Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica
A.a. 2022/2023
Sessione di Laurea Aprile 2023

Dimostrazioni a Conoscenza Zero nei Protocolli di Identificazione

Relatore:
Carlo Sanna

Candidato:
Chiara Solaro

Sommario

Le dimostrazioni a conoscenza zero sono protocolli interattivi tra due parti, chiamate Prover e Verifier, nei quali il Prover deve convincere il Verifier della validità di una determinata affermazione senza però rivelare alcuna informazione aggiuntiva. Le loro applicazioni in crittografia sono numerose, tra le principali vi sono la tecnologia blockchain, le firme digitali e i protocolli di identificazione, con particolare attenzione alla garanzia della privacy e alla riservatezza delle informazioni.

Il presente elaborato è il risultato di un approfondito studio volto a fornire una panoramica generale del concetto di dimostrazione a conoscenza zero, e una descrizione delle loro applicazioni nei protocolli di identificazione. Lo studio è stato svolto sulla base di numerose pubblicazioni, articoli e libri riguardanti l'argomento.

In seguito ad una introduzione generale dell'argomento, con un primo esempio astratto, vi è la prima parte dell'elaborato, la quale vuole descrivere in modo accurato cosa sono le dimostrazioni a conoscenza zero e le loro proprietà. Il capitolo è introdotto da un breve excursus storico seguito dalla presentazione del concetto di sistema a prova interattivo, necessario alla comprensione delle parti successive. La descrizione vera e propria delle dimostrazioni a conoscenza zero è esposta seguendo l'evolversi dei protocolli nel tempo. Prima vengono definite le dimostrazioni a conoscenza zero sotto forma di protocolli interattivi, ovvero che necessitano dello scambio di messaggi tra le due parti al fine di dimostrare la veridicità di un'affermazione. Tra questi vi sono i Σ-Protocols, un caso particolare di protocolli composti unicamente da tre messaggi e che soddisfano particolari proprietà. In seguito, vengono presentate le dimostrazioni a conoscenza zero non-interattive nei loro modelli: a stringa di riferimento comune e ad oracolo casuale; ed un metodo, noto come Trasformata di Fiat-Shamir, per convertire una dimostrazione a conoscenza zero interattiva in una non-interattiva.

La seconda parte dell'elaborato è incentrata sulla trattazione delle dimostrazioni a conoscenza zero come protocolli di identificazione e l'esposizione di alcuni dei protocolli di questo tipo allo stato dell'arte, evidenziandone criticità e punti di forza.

Indice

Capitolo 1

Introduzione

Le dimostrazioni a conoscenza zero sono protocolli crittografici che permettono ad un soggetto, detto *Prover*, di dimostrare di conoscere un segreto ad un altro soggetto, detto *Verifier*, senza rivelare il segreto stesso. Le loro applicazioni in crittografia sono molto numerose, ad esempio possono essere utilizzati come protocolli di identificazione, nella creazione di firme digitali o nelle transazioni tramite blockchain.
Le dimostrazioni a conoscenza zero rivestono particolare importanza in quanto consentono di mantenere riservate le informazioni personali, siano esse una chiave crittografica, una password o dati riservati quali la data di nascita o il saldo del proprio conto corrente. Il presente elaborato è il risultato di un approfondito studio volto a fornire una panoramica generale del concetto di dimostrazione a conoscenza zero, seguita dalla descrizione delle sue applicazioni nei protocolli di identificazione.

Per comprendere meglio il funzionamento delle dimostrazioni a conoscenza zero è riportato un esempio astratto sviluppato da Quisquater, Guillou e Berson [1] chiamato "La caverna di Ali Baba".
Per chiarezza espositiva si adatta l'esempio alla stessa notazione usata in seguito per tutti gli altri esempi, ovvero viene chiamato Victor il Verifier, colui che si occupa di verificare se l'affermazione sia vera o falsa, e Peggy il Prover, colui che deve dimostrare la veridicità dell'affermazione.

Per spiegare il concetto di dimostrazione a conoscenza zero con semplicità, Quisquater racconta la storia di come Ali Baba viene a conoscenza di una grotta ad anello circolare interrotta da una porta magica che la divide in due sentieri

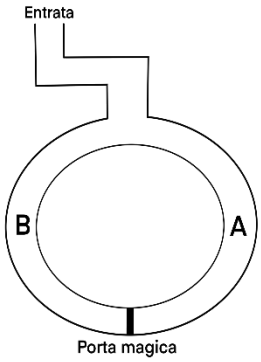

Figura 1.1: La Grotta di Ali Baba

apparentemente senza uscita, e della parola magica per aprirla. Come si può vedere in figura 1.1, dopo essere entrati nella grotta ci si trova a una biforcazione e solo chi conosce la parola magica per aprire la porta riesce a percorrere il giro completo del cerchio.

