effettuato. E' quindi un modello molto simile al problema della caverna; un Prover malizioso non riuscirà ad ingannare per sempre un Verifier onesto.

Se φ è verificato T volte, la probabilità del Prover di ingannare il Verifier risulta

$$P(E) = (\frac{1}{2})^T$$

con l'evento $\{E$: Il Prover inganna il Verifier senza conoscere il *segreto* $s\}$

Zero Knowledge : Il Verifier conosce solamente i numeri *v, x,* φ, per ogni esecuzione del protocollo. Il Prover ha la garanzia di poter mantenere private le sue informazioni sensibili, ma soprattutto, ha la sicurezza, grazie all'aritmetica modulare, che il Verifier non possa risalire al suo *segreto* s conoscendo solamente i valori da egli forniti.

Grazie all'implementazione di queste tre proprietà, sia un Prover onesto che un Verifier onesto sono protetti da attacchi maliziosi.

2.4 ZKP: Classificazione

Gli esempi di Zero Knowledge Proof illustrati nelle sezioni precedenti richiedono tutti l'interazione diretta tra le due entità comunicanti, attraverso lo schema di *domande & risposte*. Ne risalta quindi un'importante caratteristica: l'**interazione**. Basandosi su questa si può evidenziare una prima classificazione dei *Zero Knowledge Proof*:

- **Interactive Zero Knowledge Proofs**: si basano su una *comunicazione sincrona* e diretta tra i due enti comunicanti. In questo modello, il Verifier metterà a disposizione del Prover una serie di *task* o azioni, che dovrà risolvere per dimostrare di possedere realmente un Witness o segreto. Generalmente il lavoro del Prover risulta computazionalmente più oneroso, in base al meccanismo/algoritmo richiesto per risolvere i task richiesti.

- **Non-Interactive Zero Knowledge Proofs**: In questo caso non vi è la necessità di avere un interazione diretta tra Prover e Verifier; la *validazione* del *proof* fornito dal Prover può essere sostenuta ad uno stage più avanzato, anche da un terzo ente fidato. Questa tipologia di ZKP può richiedere l'utilizzo di *software* e *risorse* addizionali, ed utilizza una comunicazione *asincrona* tra Prover e Verifier.

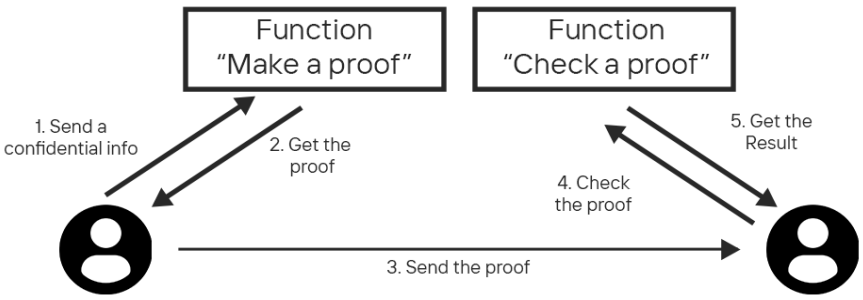


Figura 2.3: Non-Interactive ZKP

2.4.1 Uno sguardo più mirato: Interactive ZKP

Agli albori di questo approccio crittografico, le sue applicazioni ed integrazioni vedevano principalmente la filosofia di tipo *Interactive ZKP*. Questo tipologia di integrazione però coinvolge un approccio che si rivolge principalmente verso un ambito molto più applicativo e matematico, coinvolgendo soprattutto i sistemi informatici. In questo caso il *Zero Knowledge Proof* viene utilizzato per dimostrare a qualcuno la conoscenza di un fatto *matematico*, senza rivelarne però nessun informazione sensibile sullo stesso.

In tempi recenti però l'applicazione di questo tipo di approccio crittografico viene vista ed utilizzata principalmente nei sistemi distribuiti, soprattutto nell'ambito delle *blockchain*. Per questioni di prestazioni, sicurezza ed evoluzione del modello, si tende a preferire un approccio più *Non Interactive ZKP*.

