Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

I protocolli Zero-Knowledge

Ambito riguardante:
Crittografia

Relatore:
Prof. **Luciano Margara**

Presentata da:
Andrea Farneti

Sessione III
Anno Accademico 2020-2021

"Gli uomini non hanno più tempo per conoscere nulla.
Comperano dai mercati le cose già fatte."
Il Piccolo Principe, Antoine de Saint-Exupéry

Prefazione

Origine e obiettivi della tesi

I protocolli Zero-Knowledge rappresentano un argomento complesso e diramato, tutt'ora oggetto di studio ed in continua evoluzione. Essi spaziano in innumerevoli ambiti aperti di ricerca e di applicazione pratica all'interno dell'intero ecosistema di Internet, motivi che rendono la loro standardizzazione un arduo compito rispetto ad altri tipi di protocolli.
La seguente tesi nasce dall'interesse di voler riassumere un argomento così ampio e vario con un unico filo conduttore, per tanto essa si pone l'obbiettivo di fornire una descrizione semplice, chiara ma allo stesso tempo esaustiva di tali protocolli, esposti da un punto di vista principalmente crittografico, con integrazioni in ambito algoritmico e matematico.

Struttura della tesi

La tesi viene presentata in tre capitoli di discussione che suddividono l'ambito da un punto di vista logico-temporale, per poi presentare le conclusioni riguardo le aspettative future di tali protocolli. Saranno argomento di tesi storia, nascita e relativi problemi dei protocolli Zero-Knowledge sino a giungere allo stato dell'arte come argomento fulcro di essa. Si terminerà con un'esposizione dei principali ambiti futuri di ricerca e applicazione.

Indice

Sinonimi e abbreviazioni

Affermazione *x*, *statement*, stringa, parola.

Argument *computationally sound proof*, argomento.
Si contrappone al termine "proof".

Argument of Knowledge *AoK*, argomento di conoscenza.
Si contrappone al termine "Proof of Knowledge".

Common Reference String *CRS*, stringa corta casuale comune.

Eavesdropping *Eve*, origliare, intercettare.

Non-interactive Zero-Knowledge *NIZK*, conoscenza-zero non-interattiva.

Proof prova, dimostrazione.

Proof of Knowledge *PoK*, prova di conoscenza.

Sistema a dimostrazione interattiva *interactive proof system*, sistema a prova interattiva.

Witness *w*, testimonianza, segreto.

Zero-Knowledge protocol *ZKP*, *Zero-Knowledge proof.*
Protocolli, prove o dimostrazioni a conoscenza-zero.

Capitolo 1

Introduzione

1.1 Un cenno di storia

Il primo concetto di "conoscenza-zero" nasce con l'introduzione dei sistemi a dimostrazione interattiva e della loro relativa classe di complessità IP *(Interactive Polynomial time)*, argomenti definiti da Shafi Goldwasser, Charles Rackoff e Silvio Micali nel 1985 in una bozza di *"The knowledge complexity of interactive proof-systems"* [1].

1.1.1 I sistemi a dimostrazione interattiva

Un sistema a dimostrazione interattiva è un automa[1] che modella e definisce lo scambio di messaggi fra due parti: *Prover* e *Verifier*.
Il Verifier è una macchina di Turing[2] probabilistica onesta e con capacità computazionali deboli (limitate ad una velocità polinomiale) il cui obbietti-

[1]Un automa è un modello di un sistema (hardware o software) che consiste in azioni di input, output e dal set di operazioni che possono essere eseguite. La macchina di Turing è un classico esempio di automa.

[2]Una macchina di Turing è un automa costituito da un sistema di regole ben definite. In altre parole, è un modello astratto che definisce una macchina in grado di eseguire algoritmi, il cui scopo è valutare i linguaggi e risolvere funzioni matematiche. Viene utilizzata per studiare i limiti del calcolo meccanico.

vo è quello di verificare se una determinata affermazione x nota ad entrambe le parti (detta anche stringa o parola), solitamente matematica, è valida. Un'affermazione è valida se si può dimostrare in tempo polinomiale tramite una prova (*proof*) che essa risulta appartenente ad un linguaggio L.
Il Prover è, invece, una macchina con infinito potere computazionale il cui obbiettivo è riuscire a convincere il Verifier, tramite le sue risposte, che l'input x appartiene al linguaggio. Egli è capace di risolvere determinati tipi di problemi, ad esempio problemi decisionali o problemi di funzione:

- Un problema decisionale è rappresentato come un set di numeri naturali o stringhe. Un'istanza di tale problema è un numero naturale o una stringa. La soluzione di tale istanza è "si" (oppure "1") se il numero è contenuto all'interno del set, o "no" (oppure "0") altrimenti.

- Un problema di funzione è rappresentato da una funzione f ($\mathbb{N} \to \mathbb{N}$). Un'istanza del problema è un input x di f e la soluzione è il valore $f(x)$.

Il Prover potrebbe, però, agire in disonestà, per cui il Verifier deve condurre un "interrogatorio" ponendogli una serie di challenge in successione, con l'obbiettivo di verificare se egli dispone di una qualche informazione segreta (detta comunemente *witness*, testimonianza) che gli permette di comprovare la veridicità di tale affermazione. In questo modo il Verifier può asserire, con un certo grado di sicurezza, che l'input appartiene al linguaggio e quindi il Prover è onesto.
Qualsiasi sistema a dimostrazione interattiva deve soddisfare specifici requisiti:

- **Completezza** *(Completeness)*: se un'affermazione è vera, un Verifier onesto ne accetta sempre la dimostrazione. Si parla in questo caso di *Perfect Completeness*. Se è invece presente un certo margine di errore dovuto, ad esempio, ad algoritmi probabilistici, si parla di normale completezza.

- **Correttezza** *(Soundness)*: la probabilità che un Prover non onesto riesca a convincere il Verifier è molto bassa se egli non conosce la witness. Se si dimostra che è esattamente zero si parla di *Perfect Soundness*.

Dato che i problemi facenti parte della classe di complessità IP richiedono Verifier probabilistici, vale a dire il cui comportamento varia ad ogni loro nuova interazione tramite un numero random, l'utilizzo di tali algoritmi introduce nei sistemi a dimostrazione interattiva una certa probabilità di errore ε, ma prima di esporne il concetto nella loro classe di complessità, viene introdotto l'argomento in uno scenario più semplice, ovvero quello deterministico, all'interno della classe di complessità NP *(Non-deterministic Polynomial time)*.

I sistemi a dimostrazione interattiva in NP

La classe NP può essere considerata un caso triviale di sistema a dimostrazione interattiva in cui il Verifier viene ulteriormente limitato ad essere una macchina di Turing deterministica (cioè un sistema in cui non si ha casualità ed il cui comportamento è determinato dall'input ricevuto, producendo sempre lo stesso output a parità di stato e condizioni iniziali) ed in cui la procedura di dimostrazione è ristretta ad un solo round (ovvero, il Prover invia solamente una singola witness al Verifier, alla quale ci si riferisce tipicamente col termine di *certificato*).
Per questa ragione, NP è anche chiamato dIP (*deterministic Interactive Proof*), anche se questo nome viene raramente utilizzato.

I requisiti di completezza e correttezza vengono rispettati:

- **Completezza**: $x \in L \rightarrow$ la probabilità che V accetti w di P è 1.
- **Correttezza**: $x \notin L \rightarrow$ la probabilità che V non accetti w di P è 1.

La classe NP viene utilizzata per la classificazione dei problemi decisionali, quei problemi precedentemente definiti come aventi solamente due possibili output per ogni input: "si" o "no" (o, alternativamente, "1" o "0").
Un esempio di problema decisionale è il seguente:

Esempio 1. Dato in input un grafo arbitrario, stabilire se esso risulta essere connesso o meno.

Il linguaggio di questo problema, ovvero tutte le risposte considerate valide, sarà quindi il set di tutti i grafi connessi.

In particolare, rientrano nella definizione di NP tutti i problemi per cui, quando la loro istanza ha risposta "si", essi hanno witness *verificabile* in tempo polinomiale da una macchina di Turing deterministica o, alternativamente, tutti quelli che possono essere *risolti* in tempo polinomiale da una macchina di Turing non-deterministica.
E' opportuno notare che il concetto di "non-determinismo" include quello di "probabilismo" visto precedentemente: esso è infatti il macroinsieme contenente gli algoritmi il cui comportamento risulta essere imprevedibile ad ogni interazione (e.g. un algoritmo parallelo può performare differentemente in diverse esecuzioni a causa di una *race condition*[3]).

Nella computazione non-deterministica la macchina non prosegue unicamente dallo stato in cui essa si trova ad un altro, bensì ogni step può contenere multiple strade che generano un albero di possibilità differenti, ma la differenza fra i vari casi di non-determinismo è data da come viene interpretato l'albero: quando si parla di probabilismo l'albero è quasi sempre un *Binary Tree*[4], in cui ogni ramo è considerato come equamente probabile (come nel

[3]Si parla di race condition quando, in un sistema basato su processi multipli, il risultato finale dell'esecuzione di una serie di processi dipende dalla temporizzazione o dalla sequenza con cui essi vengono eseguiti.

[4]Un albero binario è un albero i cui nodi hanno grado compreso tra 0 e 2, ovvero in cui ogni nodo può avere al più due figli.

caso del lancio di una moneta, ad ogni lancio si ha probabilità 1/2 che esca testa o croce).
Il comportamento di una macchina non-deterministica differisce, inoltre, anche dalla classe di complessità in cui opera: nel caso di NP si osserva direttamente se l'albero contiene un nodo foglia (che rappresenta il termine del calcolo) che accetta l'input. In altri casi di complessità come IP e AM (Arthur-Merlin), che verranno discussi in seguito, si è interessati alla probabilità di raggiungere un determinato stato di accettazione rispetto che uno di rifiuto.

I sistemi a dimostrazione interattiva in IP

Si sono introdotti i sistemi a dimostrazione interattiva osservando un caso speciale, ma per poter comprendere il vero potenziale che si ottiene dall'interazione bisogna lasciare che il Verifier sia probabilistico: in questa classe di problemi Prover e Verifier comunicano tra di loro un numero polinomiale di volte, e quest'ultimo pone l'interrogatorio al Prover randomizzando le domande tramite l'utilizzo di una "moneta virtuale"[5] il cui valore è mantenuto privato. Come già accennato in precedenza, però, con il passaggio ad un sistema probabilistico si introduce anche una certa probabilità di errore; in particolare, dopo lo scambio di un numero polinomiale di messaggi, il Verifier dovrà scegliere se l'input appartiene al linguaggio o meno, con una probabilità di errore massimo uguale a 1/3:

- **Completezza**: $x \in L \rightarrow$ la probabilità che V accetti dopo aver comunicato con P è $\geq 2/3$.
- **Correttezza**: $x \notin L \rightarrow$ la probabilità che V accetti dopo aver comunicato con P è $\leq 1/3$.

[5]Una moneta virtuale è un generatore di numeri casuale che può assumere valori 0 o 1.