Nella storia, dopo svariati anni dalla scoperta della caverna, Peggy vuole dimostrare di conoscere la parola magica al reporter Victor, ma senza rivelarla. Per poterlo fare, Peggy procede nel seguente modo: entra nella caverna da sola e sceglie uno dei due percorsi, successivamente Victor con una telecamera entra nella caverna e si ferma alla biforcazione. A quel punto Victor lancia una moneta e in base al risultato grida a Peggy se deve tornare all'entrata dal sentiero A oppure dal sentiero B della caverna. In questo modo, se Peggy conosce la parola magica, può tornare seguendo qualsiasi dei due percorsi venga scelto. Mentre, se non la conosce, ha probabilità del 50% di trovarsi nel percorso richiesto.

Questo procedimento viene ripetuto quaranta[1] volte: ogni volta Peggy entra da sola nella caverna, successivamente entra Victor con la telecamera e dopo aver tirato la moneta dice a Peggy da che lato deve tornare. Ogni tentativo dimezza la probabilità di successo per chiunque non possegga la parola magica, ciò significa che Peggy, che ha avuto successo in tutti i tentativi, conosce realmente la parola magica, con una probabilità prossima a 1 (precisamente $1 - \frac{1}{2^{40}}$).

Questo schema ha anche un'altra funzione, quella di non divulgare al resto del mondo la conoscenza che Peggy è in possesso della parola magica. Infatti, anche se Victor ha registrato il procedimento con la telecamera non significa che sia autentico:

[1] L'esempio cita il numero quaranta in omaggio della storia di Ali Baba e i quaranta ladroni, ma il procedimento funziona con qualunque numero grande a piacere. Più il numero è grande e maggiore sarà la probabilità che Peggy conosca realmente la parola magica.

un altro reporter ha filmato lo stesso procedimento con un'altra persona, la quale tuttavia non è a conoscenza della parola magica e quindi, per svariate volte, sbaglia il percorso; ma il reporter disonesto taglia dalla registrazione tutte le volte che la persona sbaglia, mantenendo solo i successi. In questo modo non è possibile, per una terza persona esterna, dire quale dei due filmati sia autentico e quindi quale persona possegga realmente la parola magica, solo Victor ha la certezza che sia Peggy.
Inoltre, anche se la terza persona esterna fosse fisicamente presente insieme a Victor durante il procedimento, non saprebbe dire se Peggy conosce realmente la parola magica, in quanto potrebbe sospettare che Peggy e Victor si siano accordati in precedenza e, di conseguenza, che l'esperimento non sia autentico.

Questo esempio mostra molto bene come funziona una dimostrazione a conoscenza zero, quali sono le parti coinvolte e le sue principali proprietà. Peggy, il *Prover*, è colui che possiede un determinato segreto, o testimonianza, detto *witness* (la parola magica dell'esempio), che gli permette di dimostrare una determinata affermazione (Peggy è in grado di aprire la porta magica) tramite una prova, o dimostrazione, detta *proof* (Peggy riesce sempre a tornare all'uscita dal sentiero deciso da Victor). Victor, il *Verifier*, è colui che deve decidere se l'affermazione sia vera o falsa tramite una serie di test in successione (il procedimento descritto nell'esempio) per verificare che il Prover possieda realmente il segreto.

Come si può notare nell'esempio, la probabilità che un Prover disonesto riesca a convincere il Verifier, anche se non conosce il segreto, è bassa e diminuisce notevolmente con l'aumentare delle ripetizioni della dimostrazione. Inoltre, si può osservare che per rendere la probabilità di successo nel barare nulla sarebbe sufficiente che Peggy e Victor entrassero nello stesso momento nella grotta e che Peggy mostrasse a Victor che è in grado di entrare in un sentiero, per esempio il sentiero A, e uscire dall'altro, il B.
Così facendo, tuttavia, verrebbe meno l'altra funzione del procedimento, ovvero non divulgare la conoscenza che Peggy possiede la parola magica, in quanto tale dimostrazione convincerebbe chiunque la vedesse e non solo Victor. Quindi non sussisterebbe più la proprietà di conoscenza zero, ovvero l'impossibilità di distinguere la dimostrazione reale da quella simulata (la registrazione del reporter disonesto dell'esempio), proprietà fondamentale delle dimostrazioni a conoscenza zero.