E' comunque importante evidenziare certi casi nei quali è consigliato un approccio *Interactive*:

- **Knowledge of a three-coloring graph**: è un problema molto richiesto in ambito delle telecomunicazioni dove il modello si basa sulla teoria dei grafi. Sia il Prover che il Verifier conoscono un certo grafo pubblico: il Prover vuole dimostrare di possedere un algoritmo che permette di ottenere un'istanza dello stesso grafo ma a 3 colori, dove ogni nodo ne tocca almeno un altro con un colore differente dal suo. Da questo tipo di problema ne deriva un modello non-interactive ampiamente utilizzato in blockchain: il **zkSnark**.

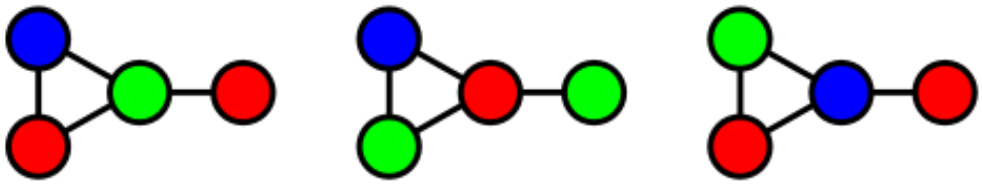


Figura 2.4: varie istanze three-coloring di un certo grafo pubblico

- **Knowledge of a discrete logarithm of some residue module p**: dato un numero primo pubblico g detto *generatore*, un numero primo pubblico p detto *modulo* e un certo *residuo* r, conosco un certo valore x tale che $g^x = y(modp)$, che è a tutti gli effetti la definizione di *logaritmo discreto*.

- **Knowledge of a private key corresponding to a publicly-known public key**

2.5 ZKP: Modelli di Implementazione

L'approccio della Dimostrazione a Conoscenza Zero trova grande applicazione nell'era del digitale e delle comunicazioni Internet ma, soprattutto, nella categoria dei *Sistemi Distribuiti* con particolare riferimento alle *Blockchain*.

Come illustrato, vi sono vari modelli e varie filosofie di implementazione di questo tipo di sistema crittografico; è utile introdurre e studiare diversi modelli ZKP pensati per un implementazione concreta. Questi modelli possono far parte della filosofia *Interactive ZKP* o della *Non-Interactive ZKP*.

2.5.1 zkSnark

Il modello zkSnark, acronimo di *zero knowledge Succinct Non-interactive ARgumentoof Knowledge*, è un modello ZKP di tipo Non-Interactive il quale introduce una nuova proprietà: la **sinteticità** (*succinctness*). Questa sposta l'obiettivo del modello verso una migliore implementazione a livello di *complessità*, quindi di un attento riguardo verso il dispendio computazionale delle risorse in gioco.

Il modello zkSnark coinvolge tre entità: un **Prover P**, un **Verifier V** ed un terzo ente fidato detto **Setup S**. Quest'ultimo avrà il compito di generare una coppia di chiavi, necessarie a P e a S per poter, rispettivamente, generare un *proof* π ed effettuarne la verifica dello stesso.

Ciò che è importante evidenziare in zkSnark sono le seguenti caratteristiche:

1. La *Verification Phase* è eseguita in un breve *running time*.

2. La dimensione del proof π è di soli pochi bytes.

3. Il Prover ed il Verifier non sono obbligati a comunicare in modalità *sincrona*.

4. Il proof π può essere verificato da un qualsiasi Verifier, secondo una filosofia di tipo **off-line way**.

5. L'algoritmo che il Prover P dovrà eseguire appartiene alla **classe di complessità NP**.

Queste caratteristiche verranno approfondite nel capitolo successivo, dedicato interamente al modello zkSnark.

zkSnark Requirements	
Trusted Setup	Required
Prover Algorithm	$O(nlogn)$
Verifier Algorithm	$O(1)$
Proof Size / Communication complexity	$O(1)$
Implementation Technique	Quadratic Arithmetic Programs