Un esempio pratico di problema probabilistico interattivo è il seguente:

Esempio 2. Peggy (Prover) dichiara a Victor (Verifier) di essere in grado di distinguere il gusto della Coca-Cola e della Pepsi tra di loro. Per verificare questa affermazione, decidono di ripetere un esperimento 50 volte: Peggy si gira, e Victor sceglie randomicamente un bicchiere da porgere a Peggy contenente uno dei due drink. Se Peggy è in grado di rispondere correttamente a tutte le prove randomiche di Victor, egli può dichiarare con un certo grado di sicurezza che l'affermazione di Peggy è veritiera.

Si nota così che la combinazione di tecniche di interazione e randomizzazione porta a grandi risultati: esse permettono di passare da NP a IP e, dato che IP = PSPACE [2], ovvero la classe di problemi risolvibili da una macchina di Turing deterministica in spazio polinomiale, ogni problema che può essere risolto in tempo polinomiale da un sistema a prove interattive può essere risolto anche in questo modo (e viceversa).

1.1.2 I giochi di Arthur-Merlin

I giochi di Arthur-Merlin, introdotti da Babai nel 1985 [3], sono un sistema a dimostrazione interattiva in cui il Verifier (Arthur) è un computer probabilistico di velocità polinomiale che dispone di un generatore di numeri casuale, mentre il Prover (Merlin) è un oracolo con infinito potere computazionale.

La classe di complessità AM

Un'altra classe di complessità strettamente collegata ai sistemi a dimostrazione interattiva è la classe AM (Arthur-Merlin). Essa è molto simile alla classe IP vista precedentemente, difatti le due sono state introdotte a distanza molto ravvicinata tra loro.
Per darne una definizione è sufficiente infatti specificare che AM è una restrizione di IP ad un numero di interazioni costante (e non polinomiale).
Esiste però un'altra differenza: in IP il Verifier è una macchina probabilistica il cui valore delle monete utilizzate per la randomizzazione è tenuto nascosto

al Prover. Egli può vedere solamente i messaggi che il Verifier produce tramite esse: questo concetto è chiamato *Private Coin*.
In AM, invece, le monete sono mantenute pubbliche (*Public Coin*), di conseguenza il Prover ha accesso a tutte le scelte randomiche effettuate dal Verifier. Inizialmente si pensava questo potesse costituire un punto di forza per IP, ma successivamente è stato dimostrato da Goldwasser e Sipser che la capacità di un Verifier di mantenere nascoste le monete non influisce più di tanto sulla sicurezza dei protocolli, in quanto il Prover di un protocollo Arthur-Merlin a monete pubbliche può, grazie alla sua potenza di calcolo e con sole due interazioni in più, giungere alle stesse conclusioni di un protocollo a moneta privata [4]. Formalmente, $\mathrm{IP}[k] \subseteq \mathrm{AM}[k+2]$.

La comunicazione in AM è definita come AM[k], dove k è una costante che rappresenta il numero di messaggi scambiati. In particolare, di default AM = AM[2], perché qualsiasi AM[k] con $k > 2$ può essere generalizzata in AM[2]: Arthur effettua il lancio di diverse monete ed invia il risultato di *tutti* i suoi lanci a Merlin, egli risponde con una presunta prova e Arthur, infine, effettua la verifica [5].
Se $k \geq 2$ e pari, nelle applicazioni pratiche del protocollo la comunicazione inizia sempre da Arthur, altrimenti da Merlin: questo perché l'ultimo messaggio dovrebbe sempre essere di Merlin, dato che Arthur non necessita di inviare un messaggio a Merlin dopo che egli ha già deciso.
Ad Arthur è concesso di rispondere a Merlin solamente con i risultati dei lanci di moneta.

La classe di complessità MA

La classe di complessità MA rappresenta il caso particolare (nonché triviale) di comunicazione in cui viene inviato un solo messaggio: Merlin invia un certificato ad Arthur, ed egli sceglie se accettarlo o meno provando ad eseguire un calcolo probabilistico in tempo polinomiale.

Teorema 1. *MA è contenuta in AM: Questo è facilmente dimostrabile, in quanto il caso di AM[3] contiene il caso MA. Merlin invia un certificato ad Arthur ed egli può, dopo averlo ricevuto, effettuare il lancio del numero di monete richiesto, inviarle a Merlin e, successivamente, ignorare la sua risposta.*

Teorema 2. *MA contiene NP: in NP non è definito il concetto di moneta pubblica/privata, ma dato che è Merlin ad avviare la comunicazione ad una via, egli non ha bisogno del lancio di monete di Arthur, per cui Merlin deve solamente inviare ad Arthur un certificato che egli possa validare deterministicamente in tempo polinomiale.*

1.2 Le prove di conoscenza

Il concetto di "prova di conoscenza" non è strettamente necessario all'introduzione della conoscenza-zero, difatti la possibilità di una non esclude l'altra (un sistema a dimostrazione interattiva può essere sia a conoscenza-zero che una prova di conoscenza), ma lo diventa se la si vuole analizzare in ogni suo aspetto.
Una prova di conoscenza (o *"Proof of Knowledge"*) è un sistema a dimostrazione interattiva nel quale un Prover riesce a convincere un Verifier che egli *conosce* qualcosa: questo segreto è già stato definito come witness all'interno dei sistemi a dimostrazione interattiva, ma il possederla e il dimostrare di conoscerla sono, in ambito crittografico, due concetti totalmente diversi.
Quando si parla di sistemi a dimostrazione interattiva in generale, per come essi sono stati definiti, il fine per un Prover non è necessariamente dimostrare di conoscere la witness, quanto riuscire a dimostrare che una parola appartiene al linguaggio, ma il primo è un requisito molto più potente: qui nasce la differenza fra un sistema a dimostrazione interattiva e una prova di conoscenza; E' possibile che un Prover riesca a dimostrare che "x è contenuto nel linguaggio", ovvero riuscire ad inviare i messaggi appropriati in un sistema

a dimostrazione interattiva probabilistico, senza che egli ne conosca la prova statica e deterministica (i.e. la witness).
Non solo, spesso accade anche che il linguaggio è triviale, ovvero ogni parola appartiene al linguaggio ma, allo stesso tempo, non è banale conoscere la witness per una determinata parola trovata.
A livello pratico, si osserva questo concetto con un esempio (5), ma per poterlo analizzare è necessario introdurre prima la nozione algebrica di "gruppo".

I Gruppi

Definiamo un gruppo come una struttura algebrica formata dall'abbinamento di un insieme non vuoto di elementi con un'operazione binaria interna che rispetta l'assioma di associatività (ad esempio la somma ed il prodotto) e gli assiomi di esistenza dell'elemento neutro e di esistenza dell'inverso di ogni elemento.
Definiamo un gruppo ciclico G come un gruppo che può essere generato da un unico elemento g [6], detto generatore (o radice primitiva), tale che ogni elemento di G è l'insieme delle potenze di g ad esponente intero se usiamo la notazione moltiplicativa (formalmente, $G = \{g^n : n \in \mathbb{Z}\}$), oppure i multipli di g se si utilizza la notazione additiva (formalmente, $G = \{gn : n \in \mathbb{Z}\}$).
In altre parole, un gruppo G è ciclico se esso contiene un elemento g il cui sottogruppo generato da esso (che si indica solitamente con $\langle g \rangle$) genera nuovamente G.
Definiamo l'ordine di un gruppo ciclico come la cardinalità |G|: se essa è finita si parla di gruppo ciclico finito, altrimenti di gruppo ciclico infinito. Se G è ciclico, allora G e $\langle g \rangle$ hanno lo stesso ordine.
L'elemento identità di un gruppo è quell'elemento e t.c. $e \cdot a = a$, dove $\cdot$ è l'operazione binaria del gruppo. Nel caso di un gruppo additivo, l'identità è 0, nel caso di un gruppo moltiplicativo è 1.
Di seguito viene fornito un esempio di gruppo ciclico con operazione additiva in modulo (3), e uno con notazione moltiplicativa (4):

Esempio 3. dato il gruppo $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ e definita la sua operazione additiva in modulo 6, otteniamo:

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Tabella 1.1: valori dell'operazione additiva in mod 6 di $\mathbb{Z}_6$

Ora, considerando i possibili sottogruppi di $\mathbb{Z}_6$:

$\langle 1 \rangle = \{1, 2, 3, 4, 5, 0\}$
$\langle 2 \rangle = \{2, 4, 0\}$
$\langle 3 \rangle = \{3, 0\}$
$\langle 4 \rangle = \{4, 2, 0\}$
$\langle 5 \rangle = \{5, 4, 3, 2, 1, 0\}$
$\langle 0 \rangle = \{0\}$

Si nota che solamente i sottogruppi $\langle 1 \rangle$ e $\langle 5 \rangle$ sono generatori di $\mathbb{Z}_6$: infatti, dato che i loro multipli generano tutto $\mathbb{Z}_6$ ($\mathbb{Z}_6 = \{gn : n \in \mathbb{Z}\}$), esso è per definizione un gruppo ciclico.

Esempio 4. dato il gruppo ciclico moltiplicativo $\mathbb{Z}_{17}$ dei numeri interi relativi t.c. $\mathbb{Z}_{17} = \{1, 2, ..., 16\}$, che ha come ordine il numero primo $p = 17$ (questo perché ogni gruppo che ha come ordine un numero primo è ciclico [7]), si mostra che $g = 3$ è un suo generatore:

$\langle 3 \rangle = \{3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1\}$, dove ogni elemento in k-esima posizione di $\langle 3 \rangle$ (con $k > 0$) è ottenuto dall'operazione: $3^k \bmod 17$.

Esempio 5. Fissato un gruppo ciclico G con generatore g che ha come ordine un numero primo p e considerando il linguaggio $\mathrm{L} = \{x \in G : \exists\, w, x = g^w\}$, questo è un linguaggio triviale; dato che G è ciclico, ogni elemento di $\mathrm{G} \in \mathrm{L}$ e qualsiasi $x \in \langle g \rangle$, di conseguenza la seguente è una valida prova interattiva di appartenenza al linguaggio:

1. il Prover non invia niente: ricordando che $x = g^x$ ed il suo valore è noto e conosciuto da entrambe le parti sin dall'inizio, non è necessaria alcuna informazione aggiuntiva per dimostrare la sola appartenenza al linguaggio; non è necessario conoscere la witness, dato che è possibile controllare direttamente se $x \in \mathrm{G}$.

2. il Verifier controlla se x è un elemento del gruppo G, e se questo è vero allora esso appartiene al linguaggio.

Tuttavia, per w questo non è banale:
in questo caso conoscere una witness tale per cui sia possibile verificare in tempo polinomiale che la relazione $R(x, w)$ per $g^w = x$ ha come output 1, ovvero risulta verificata, è molto più difficile: bisogna dimostrare di conoscere il logaritmo discreto di x in base g. Infatti, se il problema è triviale nel senso opposto, cioè dati g e w è banale calcolare x, non si può dire lo stesso nel caso in cui si dispone di g e x e si vuole trovare w: questo tipo di relazione prende il nome di funzione *one-way*[6], la cui potenza risiede nel tempo necessario ad invertirla; per risolverla e poter trovare la witness è necessario un meccanismo di *"trial and error"*.

[6]Una funzione one-way (o unidirezionale) è una funzione per la quale esiste un algoritmo che calcola $f(x)$ in tempo polinomiale, ma non per calcolarne la controimmagine (a meno di una probabilità trascurabile).

