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 Uno sguardo alla Dimostrazione a Conoscenza Zero

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 La struttura del protocollo Zero Knowledge

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 queste *regole* un generico protocollo ZKP si può vedere 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 dimostrazione, dove il Prover riesce a convincere il Verifier di essere in possesso di un *Witness* grazie ad un *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, in base al 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 un pagamento 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 che lo pagherà per il segreto, ma non prima di essere sicura che Bob 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%).

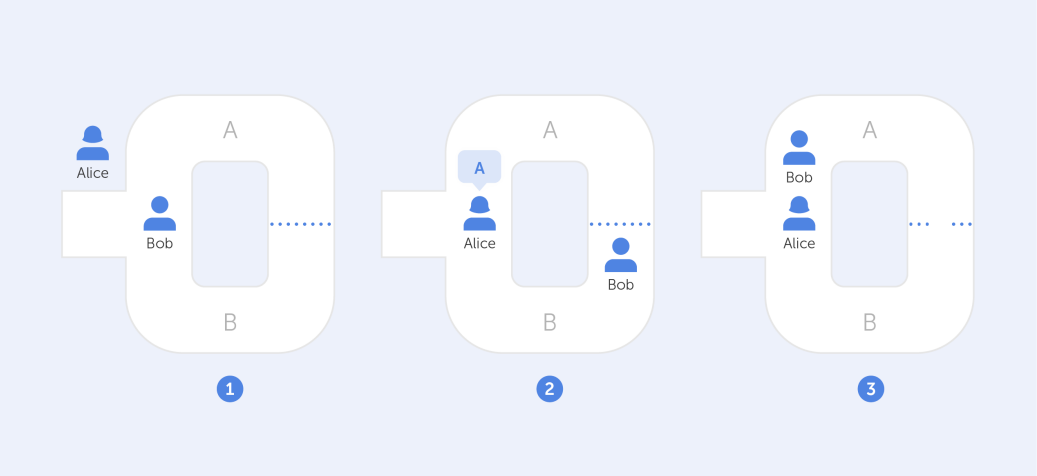


Figura 2.2: ZKP, un esempio attraverso una storia astratta

2.3 Le proprietà di un Zero Knowledge Proof

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.
   Inoltre 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 Classificazione dei Zero Knowledge Proof

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 si questa si può evidenziare una 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.

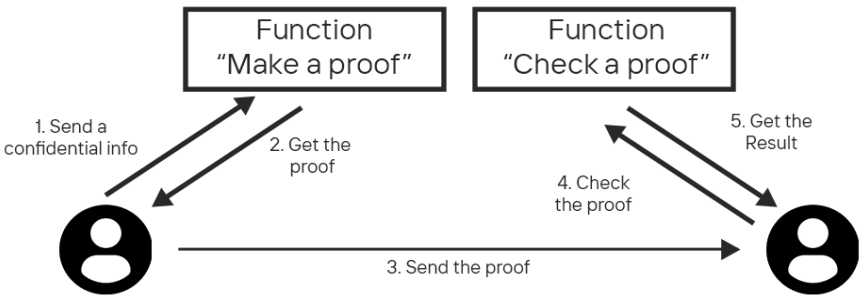


Figura 2.3: Non-Interactive ZKP

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 *Zwero 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 *blockchains*. 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 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**.

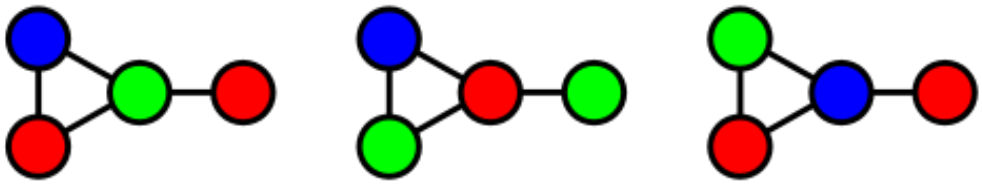


Figura 2.4: varie istanze three-coloring di un certo grafo pubblico

- **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 Modelli Zero Knowledge Proof

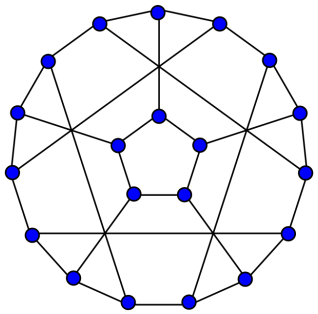
L'approccio della Dimostrazione a Conoscenza Zero trova grande applicazione nell'era del digitale e delle comunicazioni Internet ma, soprattutto, nella categoria dei *Sistemi Distribuiti* con particolare riferimento alle *Blockchains*.

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, ritrovando ad esempio:

- zkSNARK
- STARKS
- Ben-Sasson's Model
- Ligero
- Bulletproofs
- Hyrax
- Aurora
- Libra

Questi modelli possono far parte della filosofia *Interactive ZKP* o della *Non-Interactive ZKP*.

2.5.1 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 l'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.

La struttura di zkSnark

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**.

Ciò che questo protocollo introduce è l'utilizzo di una coppia di chiavi, dette **Proving Key [PK]** e **Validation Key [VK]**, necessarie al Prover ed al Verifier per poter portare con successo 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* G, 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**: $KG(\lambda, C)$
+ **Proof Generator PG**: $PG(PK, x, W)$
+ **Proof Validator PV**: $PV(VK, x, \pi)$

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* $\in F^h$

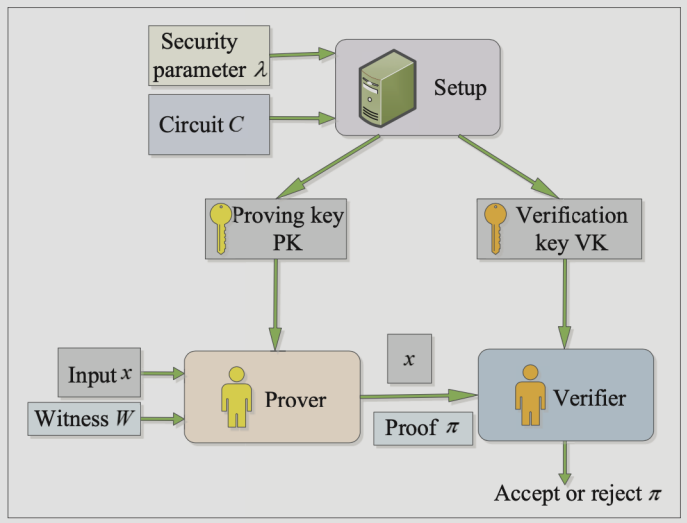


Figura 2.5: zkSnark schema.