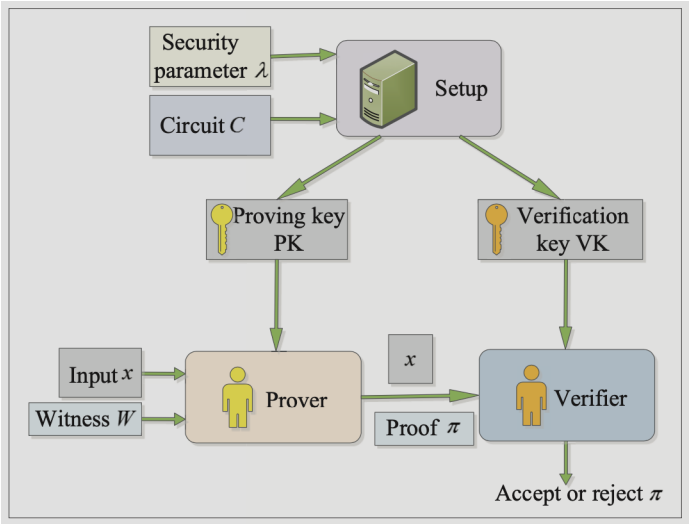


Figura 2.5: zkSnark schema

Ben-Sasson's Model

Basandosi su zkSnark, *Ben-Sasson et al.* proposero un nuovo modello zkSnark per circuiti aritmetici, incentrando il loro obiettivo nello scindere in due processi indipendenti la creazione di una coppia di chiavi dalla validazione di un proof tra due entità; questo modello vede quindi la distinzione tra una **off-line phase** ed una **on-line phase**. Dato che si basa su zkSnark, anche questo modello vede l'utilizzo di **NP-Statements**, i quali sono istanze di

NP-Problems che possono essere verificate/provate seguendo le proprietà di *Zero Knowledge*.

Off-line Phase : Il Setup S in questo caso viene sostituito da un *zkSnark key generator*, il quale prende in ingresso un *circuito universale*, otteunuto da un programma generatore di circuiti a partire da tre vincoli strutturali usati come input: un **program size bound**, **input size bound** ed un **time bound**. Il *zkSnark KG* genera la coppia di chiavi (PK, VK) a partire dal circuito universale ottenuto.

On-line Phase : Un Prover P computa un certo *proof* π a partire da un *circuito assegnato* e dalla chiave PK ricevuta dallo *zkSnark KG* della *off-line phase*. Il Circuito è il risultato della computazione di una certa *Witness map*, la quale rappresenta la conoscenza del segreto da parte di P.

Un Verifier V riceve un certo *proof* π da un Prover. Deciderà se accettarlo o rifiutarlo usando la chiave VK ricevuta dallo *zkSnark KG*.

Il vantaggio di questo modello vede l'aggiunta di un ulteriore livello di integrità e privacy a riguardo della generazione di una coppia di chiavi (PK, VK) coinvolte nella *generazione/validazione* di un *proof* π.

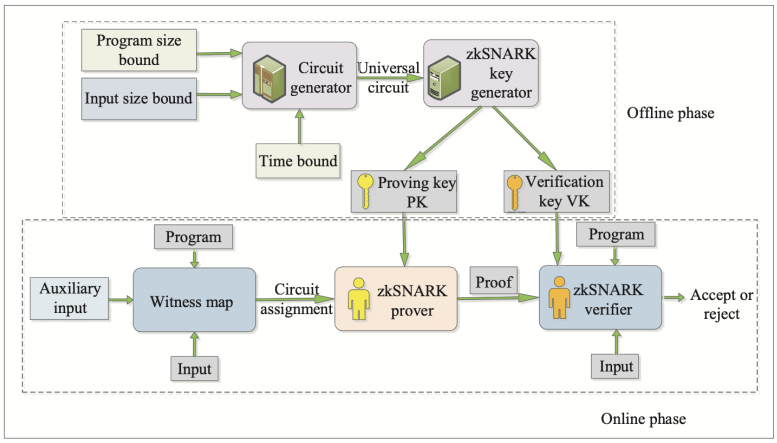


Figura 2.6: Ben-Sasson's zkSnark schema

2.5.2 Ligero

Ligero è un modello che propone un *argomento interattivo a zero knowledge* che vuole essere più compatto. La sua caratteristica principale è quella di non avere bisogno di un *trusted setup* tra i due enti comunicanti. Fu introdotto in una pubblicazione scientifica scritta da *Ames Scott et al.* e risalente al 2017.