Per valori piccoli di p questo è banale, ma più esso è grande, più sono i tentativi di forza-bruta necessari per trovare una soluzione valida.
In particolare, la potenza di questa tecnica risiede proprio nei gruppi ciclici: essi introducono l'aritmetica modulare, e molti problemi in crittografia diventano computazionalmente difficili da risolvere grazie ad essa. Le radici o i logaritmi in aritmetica di base sono semplici da calcolare sia per valori interi che reali, ma possono diventare un problema difficile se vi si introduce una riduzione in modulo.

A seguito di ciò, risulta necessario introdurre una definizione di correttezza molto più stretta:

- **Correttezza** (o "validità"): la probabilità che un Knowledge Extractor E riesca ad estrarre la witness, avente accesso a oracolo ad un Prover disonesto, deve essere almeno alta quanto la probabilità di successo che ha il Prover di convincere il Verifier.

Questa proprietà assicura che nessun Prover con determinate capacità computazionali che non è a conoscenza della witness possa riuscire a convincere il Verifier.

1.2.1 I Knowledge Extractor

Un Knowledge Extractor è uno speciale tipo di Verifier computazionalmente forte e volutamente non onesto, utilizzato per dimostrare la proprietà di correttezza di un Proof of Knowledge. Egli interagisce con un Prover del quale ha accesso a oracolo: in termini pratici, egli ha a disposizione il suo codice sorgente e può "riavvolgerlo" o effettuare altre operazioni, come modificare i suoi valori casuali. In questo modo, se il Prover completa la prova, il Knowledge Extractor è in grado di estrarre il suo segreto w.
Per verificare la proprietà di correttezza bisogna dimostrare che esiste un

Knowledge Extractor per ogni possibile Prover: questo può sembrare totalmente contraddittorio dato che l'obbiettivo di questi protocolli dovrebbe essere quello di non permettere la rivelazione del segreto, ma il Knowledge Extractor non deve esistere durante la normale esecuzione degli stessi, quanto solamente in contesti di simulazione tramite speciali Verifier.
La dimostrazione avviene utilizzando l'operazione di *"rewinding"*, ovvero tornando indietro nel tempo all'esecuzione del protocollo, durante l'invio del *commitment*[7] da parte del Prover, permettendo così di produrre due esecuzioni valide per challenge diverse: questo processo permette di semplificare il valore random utilizzato nell'equazione e ricavare w.
Nel capitolo successivo viene fornita una dimostrazione pratica di tale operazione con il protocollo di identificazione di Schnorr.

[7]In un sistema fra due parti, il commitment è un valore generato randomicamente da una delle due parti ed inviato all'altra con lo scopo di impegnare sé stessa alla casualità.

Capitolo 2

I protocolli a conoscenza-zero

Nel capitolo precedente sono state discusse le basi su cui è fondato il principio di conoscenza-zero che Shafi Goldwasser, Charles Rackoff e Silvio Micali hanno introdotto nel 1985 [1], ma è solo nel 1991, con *"Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems"* che l'argomento assume più rilevanza, dopo aver dimostrato che per ogni linguaggio appartenente alla classe di complessità NP possono essere definite dimostrazioni a conoscenza-zero, assumendo l'esistenza di funzioni one-way [8].
Anni dopo, nel 2012, gli studi di Shafi Goldwasser e Silvio Micali riguardo l'importanza degli eventi casuali nella crittografia, i quali hanno dato luogo anche alla conoscenza-zero, hanno permesso loro di vincere il Turing Award, il più importante premio scientifico per l'informatica [9].

2.1 La proprietà di conoscenza-zero

Un protocollo a conoscenza-zero (*"Zero-Knowledge Protocol"*, al quale ci si riferisce anche col termine di "dimostrazione" o "prova" a conoscenza-zero) è un sistema a dimostrazione interattiva, e come tale deve rispettare le proprietà di completezza e correttezza osservate nel capitolo precedente.

Ora, un sistema a dimostrazione interattiva, per poter essere definito a conoscenza-zero, deve rispettare una terza proprietà:

- **Conoscenza-Zero** (Zero-Knowledge): l'interazione fra Prover e Verifier rivela solamente la veridicità dell'affermazione e nient'altro.

In altre parole, la witness non viaggia direttamente nel canale di comunicazione per dimostrare la veridicità dell'affermazione x, per cui nessuna macchina disonesta può imparare qualcosa di nuovo intercettando la trasmissione fra P e V.
Si noti, infatti, che questo è diverso dal concetto di firma digitale, in quanto con quest'ultima si vuole testimoniare oltre alla provenienza di un'informazione (equivalente all'autenticazione di un Proof of Knowledge) anche la sua integrità tramite un'impronta che viaggia sulla rete criptata con la chiave privata del mittente (la quale, paragonata al caso di cui si sta trattando, rappresenterebbe la witness), di conseguenza chiunque intercetti la comunicazione può ottenere conoscenza del fatto che P e V hanno comunicato e P è onesto, verificando la firma. Allo stesso modo, un Verifier disonesto può dimostrare a chiunque di aver interagito con P; questo non accade con i protocolli Zero-Knowledge.
Per poter dimostrare che un protocollo rispetta la proprietà di conoscenza-zero è necessario introdurre il concetto di "simulatore".

2.1.1 I simulatori

Un simulatore è uno speciale tipo di Prover computazionalmente forte e volutamente non onesto, che possiede accesso a oracolo al codice del Verifier. A differenza di un Prover reale e onesto, il quale è a conoscenza di un segreto che gli permette di dimostrare la sua autenticità, il simulatore non ha alcun tipo di conoscenza: è proprio questo aspetto che lo rende utilizzabile per dimostrare che un qualsiasi Verifier non impari nulla più che la validità o l'incoerenza di un'affermazione. Perché la dimostrazione abbia successo, il simulatore deve essere in grado di ingannare V facendogli credere che il suo

segreto sia corretto e, nel mentre, produrre una trascrizione della comunicazione fra le parti che risulti "indistinguibile" da quella di un vero Verifier (da qui il nome "simulatore", poiché simula conversazioni fra P e V). A seconda del grado di indistinguibilità della stessa si definiscono tre diverse tipologie di conoscenza-zero:

- Conoscenza-zero **perfetta** (*perfect Zero-Knowledge*): la differenza fra la vista simulata e quella reale è nulla, cioè esse sono completamente indistinguibili. Di conseguenza, nemmeno un Verifier con potenza computazionale illimitata può essere in grado di ricavare qualcosa dalla comunicazione.

- Conoscenza-zero **statistica** (*statistic Zero-Knowledge*): la differenza fra la vista simulata e quella reale è statisticamente indistinguibile, ovvero esse non sono necessariamente uguali ma statisticamente vicine, per cui un Verifier con potenza computazionale illimitata non può ricavare nulla dalla comunicazione eccetto che con probabilità trascurabile.

- Conoscenza-zero **algoritmica** o **computazionale** (*computational Zero-Knowledge*): la differenza fra la vista simulata e quella reale è computazionalmente indistinguibile. Ciò significa che non vi è un algoritmo probabilistico efficiente (in tempo polinomiale all'input dato) capace di distinguere tra le due.

La logica riguardante questo processo segue spontanea: dato che il simulatore non ha "conoscenza" da estrarre sin dall'inizio, nessun Verifier può ottenere alcun quantitativo di informazione significativa dopo aver interagito con esso, il che è verificato dalle trascrizioni prodotte. Inoltre, se la trascrizione prodotta dal simulatore è analoga alla comunicazione con un vero Prover, ne consegue che il Verifier non può fare nulla di meglio contro un Prover onesto di ciò che può contro il simulatore; tramite esso si dimostra quindi che V non può estrarre nessuna informazione utile da una normale interazione con P. Si noti, infatti, che la trascrizione generata dal simulatore non è comprensibile da nessuno se non dal Verifier interessato, che possiede il valore delle

monete randomiche lanciate ad ogni interazione per generare la challenge. Come possa un simulatore riuscire effettivamente ad ingannare V è similare al procedimento osservato per i Knowledge Extractors: tramite il concetto di "rewinding" è possibile ripercorrere i passi generati dall'esecuzione del protocollo e agire di conseguenza per estrapolargli informazioni ed ingannarlo. Per dimostrare, quindi, che un protocollo è a conoscenza-zero, è sufficiente dimostrare l'esistenza di un simulatore per esso.

Di seguito viene presentato un esempio riassuntivo dei concetti visti fino ad ora che coinvolge i protocolli a conoscenza-zero:

Esempio 6. Peggy afferma di essere in grado di poter contare i granelli di sabbia di una spiaggia, ma non vuole rivelare a Victor come ci riesce (witness). Victor vuole essere in grado di verificare la veridicità di questa sua affermazione con un certo grado di sicurezza (completezza), senza farsi ingannare da Peggy (correttezza). Non gli interessa, però, di essere sicuro che Peggy sia effettivamente a conoscenza di un qualche metodo per poter rispondere (ovvero, non è richiesta la validità dei Proof of Knowledge), egli vuole solo verificare la veridicità della sua affermazione (cioè, se $x \in \mathrm{L}$) tramite una challenge, ripetuta in una serie di k tentativi (sistema a dimostrazione interattiva).
Victor non sa quanti granelli di sabbia sono presenti in spiaggia e non è in grado di contarli (capacità computazionale limitata), tantomeno sa se Peggy è in grado di farlo (si ipotizza, però, che potrebbe esserlo: capacità computazionale illimitata), ma egli può comunque verificarlo: per dimostrare se Peggy sta dicendo la verità o meno è sufficiente chiederle ad ogni tentativo di contare i granelli di sabbia (o se il loro numero è pari o dispari) e comunicarlo (fase di commitment). Infine, Victor dovrà scegliere casualmente se prendere o meno un granello di sabbia dalla spiaggia e nasconderlo nella propria tasca (bit random che definisce la challenge) e porre a Peggy nuovamente la stessa domanda. In questo modo, se le risposte date da Peggy ad ogni interazione sono corrette, cioè hanno risolto la challenge, Victor potrà ritenere la sua

affermazione veritiera.
Victor non ha imparato nulla su come poter contare il numero di granelli di sabbia in una spiaggia (conoscenza-zero).

Si noti che i punti critici di questi processi di identificazione, chiamati anche Σ-*protocols*[1], sono il numero k di interazioni effettuate e la scelta randomica delle challenge, operazioni gestite entrambe dal Verifier.
Si noti anche che, se il Prover non è a conoscenza di una witness, la probabilità che esso ha di rispondere in maniera corretta ad ogni interazione è $\frac{1}{2}$, per cui la probabilità complessiva di riuscire ad ingannare il Verifier è di $\frac{1}{2^k}$. Tuttavia, sebbene la sicurezza per un Verifier aumenti con il numero di interazioni effettuate, dall'altra parte il processo di identificazione diventa dispendioso in termini di tempo. Questo, assieme alla scelta randomica delle challenge, obbliga il Verifier rimanere attivo per tutta la durata di esso.

Nella crittografia moderna, un protocollo viene determinato "*sicuro*" sulla base di problemi matematici formali in modelli ben definiti, e tale affermazione regge finché le assunzioni su cui si basa non sono dimostrate come false. I protocolli a conoscenza-zero trovano impiego in numerosi ambiti di verifica, ma nei sistemi atti all'accertamento, oltreché il riconoscimento, dell'identità, è necessario garantire anche la proprietà di prova di conoscenza. In particolare, si distinguono tre diversi livelli di sicurezza [10]:

- **Schemi di autenticazione** (*Authentication schemes*): P può provare a V di essere P, e nessun altro può provare a V di essere P.

