Zero Knowledge Proof
Fundamentals and Applications

Nicolò Zarulli
matricola 296235

a.a. 2021-2022 - Università di Parma

Abstract

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.

1 Zero Knowledge Proof

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. 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 asincrona detta *Non Interactive*.

1.1 ZKP: La Struttura

Come ogni protocollo richiede, devono essere stabiliti dei parametri e delle regole che le entità in comunicazione devono seguire e rispettare. Un generico protocollo a Zero Knowledge può essere visto come un semplice algoritmo che vede l'implementazione di 3 fasi sequenziali. Queste sono necessarie per stabilire un terreno di *gioco* equo e coerente sia rispetto al Prover che rispetto al Verifier.

Per introdurre queste fasi, bisogna prima capire quando è utile implementare un protocollo ZKP:

- 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.

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. Grazie alla filosofia della dimostrazione a Zero Knowledge, un certo 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 entrambe le entità dovranno seguire un algoritmo, costituito da tre passi:

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. Queste permettono al Verifier di capire se il Prover sia onesto o malizioso.
3. **Response Phase**: Il Prover riceve le *domande* dal Verifier e per ognuna di queste ne formula una risposta. Il Verifier, nel complesso, valuta le risposte per prendere una decisione definitiva, accettando o rifiutando il Proof ricevuto nella Witness Phase.

E' importante evidenziare delle 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, mentre 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 computazionale delle entità potrebbe essere oneroso, dipende dai meccanismi usati per implementare le fasi.
- Nelle fasi descritte **nessun tipo di informazione privata verrà condivisa**.

1.1.1 Un Esempio Astratto

Per comprendere al meglio questo tipo di approccio crittografico, è molto efficace presentare al lettore una storia molto conosciuta che evidenzia con gran chiarezza la 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 potrà 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%).

1.2 ZKP: Le Proprietà

La nozione di *Zero Knowledge* fu introdotta negli anni '80 da un gruppo di ricercatori del MIT (*Goldwasser, Micali, Rackoff*) e descrive in modo molto più approfondito i concetti brevemente descritti nelle sezioni precedenti. Attraverso un **Interactive Proof System** viene definito un modello dove un Provider scambia dei messaggi con un Verifier per convincerlo che un certo *Mathematical Statement* in suo possesso sia vero. Un aspetto molto importante, evidenziato dai ricercatori, riguarda una proprietà di integrità sullo scambio stesso dei messaggi tra le due entitò comunicanti. E' possibile che un Prover malizioso tenti di ingannare un Verifier onesto nel credere ad uno statement falso. In

questo modello difatti il Prover è un entità in una posizione avvantaggiata, *ha tutto da guadagnare e nulla da perdere*. 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 l'attenzione anche verso una situazione 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**: si parla del comprendere di quanto dettaglio (informazioni extra) ha effettivamente bisogno il Verifier durante la fase di valutazione di un Proof ricevuto. Entrambi i lati del protocollo (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 all'approccio crittografico 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.

1.2.1 Un Esempio 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 uno **statement personale v**

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

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

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

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.

Risultati

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 che un Verifier onesto sono protetti da attacchi maliziosi.

1.3 ZKP: Classificazione

Gli esempi di Zero Knowledge Proof illustrati nelle sezioni precedenti richiedono l'interazione diretta tra le due entità comunicanti, attraverso lo schema di *domande & risposte*. Ne risalta un'importante parametro: l'**interazione**. Tramite questo si può evidenziare una classificazione dei protocolli a Zero Knowledge:

- **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.

1.3.1 Un piccolo approfondimento: Interactive-ZKP

Agli albori dello sviluppo dei Zero Knowledge Proof, il modello più diffuso era di tipo *Interactive-ZKP*, vedendone un implementazione soprattutto nei sistemi informatici, basandosi su di un approccio *matematico* piuttosto che astratto ed applicativo. In questo caso *ZKP* viene utilizzato per dimostrare a qualcuno la conoscenza di un fatto *matematico*, senza rivelarne però nessun informazione sensibile dello stesso. In tempi recenti 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 la tipologia *Non Interactive ZKP*. E' comunque importante evidenziare certi casi nei quali è consigliato un approccio interattivo:

- **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**.
- **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**

1.4 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*.

1.4.1 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

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.

1.4.2 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

1.4.3 Bulletproofs

2 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 ***Zerocash 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.

2.1 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* $\pi \in F^h$

2.2 zkSNARK: Le 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, $\quad PV(VK, x, \pi) == (\exists W \mid C(x, W))$

2.3 zkSnark: La Complessità