Il vantaggio di questo protocollo vede l'implementazione di un argomento a zero knowledge partendo sempre da problemi NP la cui complessità computazionale risulta proporzionale alla radice quadrata della dimensione del circuito di verifica dell'argomento stesso.

Inoltre, Ligero può essere costruito a partire da una qualsiasi **funzione di hashing collision-resistant**. Alternativamente, può essere reso *non-interactive* se basato sul modello **random-oracle**:

> Un modello random-oracle è utilizzato per modellare funzioni crittografiche di hashing all'interno di schemi dove sono necessarie delle forti assunzioni di randomicità sull'output della stessa funzione hash. Un **random-oracle** o *scatola nera*, è una funzione matematica che associa ogni possibile domanda ad una risposta casuale, scelta uniformemente all'interno del suo dominio di output.

Ligero riesce quindi ad implementare con gran efficenza e concretezza degli argomenti zkSNARK che non richiedono la presenza di un **trusted setup** o **public key cryptosystem**. Questo li rende estremamente efficaci in presenza di circuiti di verifica ad ampie dimensioni.

Ligero Requirements	
Trusted Setup	Not Required
Prover Algorithm	$O(nlogn)$
Verifier Algorithm	$O(n)$
Proof Size / Communication complexity	$O(\sqrt{n})$
Implementation Technique	Interactive Oracle Proofs

2.5.3 Bulletproofs

Bulletproofs è un modello di dimostrazione a zero knowledge che è strutturato sul problema del logaritmo discreto e fu introdotto da *Bunz et al.* in una pubblicazione scientifica apposita. Questo modello è stato pensato per migliorare la natura distribuita e trustless delle Blockchains, dando la possibilità di apportare sostanziali miglioramenti sulla sicurezza e confidenzialità delle transazioni in criptovalute. Per ottenere questi risultati, il modello è strutturato come un *Non-Interactive Zero Knowledge Proof* protocol che garantisce una dimensione contenuta dei *proof* e che non richiede un *setup fidato*. I Bulletproofs migliorano di molto la dimensione lineare, contenuta in n, dei proof normalmente utilizzati in transazioni confidenziali per Bitcoin e altre criptovalute. Inoltre, i Bulletproofs supportano una funzione molto importante e che riguarda il concetto di *range proofs*: il *multi-party computation (MPC)* protocol, che permette di aggregare i proof di un party di entità in un unico proof risultante, il quale non permette di rivelare nessun informazione dei segreti dei proof contenuti. I Bulletproofs sono costruiti sulla tecnina EUROCRYPT 2016, sempre formulata dal gruppo di ricercatori a capo del progetto. In poche parole, questi sono i punti chiave che questo modello introduce:

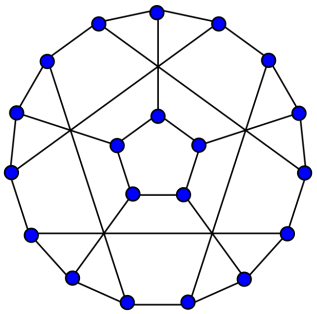
- Short zero-knowledge proofs for general arithmetic circuits, based on the discrete logarithm assumption without a trusted setup.
- Extremely efficient range proofs implemented via the MPC protocol.
- Applications focused onto an improving transaction confidentiality for the Blockchain environment.