- **Schemi di identificazione** (*Identification schemes*): P può provare a V di essere P, e V non può provare a qualcun altro di essere P.

- **Schemi di firma** (*Signature schemes*): P può provare a V di essere P, e V non può provare nemmeno a sé stesso di essere P.

[1] Un Σ-*protocol* (o *Sigma-protocol*) è un modello di protocollo basato su 3-round: commitment (P), challenge (V), risposta (P).

2.2 Il protocollo di identificazione di Fiat-Shamir

Il protocollo (o schema) di identificazione di Fiat-Shamir [10] definito nel 1987 è stato il primo schema di identificazione a conoscenza-zero introdotto, al quale si sono poi susseguiti numerosi altri come gli schemi di identificazione di Schnorr e di Guillou-Quisquater che ne hanno mantenuto la stessa struttura, vista la sua semplicità che lo rende adatto ad essere utilizzato in dispositivi a microprocessore come Smart-Cards.
La sicurezza di questo Σ-*protocol* è basata sulla difficoltà di invertire la funzione di potenza nell'algebra modulare, ovvero, calcolare la radice in modulo: Sia n un numero non primo, dato anche $t = s^2 \ mod \ n$, calcolare s, cioè la radice quadrata di t, è esponenzialmente difficile se non si conosce la fattorizzazione di n. Se n fosse primo, il problema sarebbe triviale. Al contrario, se n è grande e non primo il problema diventa esponenzialmente difficile da risolvere.
Il Prover possiede $n = pq$, con p, q primi, e un numero intero positivo $s < n$. In seguito esso calcola $t = s^2 \ mod \ n$, di conseguenza $\langle p, \ q, \ s \rangle$ sono tenuti segreti, mentre la coppia $\langle n, t \rangle$ è resa nota come una sorta di chiave pubblica. Il protocollo si sviluppa in questo modo:

1. P genera un numero intero random $r < n$ e lo nasconde calcolando ed inviando a V $u = r^2 \ mod \ n$ (commitment).

2. V genera un numero intero random e in $\{0, \ 1\}$ e lo comunica a P (challenge).

3. P calcola $z = rs^e \ mod \ n$, e lo comunica a V (risposta).
   Si noti che se $e = 0$ si ha semplicemente $z = r \ mod \ n$, altrimenti se $e = 1$ allora $z = rs \ mod \ n$.

4. V verifica che $z^2 \ mod \ n = ut^e \ mod \ n$.
   Più intuitivamente, V dispone di t, che si ricorda essere $s^2 \ mod \ n$, e $u = r^2 \ mod \ n$, per cui:

(a) Se $e = 0$: V deve aver ricevuto $r \ mod \ n$, per cui lo eleva al quadrato e verifica se corrisponde ad u inviato da P.

(b) Se $e = 1$: V deve aver ricevuto $rs \ mod \ n$, per cui lo eleva al quadrato e verifica se corrisponde a ut inviati da P.

Alcune considerazioni:
Si noti che il fatto che r sia nascosto è di fondamentale importanza per la sicurezza del protocollo: infatti, nel caso in cui $e = 1$, se r fosse noto allora ricavare s sarebbe facile anche se si opera nell'aritmetica modulare.
Si noti anche che e, il numero random di V, viene mantenuto pubblico, per cui il protocollo è Public-Coin, e che la challenge è risolvibile da chiunque con probabilità $\frac{1}{2}$, infatti quando $e = 0$ chiunque può essere in grado di risolverla dato che non è richiesto alcun segreto: questo può sembrare a primo impatto inconcludente, ma se si richiedesse sempre rs, chiunque intercetti la conversazione ha la garanzia di acquisire informazioni su parte del segreto di P, e anche se queste sono inutilizzabili perché protette dalla fattorizzazione in modulo, il protocollo non sarebbe più considerabile a conoscenza-zero. Con la randomizzazione delle challenge, invece, quando $e = 1$ chiunque intercetti la conversazione non può essere in grado di interpretare nulla, dato che tutto ciò che viaggia sul canale di comunicazione sono, dall'esterno, solamente numeri casuali. Ulteriormente, se V non randomizzasse il processo di identificazione in qualche modo, un qualsiasi P disonesto potrebbe ingannarlo facilmente:

1. P, che essendo disonesto non dispone di s, genera un numero intero random r, ma invece che inviare a V $u = r^2 \ mod \ n$, invia $u = \frac{r^2}{s^2} \ mod \ n$, dove s^2 si ricorda essere uguale a t (commitment).

2. A questo punto V, che riceve il valore random malevolo, chiede a P di calcolare $rs \ mod \ n$ (challenge).

3. Per ingannare V, P invia semplicemente $z = r \ mod \ n$ (risposta).

4. V verifica che $z^2 = ut \ mod \ n$, e questo è vero poichè P ha ingannato con successo V fornendogli $u = \frac{r^2}{s^2}$, di conseguenza $ut = r^2$, e z^2 è proprio r^2.

Se invece si introduce la randomizzazione delle challenge, aspetto che si dimostra essere fondamentale, il Prover non può ingannare il Verifier in alcun modo, dato che non sa a priori quale sarà la richiesta di V ed avrebbe quindi probabilità $\frac{1}{2}$ di indovinare. Proprio per questo motivo P deve prima rispondere con u, ed è proprio questa l'importanza del *commitment* da parte di P: se fosse V a rivelare per primo la challenge e, P potrebbe ingannarlo come è stato appena mostrato, mentre se è P a mandare il commitment per primo egli non avrà modo di scegliere e sarà necessariamente costretto ad impegnarsi alla casualità.
E' così dimostrato che entrambi gli aspetti di randomizzazione sono ciò che rende veramente sicuri questi tipi di protocolli Σ.
Formalmente, è possibile dimostrare facilmente le proprietà di completezza e correttezza:

- **Completezza**: se P è onesto la condizione finale ($z^2 = ut \ mod \ n$) è sempre verificata:
  - se $e = 0$ si ha $z^2 = r^2 \ mod \ n$ e $ut^e = r^2 \ mod \ n$.
  - se $e = 1$ si ha $z^2 = r^2 s^2 \ mod \ n$ e $ut^e = r^2 s^2 \ mod \ n$.
- **Correttezza**: se P è disonesto può tentare di prevedere i valori di e per ingannare V, ma dato che e è ricevuto dopo il commitment di P ed è generato randomicamente, le previsioni di P su e sono corrette con probabilità $\frac{1}{2}$, infatti:
  - se $e = 0$ secondo P, allora invierà $u = r^2 \ mod \ n$.
  - se $e = 1$ secondo P, allora invierà $u = \frac{r^2}{t} \ mod \ n$.

  ed in entrambi i casi invierà poi a V $z = r \ mod \ n$.

2.3 Il protocollo di identificazione di Schnorr

Il protocollo (o schema) di identificazione di Schnorr introdotto nel 1990 viene preso spesso come riferimento, questo perché esso non è solamente un protocollo-Σ Public-Coin a conoscenza-zero, ma un protocollo a conoscenza-zero a prova di conoscenza (*Proof of Knowledge*) [11][12].
Si ricorda a tal proposito, che le prove di conoscenza coinvolgono la proprietà di correttezza (*soundness*), la quale può essere espressa in due livelli: dimostrare l'appartenenza ad un linguaggio, o dimostrare la conoscenza di una witness; solamente l'ultima fa in modo che un protocollo sia definibile Proof of Knowledge.
Formalmente, il concetto di "conoscere una witness" è definito in crittografia come "è possibile dimostrare che la witness può essere efficientemente imparata da P" o, più precisamente: esiste un estrattore il quale, dato il codice del Prover, può estrarre in tempo polinomiale da esso una prova valida.
Il protocollo di identificazione di Schnorr è Proof of Knowledge, tuttavia, come molti altri nel suo genere rappresenta un'eccezione per quanto riguarda la proprietà di conoscenza-zero: non è possibile dimostrarla a meno che non si faccia l'assunzione speciale che il Verifier sia necessariamente onesto (*Honest-Verifier Zero-Knowledge, HVZK*).
La cosa potrebbe risultare perdere di efficacia, dato che si sta affermando che questi protocolli non rivelano nessuna informazione e sono quindi sicuri solamente se l'avversario è onesto, ma in realtà essa è quasi sempre un ottimo primo passo: esistono trasformazioni molto generali ed efficienti che permettono di convertire una larga classe di protocolli HVZK in protocolli a conoscenza-zero completa; questa classe di protocolli sono proprio quelli Public-Coin [13][14].
Come si è già osservato nel primo capitolo, i modelli a moneta pubblica o privata non sono troppo differenti, difatti $\mathrm{IP}[k] \subseteq \mathrm{AM}[k+2]$ ed esistono tecniche per trasformare protocolli Private-Coin in Public-Coin [4], ma questo aumenta necessariamente il numero di round necessari ad ogni interazione, e a seguito di due trasformate la complessità generale di un protocollo HVZK

Private-Coin ed il suo relativo costo computazionale potrebbero aumentare significativamente.
Uno dei vantaggi del protocollo di identificazione di Schnorr è proprio quello di essere incredibilmente leggero, tanto da permettere ad alcune sue implementazioni di funzionare in dispositivi come Smart-Card.
Sempre nel caso di Schnorr, inoltre, fatta l'assunzione del Verifier onesto è anche possibile dimostrare conoscenza-zero perfetta.
La sicurezza di questo protocollo risiede nella difficoltà di calcolare il logaritmo discreto, operazione che risulta essere sub-esponenziale:
Sia p un numero primo e g il generatore del gruppo ciclico moltiplicativo G_q definito dall'operazione di modulo p, dove q è l'ordine del gruppo.
Il Prover, denominato Alice, genera una coppia di chiavi utilizzando un numero intero casuale s, compreso fra 1 e q:

$$PK_A = g^s \ mod \ p$$
$$SK_A = s$$

Dove PK_A e SK_A sono rispettivamente la Public e Secret Key di Alice. Successivamente, quando ella vuole dimostrare la conoscenza della sua chiave segreta s ad un Verifier, denominato Bob, i due utilizzano questo protocollo interattivo:

1. Alice genera un numero intero random r compreso fra 1 e q, e lo nasconde calcolando ed inviando a Bob $u = g^r \ mod \ p$ (commitment).

2. Bob genera un numero intero random e in $\{0, 1\}$ e lo comunica ad Alice (challenge).

3. Alice calcola $z = se + r \ mod \ q$, e lo comunica a Bob (risposta).
   Si noti che se $e = 0$ si ha semplicemente $z = r \ mod \ q$, altrimenti se $e = 1$ allora $z = s + r \ mod \ q$.

4. Bob verifica che $g^z \ mod \ p \equiv PK_A^e \cdot u \ mod \ p$.
   Più intuitivamente, Bob dispone di PK_A, che si ricorda essere $g^s \ mod \ p$, e $u = g^r \ mod \ p$, per cui:

   (a) Se $e = 0$: Bob deve aver ricevuto $z = r \ mod \ q$. Per verificarlo calcola g^z e verifica se corrisponde ad u (o, più precisamente, a $PK_A^e \cdot u$, che equivale ad u dato che $e = 0$).

   (b) Se $e = 1$: Bob deve aver ricevuto $z = s + r \ mod \ q$. Per verificarlo calcola g^z e verifica se corrisponde a $PK_A^e \cdot u$.

Si dimostra ora le proprietà di completezza:

- **Completezza**: se Alice è onesta la condizione finale ($g^z \ mod \ p \equiv PK_A^e \cdot u \ mod \ p$) è sempre verificata:
  - se $e = 0$ si ha $g^z = g^0 \cdot u$, dove $u = g^r$ e $z = r$.
  - se $e = 1$ si ha $g^z = g^s \cdot u$, dove $u = g^r$ e $z = s + r$.

E la proprietà di validità, requisito più stretto di correttezza il quale verifica che il protocollo è Proof of Knowledge:

- **Validità** (correttezza):

  1. Alice genera un numero intero random r compreso fra 1 e q, e lo nasconde calcolando ed inviando a Bob $u = g^r \ mod \ p$ (commitment).
  2. Bob genera un numero intero random e_1 in $\{0, 1\}$ e lo comunica ad Alice (challenge).
  3. Alice calcola $z_1 = se_1 + r \ mod \ q$, e lo comunica a Bob (risposta).
  4. Il Knowledge Extractor rimanda Alice allo step successivo l'invio del commitment, quindi:

(2) Bob genera un numero intero random e_2 in $\{0, 1\}$ e lo comunica ad Alice (challenge).

(3) Alice calcola $z_2 = se_2 + r \ mod \ q$, e lo comunica a Bob (risposta).

L'osservazione chiave qui è che utilizzando la tecnica di rewinding il Knowledge Extractor può ingannare Alice nel fare due diverse trascrizioni utilizzando lo stesso numero casuale r, infatti ora:

$$\begin{aligned} & (z_1 - z_2)/(e_1 - e_2) \ mod \ q \\ = & ((se_1 + r) - (se_2 + r))/(e_1 - e_2) \ mod \ q \\ = & \ s(e_1 - e_2)/(e_1 - e_2) \ mod \ q \\ = & \ s \end{aligned}$$

Di conseguenza, tramite l'esistenza di un Knowledge Extractor in grado di ricavare il segreto dal Prover si è dimostrato che il protocollo non è solamente in grado di verificare se un'affermazione è appartenente ad un linguaggio ma, addirittura, se se ne conosce una witness efficiente: ciò permette al protocollo di Schnorr di garantire l'identificazione del Prover.
E' opportuno notare, tuttavia, che questa tattica può anche risultare una seria vulnerabilità in implementazioni errate del protocollo: se si utilizza accidentalmente lo stesso valore r in due diverse esecuzioni dell'algoritmo, un Verifier disonesto può risalire alla chiave segreta: ciò può verificarsi se si utilizza un inaffidabile "Random Number Generator" o, più precisamente, uno *"Pseudo-Random Number Generator"*[2].

Si dimostra ora la proprietà di conoscenza-zero (con Verifier onesto):
in normali circostanze si cerca di realizzare un simulatore che possa interagire con *qualsiasi* possibile Verifier e produrre una trascrizione simulata della

[2]Uno Pseudo-random Number Generator (PRNG) è un algoritmo per la generazione di numeri che possono apparire casuali ma sono, di fatto, predeterminati da condizioni riproducibili, a differenza di vero generatore di numeri casuali del quale non è possibile modellare in alcun modo gli stati.

conversazione, in modo tale che, anche se il simulatore non conosce il segreto, ne prova comunque la conoscenza.
Il protocollo di identificazione di Schnorr non possiede questo tipo di simulatore, per cui per poterne realizzare uno è necessario garantire che il Verifier esegua la sua parte di protocollo correttamente, ovvero che esso scelga la challenge e usando solamente il suo generatore di numeri casuali, e non in base all'input che il Prover fornisce.
Per ingannare il Verifier e provare che si conosce il segreto s di $g^s \ mod \ p$ si procede in questo modo:

1. Alice genera un numero intero random r_1 compreso fra 1 e q, e lo nasconde calcolando ed inviando a Bob $u = g^{r_1} \ mod \ p$ (commitment).

2. Bob genera un numero intero random e in {0, 1} e lo comunica ad Alice (challenge).

3. Il simulatore rimanda Alice allo step precedente l'invio del commitment, quindi:

   (1) Alice genera un numero intero random z compreso fra 0 e $q - 1$, e un nuovo intero malevolo r_2 tale che $u = g^{r_2} = g^z \cdot g^{s(-e)}$ e lo invia a Bob (commitment).

   (2) Bob genera lo stesso intero random e in {0, 1} e lo comunica ad Alice (challenge).

   (3) Alice, che ha ingannato Bob conoscendone già la challenge e rispondendo con un r opportuno (r_2), ignora la challenge di Bob e spedisce semplicemente z (risposta).

   (4) Bob sarà stato ingannato con successo. Infatti egli verifica che $g^z \ mod \ p \equiv PK_A^e \cdot u \ mod \ p$ come di consueto, ma ora $u = g^z \cdot g^{s(-e)}$, per cui si ha che $g^z \ mod \ p \equiv g^{se} \cdot g^z \cdot g^{s(-e)} \ mod \ p$, ed inviando z:

   - se $e = 0$ si ha $g^z = g^0 \cdot g^z \cdot g^0$
   - se $e = 1$ si ha $g^z = g^s \cdot g^z \cdot g^{-s}$

Di conseguenza, la trascrizione $\langle u, e, z \rangle$ risulterà agli occhi del Verifier come perfettamente valida, anche se il simulatore non ha idea del segreto s.
Inoltre, dato che la distribuzione statistica del protocollo del simulatore è identica a quella del protocollo reale, ne consegue che il protocollo di identificazione di Schnorr è a conoscenza-zero perfetta (ricordando di essere sempre sotto l'ipotesi di un Verifier onesto).

2.4 Conoscenza-zero non-interattiva

A seguito della prima pubblicazione riguardo la conoscenza-zero venne rilasciata, tre anni dopo, la dimostrazione matematica dell'esistenza di prove a conoscenza-zero non interattive da Manuel Blum, Silvio Micali e Paul Feldman in "Non-Interactive Zero-Knowledge and Its Applications" [15]. Nel 1991 inoltre, sempre Blum e Micali assieme ad altri due crittografi italiani, Alfredo De Santis e Giuseppe Persiano, ne parlarono più approfonditamente estendendone i concetti in "Non-Interactive Zero-Knowledge" [16].
Il difetto principale dei normali Zero-Knowledge Protocols, infatti, è che essi sono in grado di funzionare solamente se il Verifier è online e disposto ad interagire col Prover scegliendo una challenge randomica; ciò di cui si discute in questi paper è la possibilità di disfarsi perfino dell'interazione ciclica fra le parti, della scelta randomica della challenge del Verifier e consentire la verifica del segreto del Prover con un singolo messaggio: tutto questo è possibile se viene scambiata prima di comunicare una stringa corta casuale comune (*Common Reference String*, CRS) da utilizzare come chiave. Ciò si traduce in linguaggio matematico col concetto di funzione one-way (o unidirezionale), ed in crittografia col concetto di funzione di hash[3]: essa è infatti sufficiente per ottenere Zero-Knowledge computazionale.
Per permettere lo scambio della chiave è necessaria, tuttavia, una fase iniziale di comunicazione, detta di *"trusted setup"*, in cui un *"trusted party"* la

[3]Una funzione di hash è una funzione one-way che mappa dati di dimensione arbitraria (messaggio) in una stringa binaria di dimensione fissa (message digest).

genera pubblicamente.
Un'alternativa al modello CRS è utilizzare un modello a oracolo random[4].
Si osserva ora più formalmente il processo appena descritto che consente di trasformare un protocollo Zero-Knowledge interattivo in non-interattivo.

2.4.1 L'euristica di Fiat-Shamir

L'euristica (o trasformata) di Fiat-Shamir è una tecnica per convertire uno schema di identificazione Σ Public-Coin in uno schema di firma [10].
Il modo in cui essa opera è:

- facendo collassare il numero di round necessari in un unico round, aumentando lo spazio utilizzato per la challenge da {0,1} in uno spazio più grande che permette di controllare l'errore di correttezza (ad esempio $\mathbb{Z}_q$), al costo di rendere il protocollo Honest-Verifier Zero-Knowledge.

- non lasciando al Verifier il compito di generare le challenge, ma calcolandola utilizzando una funzione di hash.

Dato che questa tecnica va a sostituire completamente la parte di interattività del protocollo (considerando che riduce il numero di round a uno e il Verifier non deve nemmeno rispondere e generare la challenge), essa può essere vista anche come convertire un protocollo PoK interattivo in uno non-interattivo (la Zero-Knowledge non è necessariamente richiesta).
Il motivo per il quale il protocollo deve essere necessariamente Public-Coin è che la trasformata di Fiat-Shamir sostituisce il passo di generazione delle monete da parte del Verifier con la generazione delle monete da parte del Prover tramite la funzione di hash, per cui il protocollo utilizzato non può essere uno a moneta privata, la cui forza dipende proprio dal fatto che il Prover non ha accesso alle monete del Verifier.

[4]Un oracolo random è una funzione matematica che associa ad ogni possibile domanda una risposta veramente random scelta all'interno del suo dominio di output. Essi sono utilizzati nelle dimostrazioni crittografiche quando non sono note implementazioni concrete di funzioni con proprietà matematiche richieste dal problema.

Si considera ora come esempio lo schema di identificazione di Schnorr per osservare come viene applicata la trasformata di Fiat-Shamir.

2.4.2 Lo schema di firma di Schnorr

Lo schema di firma di Schnorr è un protocollo Zero-Knowledge non-interattivo che è anche un Proof of Knowledge ed un concetto di firma digitale, ottenuto applicando l'euristica di Fiat-Shamir al protocollo di identificazione di Schnorr visto precedentemente.
Infatti, inserendo un messaggio M come valore opzionale si può ottenere una firma su M, la quale può essere prodotta solamente da qualcuno che conosce la chiave segreta s.
Definita una funzione di hash $H : \{0,1\} \rightarrow \mathbb{Z}_q$ comune fra le parti, il protocollo rivisitato per provare la conoscenza del segreto s riguardante la chiave pubblica PK_A è il seguente:

1. Alice genera un numero intero random r compreso fra 1 e q, e lo nasconde calcolando $u = g^r \ mod \ p$ (commitment).

2. Alice utilizza $H()$ per calcolare la challenge $e = H(u \ || \ M)$, dove u viene rappresentato come una stringa di bit, M è un messaggio opzionale e "||" rappresenta l'operazione di concatenazione (challenge).

3. Alice calcola $z = -se + r \ mod \ q$ e comunica a Bob la coppia (z, e) che costituisce la firma del messaggio (risposta).

4. Bob deve solamente effettuare delle verifiche grazie ad H() in comune, per cui calcola il suo $u_B = PK_A^e \cdot g^z$ ed il suo $e_B = H(u_B \ || \ M)$, e controlla se $e_B = e$. Ciò è valido perché se Alice è onesta allora conosce s e può calcolare z, per cui si ha $u_B = g^{se} \cdot g^{-se+r} \ mod \ p$ e $u = g^r \ mod \ p$ (completezza).