Bulletproofs Requirements	
Trusted Setup	Not Required
Prover Algorithm	$O(c)$
Verifier Algorithm	$O(c)$
Proof Size / Communication complexity	$O(\log n)$
Implementation Technique	Discrete Logarithm

2.6 ZKP: Le Applicazioni

Capitolo 3

zkSnark: *zero knowledge Succint Non-Interactive Argument of Knowledge*



Questo modello vede l'implementazione di una tipologia di Zero Knowledge *Non-Interactive* e si basa sulla teoria dei grafi detta *SNARK (Succint Non Interactive Argument Of Knowledge)*. Uno snark è un grafo cubico connesso, privo di ponti, con indice cromatico uguale a 4.

In altre parole, uno **snark** è un grafo in cui ogni vertice ha tre nodi vicini, basandosi sulla condizione che gli spigoli non possono essere colorati solo con tre colori senza che due spigoli dello stesso colore si incontrino in un punto.

La filosofia zkSnark venne introdotta nel 2012 in un articolo pubblicato da *Bitanksy Nir*. Una prima implementazione fu integrata nel protocollo ***Zero-cash blockchain***, divenendo la colonna portante del lavoro computazionale svolto per ottenere una validazione sull'aggiunta di blocchi, introducendo la possibilità ad un certo party di creare e gestire dei **mathematical proofs** per dimostrare di possedere o meno un certo tipo di informazione, senza rinunciare alla sua integrità.

3.1 zkSnark: La struttura

Essendo un modello Non-Interactive, l'interazione tra Prover e Verifier viene gestita da un terzo ente fidato ad entrambi: un *setup*, che mette a disposizione del protocollo circuiti e software aggiuntivi. Nel caso di zkSnark quindi, il modello vede la presenza di tre enti in comunicazione asincrona: un **Prover P**, un **Verifier V** ed un **Setup S**.

Viene introdotto l'utilizzo di una coppia di chiavi, dette **Proving Key [PK]** e **Validation Key [VK]**, necessarie al Prover ed al Verifier per poter portare a termine la loro comunicazione/validazione:

Proving Key, PK : utilizzata da P per computare un *proof* π verificabile.

Validation Key, VK : utilizzata da V per verificare un *proof* π generato da P attraverso PK.

Queste chiavi sono generate e distribuite da S attraverso un algoritmo di *Generazione KG*, il quale prende in ingresso due parametri: un valore predefinito di sicurezza λ ed un *F-arithmetic circuit C*.

Il modello definisce **tre algoritmi indipendenti**, destinati alle entità del protocollo:

+ **Key Generator KG [Setup]**: $KG(\lambda, C)$
+ **Proof Generator PG [Prover]**: $PG(PK, x, W)$
+ **Proof Validator PV [Verifier]**: $PV(VK, x, \pi)$

Legenda:

λ: *parametro di sicurezza*
C: *circuito aritmetico con input ed output* $\in$ *campo* F
PK: *proving key,* $PK \in F$
VK: *validation key,* $VK \in F$
x: *input pubblico di P, hashed value* $x \in F^n$
W: *input segreto di P, witness value* $w \in F^h$
π: *proof generato da P, proof value* $\pi \in F^h$

Di seguito uno schema di implementazione del modello zkSnark.

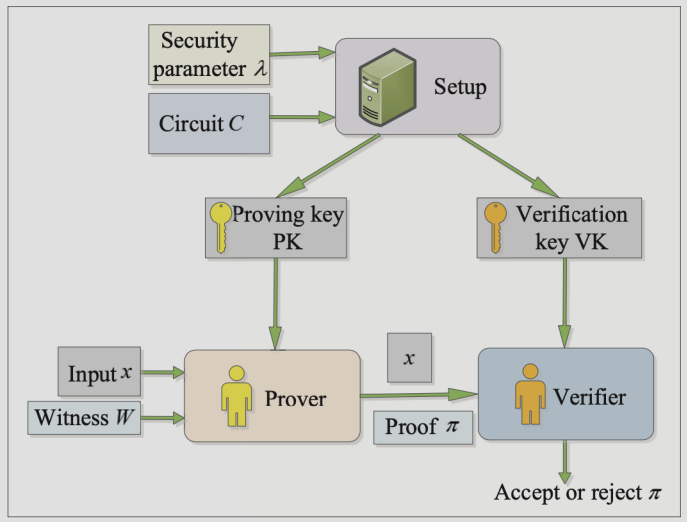


Figura 3.1: zkSnark schema

3.2 zkSnark: I ruoli delle entità

Data la natura del modello, è importante scindere i ruoli delle tre entità in gioco ma, soprattutto, definire ed analizzare gli algoritmi che verranno eseguiti ad ogni livello della comunicazione.