Anche in questa implementazione del protocollo, tuttavia, se lo stesso r viene utilizzato in due firme distinte è possibile risalire alla chiave privata.

E' infatti è sufficiente sottrarre i due valori di z:

$$z_2 - z_1 = (r_2 - r_1) - s(e_2 - e_1)$$

Per cui, se $e_2 \neq e_1$, si ha:

$$s = \frac{z_1 - z_2 + r_2 - r_1}{e_2 - e_1}$$

Per quanto riguarda la sicurezza, invece, è stato dimostrato che lo schema di firma di Schnorr si può considerare tale se la funzione di hash H viene definita e modellata come un oracolo random [17].

Teorema 3. *Dato uno schema di identificazione* Π*, sia* Π' *il suo schema di firma ottenuto applicando la trasformata di Fiat-Shamir a* Π*, se* Π *è sicuro ed* H *è modellato come un oracolo random, allora* Π' *è sicuro [18].*

Teorema 4. *Se il problema del logaritmo discreto è difficile relativamente al problema definito (G, q, g), allora il protocollo di identificazione di Schnorr può definirsi sicuro [19].*

2.5 Analisi dei problemi

Nonostante proprietà come la conoscenza-zero e la validità dei Proof of Knowledge possano sembrare assunzioni forti, esse non sono necessariamente una garanzia di sicurezza del protocollo, difatti si è già osservato nel corso di questo capitolo come tali protocolli possono nascondere diverse vulnerabilità. Di seguito si propone un'analisi dei principali tipi di problemi che possono coinvolgere i protocolli a conoscenza-zero.

2.5.1 Confronto con la firma digitale

Sono già state mostrate le principali differenze fra un normale protocollo a conoscenza-zero interattivo ed una firma digitale, tuttavia l'introduzione dei protocolli a conoscenza-zero non interattivi e dei conseguenti schemi di firma può portare ad ulteriori domande. Ci si potrebbe chiedere, ad esempio, perché utilizzare la firma digitale invece di uno schema di firma. Si vuole quindi distinguere ogni aspetto di tali sistemi tramite un esempio, in modo da rivelarne e discuterne i punti deboli e di forza.

Esempio 7. A pubblica un libro M, il quale è associato ad A con una chiave pubblica. In seguito, egli vuole provare a B di aver scritto il libro ed essere l'autore originale.

Ora, se si sceglie di utilizzare il concetto di firma digitale per dimostrare che A ha scritto il libro, è sufficiente che egli spedisca M a B, e assieme ad esso anche $SK(H(M))$, dove H è una funzione di hash e SK è la chiave privata di A: in questo modo B può verificare tramite la chiave pubblica di A non solo la provenienza del messaggio, ma anche che esso non sia stato modificato (integrità).
Tuttavia, questo meccanismo di interazione fra A e B non è sicuramente a conoscenza-zero: il segreto di A viaggia nella rete, e chiunque intercetti la comunicazione può verificare:

- che A ha scritto il libro, utilizzando la chiave pubblica di A.
- che A e B hanno comunicato, e B stesso potrebbe comunicarlo a chiunque.

Di conseguenza, se si aggiunge come richiesta il vincolo che A voglia dimostrare di aver scritto il libro solamente a B allora questo meccanismo non è più adatto, a meno che non si utilizzi un canale di comunicazione sicuro.
Questo sistema presenta però un punto di forza: vedere la firma di A molteplici volte e sotto diverse forme e istanze non diminuisce la sicurezza della firma digitale nel lungo termine.

Se si sceglie invece di utilizzare un meccanismo a conoscenza-zero interattivo e con proprietà di validità per dimostrare che A è a conoscenza del segreto, allora nessuno potrà sapere che A ha scritto il libro (tranne B) e che A e B hanno comunicato. Inoltre, nemmeno B stesso sarà in grado di dimostrare di aver comunicato con A.
Tuttavia in questo modo, non essendo presente un meccanismo di firma digitale, non è possibile verificare se il libro che A manda a B è stato modificato o meno. Inoltre, dato che i protocolli Zero-Knowledge utilizzano un meccanismo interattivo e probabilistico per garantire tale proprietà (A deve generare un numero casuale r), B deve essere necessariamente online e disposto a comunicare, e la sicurezza di tali protocolli diminuisce nel tempo col numero di interazioni effettuate (se si genera lo stesso r in due interazioni diverse del protocollo, è possibile risalire al segreto).
Ciò che ne risulta è che i protocolli Zero-Knowledge non condividono conoscenza all'esterno rispetto alla firma digitale, ma le informazioni segrete e pubbliche devono essere aggiornate più di frequente (si parla comunque di periodi di tempo lunghi).
Nel caso si scelga invece di utilizzare un meccanismo a conoscenza-zero non-interattivo che è anche uno schema di firma, è possibile garantire, assieme ai vantaggi dei protocolli a conoscenza-zero, anche l'integrità del messaggio concatenandolo alla challenge, tramite l'utilizzo della funzione di hash.
Tuttavia, nonostante venga rimossa anche l'interazione fra A e B, i punti deboli della conoscenza-zero persistono e gli schemi di firma possono essere costosi a livello computazionale.
Si analizza ora un altro tipo di attacco al quale i protocolli Zero-Knowledge possono essere soggetti.

2.5.2 Il Chess Grandmaster Problem

Finora si è osservato come dall'interazione fra le parti di un protocollo a conoscenza-zero possano esistere Prover disonesti che tentano di falsificare la propria identità, come Verifier disonesti che tentano di ricavare più informa-

zioni dalla comunicazione costruendo challenge apposite. Si sono mostrate le tattiche e le proprietà necessarie a prevenire questi tipi di attacchi ed è stata mostrata l'importanza di possedere un buon Random Number Generator.
Uno dei problemi del quale non si è ancora parlato per questi protocolli è il Man-in-the-Middle, identificato nel "Chess Grandmaster Problem" e descritto da Goldwasser nel 1985. L'analogia è la seguente:

Esempio 8. Eve (da *"Eavesdropping"*, ascoltare segretamente conversazioni per acquisire informazioni), che non sa giocare a scacchi, può sconfiggere un campione. Sfida contemporaneamente Gary Kasparaov e Magnus Carlsen ad una partita allo stesso tempo, ma in stanze diverse. Gioca i bianchi contro Kasparaov e i neri Carlsen. Nessun campione conosce dell'esistenza dell'altro. Carlsen, giocando i bianchi, fa la prima mossa. Eve registra la mossa e va nella stanza di Kasparaov e, giocando i bianchi in quella stanza, fa la prima mossa contro di lui. Kasparaov fa la sua prima mossa giocando i neri, ed Eve registra la mossa e va nella stanza di Carlsen, facendo la stessa mossa. Questo continua finché Eve vince una partita in una stanza e perde nell'altra, oppure finché entrambi i match non terminano in parità. In realtà, Kasparaov ha giocato contro Carlsen e Eve fa solamente da intermediaria (middleman) copiandone le mosse di entrambi, ma se né Carlsen né Kasparaov sanno dell'esistenza dell'altro, entrambi saranno impressionati dal gioco di Eve.

Potenzialmente questo tipo di attacco può essere utilizzato contro i protocolli Zero-Knowledge: mentre Alice sta provando la sua identità a Bob, Eve può fare da intermediario e provare simultaneamente a Bob che lei è Alice; tuttavia è necessario osservare il problema anche da un punto di vista concreto: molti crittografi non considerano il Chess Grandmaster Problem come un attacco valido, tra i quali Douglas R. Stinson che ne discute in *Cryptography: Theory in Practice* [20].
A livello pratico, infatti, ogni router su internet può essere considerato un "man in the middle", dato che inoltra messaggi inalterati. L'obbiettivo degli schemi di identificazione è quello di assicurarsi che Bob possa verificare che *qualcuno* (in questo caso Alice) all'interno del canale sia chi dice di essere.

Inoltre, essi sono stati originariamente introdotti per dispositivi come Smart Cards dove il canale di comunicazione è garantito essere sicuro, per cui se utilizzati in un contesto con potenziali attaccanti man-in-the-middle sono necessari ulteriori meccanismi di sicurezza.
In situazioni reali si combina lo schema di identificazione ad un meccanismo di *Key-Exchange* (ad esempio Diffie-Hellman, che utilizza la stessa chiave pubblica e privata osservata nel protocollo di Schnorr basata sul problema del logaritmo discreto), e li si vincola insieme tramite un valore segreto s. Ad esempio:

- Si esegue il protocollo DH di Key-Exchange in modo tale che Bob e un'altra entità X condividano un segreto s.
  Bob spera che X sia Alice, ma non può saperlo con certezza, per cui decide di utilizzare uno schema di identificazione. Tuttavia, egli non può eseguirlo in maniera standard su tale canale di comunicazione, perché X potrebbe essere Eve, la quale potrebbe aver creato un secondo canale di comunicazione sicuro s' per inoltrare i messaggi ad Alice (Diffie-Hellman man-in-the-middle).

- Avendo creato il segreto condiviso s, si osserva che Bob ha già ricevuto qualcosa in input, di conseguenza Alice può semplicemente utilizzare s come challenge per lo schema di identificazione. Questo ha l'effetto di vincolare il canale di comunicazione allo schema, ed una esecuzione avvenuta con successo di tale protocollo garantisce che Alice sia effettivamente dall'altra parte del canale.

L'unico modo che Eve ha di ingannare Bob è riuscendo a creare due canali tali che $s = s'$. Lo stesso principio viene utilizzato in ogni protocollo internet sicuro, incluso TLS.
Può accadere, quindi, che questo tipo di attacchi definiti come "passivi" perché l'attaccante stesso agisce in tale maniera, spesso non vengano considerati validi a livello pratico. Il caso in esame è considerato un attacco passivo perché lo schema non viene modificato in alcun modo (Bob ed Alice

eseguono il protocollo ed Eve fa solo da intermediario; il risultato sarebbe stato lo stesso anche se Eve non fosse stata presente).
L'attacco si può considerare, invece, attivo, se Eve si comporta nella seguente maniera [21]:

- Eve crea un nuovo messaggio e lo inserisce nel canale.
- Eve modifica un messaggio nel canale.
- Eve devia un messaggio nel canale in modo tale che esso sia inviato a qualcuno di diverso rispetto al destinatario previsto.

Un altro modo per proteggersi dal Chess Grandmaster Problem è predeterminare il tempo di invio dei messaggi e definire un opportuno limite di tempo per le risposte, non lasciando ad Eve il tempo di operare.

Capitolo 3

Sviluppi e applicazioni future

La rimozione dell'interattività nella comunicazione ed il fatto che non sia più necessario che il Verifier sia online e generi challenge apre la strada a numerose applicazioni della conoscenza-zero.
Ad oggi, gli studi spaziano in metodi per rendere la conoscenza-zero più fruibile, come la possibilità di creare protocolli a conoscenza-zero non-interattivi senza la necessità di oracoli random [22], utilizzando una trasformata diversa da quella proposta da Fiat-Shamir [23], ma soprattutto, in sistemi atti a ridurre il costo di tali protocolli e aumentarne le performance.
Nel 2012 viene così introdotta una variante dei protocolli NIZK in "*From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again*", denominata ZK-SNARK [24].

3.1 ZK-SNARK

Un protocollo ZK-SNARK è un protocollo NIZK che combina la non-interattività a meccanismi atti ad ottenere prestazioni elevate. Esso significa "Zero-Knowledge succinct non-interactive argument of knowledge".

L'acronimo indica infatti:

- **ZK** - Zero Knowledge: la w non viaggia nel canale di comunicazione.
- **S** - *Succinct* (succinto, sintetico): la dimensione della proof è sempre piccola e costante e la loro verifica è al più sub-lineare.
- **N** - *Non-interactive*: la comunicazione avviene in un unico messaggio.
- **AR** - *Argument (o "computationally sound proof")*: la proprietà di correttezza è computazionalmente sicura (*computational soundness*).
- **K** - *of Knowledge-validity*: la probabilità che un Knowledge Extractor riesca ad estrarre la witness è almeno alta quanto la probabilità di successo che ha il Prover di convincere il Verifier.

A seguito di questa definizione è necessario analizzare più approfonditamente la proprietà di correttezza. Essa, come la proprietà di conoscenza-zero, può essere definita sotto tre diversi livelli di sicurezza, a seconda delle assunzioni effettuate per l'esistenza del Knowledge Extractor:

- Correttezza **perfetta** (*perfect soundness*): nemmeno Prover con capacità computazionali illimitate sono in grado di ingannare il Verifier.
- Correttezza **statistica** (*statistic soundness*): la probabilità che un Prover con capacità computazionali illimitate riesca ad ingannare il Verifier è statisticamente trascurabile.
- Correttezza **algoritmica** o **computazionale** (*computational soundness*): Prover con capacità computazionali limitate non sono in grado di ingannare il Verifier.

Il termine "*argument*" indica quindi una prova la cui sicurezza dal punto di vista della correttezza dipende da assunzioni computazionali. Ciò si contrappone al termine "*proof*", nel quale la correttezza è statistica.

Di conseguenza, la proprietà di correttezza nelle Zero-Knowledge Proofs (ZKP) è più sicura rispetto a quella dei *Zero-Knowledge Argument* (ZKA), i quali vengono utilizzati all'interno dei protocolli SNARK. Lo stesso concetto si applica ai Proof of Knowledge e gli *Argument of Knowledge*.
Tuttavia, si ricorda che la maggior parte delle applicazioni pratiche attuali in ambito crittografico si basano su problemi di difficoltà computazionale, per cui esse sono ritenute, nella maggior parte dei casi, sufficientemente sicure.
Inoltre, per introdurre i protocolli ZK-SNARK risulta utile approfondire i concetti di "proof" e "witness": una witness, oltre che come segreto può essere vista come uno specifico tipo di prova, ovvero una prova che dimostra l'appartenenza al linguaggio di statement definiti in NP. La classe di complessità NP può, a sua volta, essere vista come la classe dei linguaggi per i quali esistono dimostrazioni "efficienti" agli statement, ovvero esiste una macchina deterministica V che verifica in tempo polinomiale:

$$x \in L \iff \exists\ w : V(x, w) = 1$$

Esistono però molti altri modi di dimostrare gli statement, e non tutti gli statement sono contenuti in NP. Un modo che si è già osservato è proprio tramite un sistema a prove interattive, nel quale si lascia che il Verifier sia probabilistico e gli si concede di commettere errori con una bassa probabilità. E' possibile dimostrare anche proprietà aggiuntive come la conoscenza-zero, per la quale il Prover utilizza la witness ma non la invia nel canale di comunicazione.

Nel caso dei protocolli SNARK sono richieste prove di dimensione molto piccola, in particolare di dimensione minore della witness NP originale, in modo tale che il tempo richiesto per verificarle sia minore del tempo necessario a V di calcolare $V(x, w)$ e verificarlo nella classica maniera di NP.
Come ogni protocollo a conoscenza-zero non-interattivo, anch'essi possono necessitare di una fase iniziale di "trusted setup" (onetime): questo compito è affidato ad una terza entità oltre a P e V, che prende il nome di "Key Generator" (G).

Egli deve generare i parametri pubblici, ovvero una "*Proving Key*" (PK) ed una *Verification Key* (VK), le quali vengono generate tramite un programma C ed un parametro segreto *lambda* ricevuto in input.
Il Prover dispone quindi di PK, una witness w e lo statement x, e tramite essi egli genera la proof $prf = P(PK, x, w)$.
Il Verifier calcola $V(VK, x, prf)$, che ritorna 1 (o "true") se la proof è valida, 0 (o "false") altrimenti. In altre parole, questa funzione sarà valida se il Prover dimostra di conoscere una witness $w : C(x, w) = 1$.
Ciò che può risultare essere un problema per la sicurezza è il parametro segreto lambda utilizzato nella fase di generazione: se esso viene scoperto è possibile generare una proof fasulla prf', tale che $V(VK, x, prf')$ sia verificata senza disporre della conoscenza di w. Di conseguenza, un aspetto fondamentale di questi protocolli è assicurarsi che la fase di generazione avvenga in maniera sicura.
Un esempio di interazione fra le parti nei protocolli ZK-SNARK è il seguente:

1. Viene utilizzato un programma C tale che:

```
function C(x, w) {
    return (sha256(w) == x);
}
```

   Viene generato randomicamente il parametro *lambda* ed eseguito il generatore G per ottenere le chiavi necessarie:

```
(PK, VK) = G(C, lambda)
```

   Ad Alice viene consegnata PK e a Bob VK, facendo attenzione a non divulgare anche lambda, altrimenti chiunque potrebbe generare prove fasulle.

2. Alice deve dimostrare di conoscere il segreto s il cui valore di hash corrisponde allo statement x, ovvero una pubblica hash H.

Esegue quindi l'algoritmo utilizzando i suoi parametri in input per generare la prova:

```
1    prf = P(PK, H, s)
```

ed invia a Bob prf.

3. Bob esegue la sua funzione di verifica:

```
1    V(VK, H, prf)
```

la quale ritornerà "true" se Alice conosce il segreto.
Alice non ha rivelato nulla riguardo s a Bob se non la sua conoscenza.

Si noti che in contesti peer-to-peer come Blockchain in cui un Verifier può essere anche Prover e viceversa, la fase di trusted setup non può essere eseguita da P o V. Se in questo esempio fosse Bob ad avviare G e generare il parametro lambda ed in seguito volesse provare ad Alice di conoscere un segreto utilizzando le stesse chiavi, egli non potrebbe farlo, in quanto Alice non può sapere se Bob ha tenuto salvato il segreto: per ovviare a ciò, tale fase deve necessariamente essere eseguita da un'autorità fidata, indipendente da P e V.

L'utilizzo di meccanismi interattivi, fino a questo momento, ha rappresentato diverse problematiche legate alla sicurezza dei dati. In un meccanismo interattivo, ad esempio, un Verifier deve inevitabilmente mantenere salvati al suo interno i dati scambiati con il Prover finché quest'ultimo non viene autenticato; questo aspetto implica che il Verifier utilizzi sistemi per mantenere le informazioni raccolte al sicuro, altrimenti potrebbe essere soggetto a diverse tecniche di attacco al protocollo, con conseguente perdita dei parametri segreti.
Inoltre, sempre nei meccanismi interattivi, la prova rilasciata da P può essere considerata valida da un solo Verifier; nessun altro può fidarsi della stessa

proof, dato che un ipotetico Verifier disonesto potrebbe cospirare con il Prover, rivelando parametri segreti importanti che permettono di falsificare la prova [25].
Se si volesse convincere diversi "*parties*" allo stesso tempo è necessaria un'interazione separata per ognuno di essi, il che rende tali protocolli inefficienti sotto questo contesto, dato che il Prover dovrebbe rimanere costantemente online. I parametri generati dai protocolli ZK-SNARK rappresentano, invece, un vantaggio. Essi sono:

- **Pubblici**: chiunque può vederli senza che questo sia un problema per la sicurezza del protocollo.
- **Riutilizzabili**: è possibile utilizzare gli stessi parametri per verifiche diverse.
- **Affidabili**: una volta generati, essi sono veloci e consistenti nel tempo.
- **Sicuri**: essi non permettono utilizzo illecito se chi li usa non dispone di informazioni segrete.

Tali parametri, grazie alla loro efficienza, fanno sì che l'esecuzione dell'algoritmo del Verifier sia proporzionale alla dimensione dell'input x, cioè $O(|x|)$, e che la proof abbia dimensione costante, cioè $O(1)$ (da qui il suffisso "succinct"). Tuttavia, la fase di setup di G e l'algoritmo del Prover per generare la proof sono entrambi molto costosi: il loro costo è proporzionale al tempo di esecuzione del programma C, e la dimensione della Proving Key è grande per lo stesso motivo; la loro creazione è quindi molto dispendiosa, specialmente in termini di tempo e overhead di memoria. La fase di setup, sebbene debba essere eseguita solo una volta, rappresenta il "collo di bottiglia" dei protocolli SNARK.
Per ovviare a questi problemi, negli ultimi anni si è pensato a soluzioni che permettessero di rendere tale sistema distribuito, utilizzando la tecnica di "cluster computing" per aumentare sicurezza e potenza di calcolo, invece che affidarsi ad un sistema monolitico e centralizzato [26].

3.1.1 Zcash

Una delle applicazioni pratiche più comuni dei protocolli ZK-SNARK è nel contesto di sistemi distribuiti come le blockchain: nel 2014 è stato realizzato "Zerocash" [27], protocollo successore di "Zerocoin" (2013) al quale ha contribuito alla realizzazione anche Alessandro Chiesa, crittografo italiano coinvolto nella definizione degli stessi protocolli ZK-SNARK. Tale protocollo è divenuto poi, nel 2016, una vera e propria criptomoneta: Zcash.
Zcash è una criptomoneta decentralizzata peer-to-peer, ed essendo una fork di Bitcoin anch'essa ha un limite massimo di 21 milioni di monete. Grazie all'utilizzo dei protocolli ZK-SNARK, essa è in grado di garantire, a differenza di Bitcoin, trasparenza selettiva e totale privacy delle transazioni e dell'identità di mittente e/o destinatario, mediante l'utilizzo di indirizzi schermati ("*shielded*"). Ad ogni indirizzo shielded è corrisposta una "Viewing Key" la quale, se diffusa, permette di decodificare tutte le informazioni nascoste delle proprie transazioni protette; futuri aggiornamenti ne permetteranno anche la decodifica selettiva.
La privacy delle transazioni shielded è ottenuta mediante meccanismi di hash, ma è l'aggiunta di un algoritmo ZK-SNARK che permette ai mittenti e ai destinatari interessati di comprovare che la transazione protetta sia corretta, garantendo, senza rivelare nulla, la validità della stessa.
Tale meccanismo permette la pubblicazione di transazioni Zcash completamente crittografate su di una Blockchain pubblica, di conseguenza un destinatario che non dispone della Viewing Key del mittente può vedere solamente la quantità di monete ricevute, il transaction ID che permette di risalire alla transazione in Blockchain, ed un eventuale nota lasciata dal mittente stesso.
Il problema principale di Zcash come criptomoneta è causato dalla sicurezza intrinseca del protocollo: le transazioni protette non permettono l'identificazione di monete contraffatte. In sistemi "open ledger" come Bitcoin, cioè in cui ogni transazione è pubblica ed interamente visualizzabile in Blockchain, è possibile individuare e risalire alla generazione di eventuali monete contraffatte, mentre in Zcash ciò non è possibile, il che costituisce un critico