Setup S : il compito di S è quello di generare una coppia unica di chiavi *(PK,VK)* da distribuire ad un P ed un V, in modo che possano comunicare tra loro per eseguire una verifica a Zero Knowledge. Egli esegue l'algoritmo di Key Generator $KG(\lambda, C)$, a partire da un parametro λ sicuro e sconosciuto sia a P che a V. Quindi:

$$(\text{PK, VK}) = KG(\lambda, C)$$

con *C F-arithmetic circuit*; PK, VK $\in F$; *F è un Field/Campo.*

L'introduzione di questo ente terzo e fidato è fondamentale: permette di tenere privato il parametro di sicurezza λ; di fatti, tramite questo, vi è

la possibilità di eseguire l'algoritmo KG per la generazione di chiavi. Se λ fosse conosciuto da P od S, sarebbero introdotte delle problematiche di integrità della comunicazione stessa. Allora si possono analizzare due situazioni differenti che descrivono questa falla di sistema, risolta dall'affidamento ad S dell'esecuzione dell'algoritmo KG:

V esegue KG : se V ha il compito di eseguire KG, avrà anche il compito di scegliere casualmente il parametro di sicurezza λ per creare la coppia di chiavi. In questo schema il problema si focalizza sul mantenimento dell'integrità di λ da parte di V: se P fosse in grado di conoscere od ottenere λ usato da V per la comunicazione instaurata, sarebbe in grado lui stesso di generare fake proofs.

P esegue KG : al contrario, se un Prover P avesse il compito di generare una coppia di chiavi per la comunicazione, un verifier V onesto non potrebbe mai accettare da P nessun π *proof* ricevuto, dato che è P a scegliere il parametro λ, avendo anche la possibilità di cambiarlo quando vuole per generare proofs maliziosi.

La soluzione vive quindi nel mezzo: ***"il generatore di chiavi KG viene affidato ad un entità/gruppo fidato sia per P che per V. In questo modo il parametro λ sarà nascosto ad entrambi, rimanendo privato e conosciuto solo al third party. Quest'ultimo avrà quindi il compito di generare e distribuire a P e V una coppia di chiavi (PK, VK), da utilizzare appositamente per la verifica di certo un segreto W partendo da un circuito aritmetico prestabilito C."***

Prover P : partendo da una chiave PK ricevuta da S, genera un **proof** π, tale che:

$$\pi = PG(PK, x, W)$$

con $PK \in F, \quad x \in F^n, \quad W \in F^h$.
Questo proof generato viene inviato a V e dimostra che P consce un *witness/segreto* W e che questo witness soddisfa il *circuito aritmetico* C conosciuto. Di fatti, il circuito aritmetico C definisce un certo

programma, tale che:

$$C(x, W) = 0^l$$

Il risultato di C, può essere visto come un valore *booleano* che descrive la validità di un certo *segreto* W attraverso il circuito aritmetico stesso computando un **messaggio** in uscita dallo stesso. Questa nozione è fondamentale a livello del Verifier V.

Verifier V : riceve un certo **proof** π da P e, partendo dalla chiave VK ricevuta da S, computa la verifica di π ricevuto tramite l'algoritmo PV, tale che:

$$PV(VK, x, \pi) = True/False$$

allora, $PV(VK, x, \pi) == (\exists W \mid C(x, W))$

3.3 zkSnark: La Complessità

Conoscendo la definizione delle proprietà fondamentali dei protocollo a Zero Knowledge, è intuibile come zkSNARK verifichi con facilità ognuna di queste: completeness, solidity e zero-knowledge. Questo protocollo non solo si distingue dagli altri per la necessità di dover coinvolgere un terzo ente nella dimostrazione, bensì introduce una nuova proprietà che ridefinisce lo standard dei zero knowledge proofs: la *sinteticità* o **succinctness**. Essa descrive principalmente le caratteristiche del modello a livello di spesa computazionale; per parlare quindi della sinteticità di zkSNARK si deve fare un passo indietro e capire di cosa tratta l'insiemistica di problemi che questo protocollo utilizza: si parla della classe di problemi *NP*.