problema di sicurezza.
Per poter eseguire le transazioni sono necessari i parametri pubblici generati dal protocollo ZK-SNARK che permettono la creazione e verifica delle proof. La generazione di questi parametri avviene nella fase di trusted setup, la quale consiste nella creazione di una coppia di chiavi pubblica e privata ed, in seguito, nella distruzione della chiave privata (denominata anche "*toxic waste*") ed il mantenimento della pubblica, ma se si risalisse alla prima sarebbe possibile generare monete senza essere scoperti.
Proprio per questo motivo, la fase di trusted setup per la generazione delle chiavi pubbliche che è avvenuta per Zcash è stata altamente delicata, complessa e lunga, tanto da assumere il nome di "cerimonia": sei partecipanti, ubicati in parti diverse del mondo, sono stati coinvolti per la generazione di una coppia di chiavi pubblica e privata a frammenti; l'unico modo per risalire alla chiave privata, così facendo, è che tutti i partecipanti siano stati disonesti, mentre perché la cerimonia abbia avuto successo è sufficiente che uno solo abbia cancellato il proprio frammento.
Oltre a questo primo livello di sicurezza sono state adottate molte altre tecniche, come l'utilizzo di computer "air gap" appositamente acquistati per mantenere i generatori fisicamente disconnessi da qualsiasi rete, l'utilizzo di DVD come memoria di massa per la registrazione di ogni operazione ed una distribuzione di Linux creata appositamente per la cerimonia. Il tutto è stato documentato da videocamere ed, infine, ogni macchina contenente i frammenti della chiave privata è stata distrutta fisicamente.
La generazione delle proof di Zcash è stata progettata in modo tale che sia il Prover a dover svolgere la maggior parte del lavoro in anticipo, semplificando così il processo di verifica.
Il protocollo ZK-SNARK implementato è il seguente:

- Il creatore di una transazione Zcash shielded svolge il processo di generazione della proof ($\sim$ 200 byte) tramite la Proving Key, lo statement ed il suo segreto assegnato con la generazione dell'indirizzo, operazione che può richiedere diversi secondi ($\sim$ 2.3s).

- I miner effettuano la verifica di validità della transizione tramite la Verifying Key, la quale richiede pochi millisecondi ($\sim$ 10ms).

L'implementazione di ZK-SNARK per Zcash risulta essere quindi molto efficiente e sicura a seguito della fase di trusted setup.

Un altro problema significativo di questo protocollo è costituito dal fatto che esso non è "*quantum-secure*", ovvero la sua sicurezza potrebbe essere violata da un computer quantistico. Ciò viene analizzato dettagliatamente in uno studio effettuato nel 2021 nel quale Zcash, assieme a tutte le principali criptomonete di oggi, risulta essere vulnerabile al problema: questo è dovuto al meccanismo di generazione delle chiavi asimmetriche utilizzato, ECDSA, la cui sicurezza si basa sul problema del logaritmo discreto definito su curve ellittiche; se da una parte esso è difficile da risolvere per normali computer e permette di rendere le prove corte e facilmente verificabili, dall'altra tale assunzione crittografica risulta debole contro computer quantistici, per i quali esistono algoritmi che possono risolvere efficientemente l'aritmetica modulare e risalire così alla chiave privata [28].

3.2 ZK-STARK

ZK-STARK è una variazione dei protocolli ZK-SNARK definita nel 2018 la quale è, a differenza di questi ultimi, trasparente ("*transparent*") [29]:

- **T** - *Transparency*: la fase di trusted setup non è richiesta e sono necessarie assunzioni crittografiche minori.

Ciò rende questi protocolli molto più sicuri e resistenti dei ZK-SNARK.
La proprietà di trasparenza è ottenuta mediante meccanismi di randomness pubblica: esso è, infatti, un protocollo Public-Coin definito nel modello a oracolo random (a differenza di ZK-SNARK che è definito mediante delle Common Reference Strings), il quale fa utilizzo di funzioni di hash con caratteristiche di "*collision-resistance*" come SHA-256 o SHA3.

Queste assunzioni fanno sì che il protocollo non necessiti della fase di trusted setup, ma a discapito di proof molto più grandi ($\sim$ 45.000 byte).
Nonostante questo, la generazione delle prove è leggermente più veloce dei protocolli SNARK ($\sim$ 1.6s) e la loro verifica leggermente più lenta ($\sim$ 16ms), anche se i due protocolli rimangono sullo stesso ordine di magnitudine[1].
Il protocollo è, oltre a ciò, quantum-resistant, infatti ad oggi non esistono algoritmi quantistici efficienti per invertire una funzione di hash o trovare delle collisioni: contro questi problemi un computer quantistico può solamente accelerare la risoluzione di un fattore quadratico tramite l'algoritmo di Grover, di conseguenza esso non costituisce un attacco verso questi problemi, dato che è sufficiente aumentare la dimensione delle chiavi di un fattore due per compensare allo "speedup" (ad esempio, passando da SHA-128 a SHA-256).
Inoltre, a differenza di ZK-SNARK esso è un protocollo interattivo basato sul modello "IOP" (*Interactive Oracle Proofs*, può essere visto come un'alternativa ad IP ed altri proof systems come PCP[2]), la cui idea è costruire un sistema di argomenti interattivi in cui è concessa al Verifier la libertà di esaminare probabilisticamente i messaggi del Prover invece che leggerli interamente. Se i valori randomici generati dal Verifier hanno una distribuzione uniforme, cioè ogni risultato è equamente probabile, è possibile rendere il sistema IOP non-interattivo utilizzando l'euristica di Fiat-Shamir [30].

[1] Un ordine di magnitudine è un'approssimazione del logaritmo di un valore relativo a un valore di riferimento (solitamente dieci o due), interpretato come base del logaritmo. Se esso è dieci, l'ordine di grandezza può essere inteso come il numero di cifre, se è una di determinate potenze di due può essere inteso come la quantità di memoria necessaria per memorizzare l'esatto valore intero.

[2] PCP (Probabilistically checkable proofs) è un proof system (oltreché una classe di complessità) in cui il Verifier fa utilizzo della randomness per verificare la correttezza di uno specifico tipo prova senza doverla leggere nella sua interezza.

Conclusioni

La conoscenza-zero rappresenta un ambito smisurato, in continuo studio e sviluppo della crittografia, costituito da innumerevoli protocolli ed applicazioni pratiche degli stessi.
Un esempio fra i più conosciuti e non ancora citato, assieme a SNARK e STARK, è *Bulletproof*, una variante recente di protocollo ZK non-interattivo in cui non è richiesta la fase di trusted setup e con prove di dimensione irrisoria, la cui verifica richiede, però, più tempo rispetto a questi ultimi (tempo lineare invece che sub-lineare) [31].
Le differenze fra questi protocolli fanno si che ognuno di essi abbia un ambito di applicazione diverso; Bulletproof non è ritenuto migliore di ZK-SNARK in contesti come Blockchain a causa della lentezza di generazione e verifica delle prove, ma è ottimo per fornire delle "range proofs", ovvero prove che dimostrano l'appartenenza di un valore all'interno di un certo range.
Anche ZK-STARK, il quale è significativamente più sicuro di ZK-SNARK non è, ad oggi, ritenuto maggiormente adatto in ambito Blockchain, dato che la dimensione delle prove è due ordini di magnitudine più grande.

Per ovvie ragioni gli Zero-Knowledge Protocols non sono stati ancora standardizzati, compito che risulta essere arduo. Esiste però una community, "ZKProof", il cui obbiettivo è proprio quello di racchiudere qualsiasi informazione riguardo l'argomento per giungere alla standardizzazione.
Reputo essa sia un interessante punto di partenza riguardo lo studio della Zero-Knowledge [32].

Bibliografia

[1] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[2] Adi Shamir. Ip= pspace. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

[3] László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429, 1985.

[4] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68, 1986.

[5] László Babai and Shlomo Moran. Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[6] O.A. Ivanova. Cyclic group. encyclopedia of mathematics. URL `http://encyclopediaofmath.org/index.php?title=Cyclic_group&oldid=31609`.

[7] Proof that every group of prime order is cyclic, 1863. URL `https://planetmath.org/proofthateverygroupofprimeorderiscyclic`.

[8] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[9] The Association for Computing Machinery Advancing Computing as a Science & Profession. Goldwasser, micali receive acm turing award for advances in cryptography, mar 2012. URL `https://web.archive.org/web/20130316052703/http://www.acm.org/press-room/news-releases/2013/turing-award-12`.

[10] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[11] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[12] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[13] Juan A Garay, Philip MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 177–194. Springer, 2003.

[14] Jens Groth. *Honest verifier zero-knowledge arguments applied*. PhD thesis, BRICS, 2004.

[15] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 329–349. 2019.

[16] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6): 1084–1118, 1991.

[17] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[18] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, second edition, 2015. Teorema 12.10.

[19] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, second edition, 2015. Teorema 12.11.

[20] Douglas R Stinson. *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005.

[21] Douglas R Stinson. *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005. Pag. 361-362.

[22] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer, 2010.

[23] Michele Ciampi, Giuseppe Persiano, Luisa Siniscalchi, and Ivan Visconti. A transform for nizk almost as efficient and general as the fiat-shamir transform without programmable random oracles. In *Theory of Cryptography Conference*, pages 83–111. Springer, 2016.

[24] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.

[25] Maksym Petkus. Why and how zk-snark works: Definitive explanation.

[26] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. {DIZK}: A distributed zero knowledge proof system. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 675–692, 2018.

[27] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[28] Joseph J Kearney and Carlos A Perez-Delgado. Vulnerability of blockchain technologies to quantum attacks. *Array*, 10:100065, 2021.

[29] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[30] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2016.

[31] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[32] Talia Barta Daniel Benarroch. Zkproof standards. URL `https://zkproof.org/about/`.

Ringraziamenti

La conclusione di questa tesi segna per me la chiusura di un periodo di vita e con con essa voglio ringraziare chi mi ha permesso di arrivare a questo punto:

Ringrazio il mio relatore nonché professore Luciano Margara per avermi fatto appassionare all'argomento con le sue lezioni di crittografia.

Ringrazio i miei genitori per avermi permesso di studiare ed aver portato pazienza tutti questi anni, con le fatiche che gli ho fatto passare questa è anche per loro una bella soddisfazione. Grazie mamma, grazie babbo!

Ringrazio mia sorella per avermi sopportato nelle notti che hanno accompagnato la stesura di questa tesi durante le quali non l'ho lasciata dormire, grazie Chiara!

Ringrazio tutti i miei compagni universitari che mi sono stati vicino, in particolare Michele Faedi e Andrea Pantieri, con i quali ho condiviso questo percorso facendoci da spalla a vicenda.

Ringrazio i miei amici fuori dall'università per aver portato pazienza (e non poca). Troverò il modo di sdebitarmi con tutti ragazzi, promesso!

Ringrazio me stesso, infine, per non aver mollato.
Nessuno ci avrebbe scommesso.