3.3.1 Non Deterministic Polynomial Problems

Un problema di ottimizzazione R fa parte della classe *Non Deterministic Polynomial (NP)* quando nota la sua soluzione ottima, il valore ottimo del problema può essere calcolato in un tempo polinomiale. Per descrivere il

lato "non deterministico" di questa classe di problemi, viene particolarmente utile la definizione rivista utilizzando il concetto di *macchine di turing*:

> Un problema di ottimizzazione appartiene alla classe di complessità NP quando le sue soluzioni ottime possono essere verificate in un tempo polinomiale con una macchina di Turing *non deterministica*, ossia che in presenza di un determinato stato e un determinato carattere letto, la macchina permette più operazioni/transizioni in contemporanea.

Questa rappresenta una categoria molto ampia di problemi di ottimizzazione: non importa con che complessità un algoritmo ci porti una soluzione ottima del problema, basta che il valore ottimo ottenuto a partire da questa sia trovato in un tempo polinomiale. Diverso è invece per i problemi della classe *Polynomial (P)*, i quali richiedono che esista un algoritmo A che risolva il problema di ottimizzazione in un tempo strettamente polinomiale. E' comunque importante evidenziare una relazione tra queste due classi di problemi: $P \in NP$. Se un problema appartiene alla classe P, egli farà parte anche della classe NP; non vale il viceversa. Quando si parla, generalmente, di Zero Knowledge Proof si sta analizzando una primitiva crittografica che si basa sulla compitazione di un *NP Statement*, ossia un'istanza di un certo problema NP. La maggior parte dei protocolli a Zero Knowledge sono quindi costruiti sulla classe di problemi NP, mettendo a disposizione un'ampia varietà di algoritmi tramite i quali si possa sviluppare un modello in base alle proprie esigenze. Seguendo questa introduzione, si può dire che:

> un Prover P intende dimostrare ad un Verifier V che è in possesso di una valida soluzione W di un qualche *problema NP pubblico*, ma senza rivelarne la soluzione stessa. Essa viene dimostrata grazie alla computazione, da parte di P, di uno statement NP, il quale verrà poi verificato in tutta sicurezza da V.

3.3.2 Succinctness

La sintenticità, *succinctness*, richiede che il proof utilizzato dal modello sia **compatto** e **facilmente** verificabile. Questa proprietà risulta fondamentale ogniqualvolta un modello preveda un tipo di comunicazione **costosa** in termini di risorse, oppure quando il Verifier risulta debole a livello di risorse di computazione. Per i motivi citati nella sezione precedente, vi è particolare interesse nel utilizzare problemi della classe NP per modelli che prevedano l'implementazione della proprietà di sinteticità. Questo anche perché la classe NP risulta contenere la maggior parte di problemi reali e di ottimizzazione, tali:

- Boolean Satisfaction (NP-complete $\subset$ NP)
- Polynomial Time on a Deterministic Turing Machine (**sorting**) : la classe NP contiene problemi che sono infatti risolvibili su una macchina di Turing (non deterministica) in tempo esponenziale o più veloce se possibile.

Allora quando si dice che

> un NP-Statement può essere provato in Zero Knowledge

vuol dire che una qualsiasi istanza di un problema NP può essere verificata in Zero Knowledge. **Gli zkSNARKs sono un esempio di NP-Statement**.

zkSnark Complexity Requirements	
Prover Algorithm	$O(nlogn)$
Verifier Algorithm	$O(1)$
Proof Size / Communication complexity	$O(1)$
Implementation Technique	Quadratic Arithmetic Programs