1.1 Struttura della tesi

L'elaborato è composto da tre capitoli principali: il presente capitolo introduttivo, un capitolo incentrato sulla trattazione delle dimostrazioni a conoscenza zero nella loro interezza, e un capitolo riguardante la loro applicazione all'interno dei protocolli di identificazione.

Nello specifico, il capitolo seguente inizia con un breve excursus storico che elenca le principali pubblicazioni riguardanti le dimostrazioni a conoscenza zero, seguito dalla definizione di sistema a dimostrazione interattiva, un concetto necessario alla comprensione della trattazione seguente.
La parte centrale del capitolo è dedicata alla definizione vera e propria delle dimostrazioni a conoscenza zero da un punto di vista matematico. Vengono presentate le dimostrazioni a conoscenza zero sotto forma di protocolli interattivi, ovvero che necessitano dello scambio di messaggi tra le due parti coinvolte, e viene formalizzata la proprietà di conoscenza zero su cui si basano.
Successivamente vengono presentati dei casi particolari di protocolli, i Σ-Protocols (protocolli sigma), composti unicamente da tre messaggi e che soddisfano particolari proprietà. Oltre a presentarne le proprietà più importanti, vengono anche descritte delle metodologie per combinare più protocolli sigma. In seguito, vengono descritte le dimostrazioni a conoscenza zero non-interattive, e un metodo, noto come Trasformata di Fiat-Shamir, per convertire una dimostrazione a conoscenza zero interattiva in una non-interattiva.

Infine, l'ultima parte dell'elaborato è incentrata sull'applicazione delle dimostrazioni a conoscenza zero nei protocolli di identificazione. Vengono prima introdotti i protocolli di identificazione in modo generico, descrivendone le proprietà e alcuni possibili schemi di realizzazione. Successivamente, vengono esposti i miglioramenti che le dimostrazioni a conoscenza zero possono portare nei protocolli di identificazione e vengono presentati tre dei primi protocolli di identificazione a conoscenza zero ad essere stati sviluppati. Infine, il capitolo si conclude con l'esposizione di alcuni dei protocolli più importanti allo stato dell'arte.

A conclusione dell'intero elaborato vi è una piccola sintesi che ripercorre i punti salienti e più importanti della trattazione.

Capitolo 2

Dimostrazioni a conoscenza zero

2.1 Cenni storici

Le dimostrazioni a conoscenza zero vennero definite in modo formale per la prima volta nel 1985 da Shafi Goldwasser, Silvio Micali e Charles Rackoff in una bozza di "*The knowledge complexity of interactive proof-systems*" [2].

In questo documento viene analizzato il modello di dimostrazione a conoscenza zero come un concetto a sé che può presentare diverse applicazioni nei protocolli crittografici. Gli autori hanno come obiettivo il voler comprendere e definire quanta conoscenza è trasmessa ad un verificatore in un sistema di prova interattivo. Nel farlo, definiscono le dimostrazioni a conoscenza zero come quei protocolli che non divulgano alcuna informazione addizionale oltre alla veridicità della prova stessa, anche nel caso in cui il verificatore cerchi di ingannare il dimostratore nel tentativo di far rivelare informazioni aggiuntive.

Insieme al concetto di dimostrazione a conoscenza zero vengono introdotti anche i sistemi a dimostrazione interattiva con la loro classe di complessità IP (*Interactive Polynomial time*), tramite i quali è possibile ridurre drasticamente la conoscenza da dover divulgare al fine di provare un teorema.
Parallelamente, alcuni sistemi di questo tipo erano già stati scoperti, ad esempio il sistema Arthur-Merlin (*Arthur-Merlin games*) introdotto nel 1985 da Babai [3], ma in una forma leggermente diversa, e solo successivamente si dimostrò la loro equivalenza [4].
Nel documento vi è, infine, anche una dimostrazione pratica di una dimostrazione a conoscenza zero applicata al problema di decisione dei residui quadratici e non-residui quadratici.

A partire dalla definizione data da Goldwasser, Micali e Rackoff, vi furono numerosi studi circa l'utilizzo delle dimostrazioni a conoscenza zero in diversi problemi matematici, ad esempio in "*A zero-knowledge proof that a two-prime moduli is not a Blum integer*" [5] a cura di Goldreich e Oded, si illustra l'utilizzo di una dimostrazione a conoscenza zero per verificare che un numero intero con fattori due numeri primi distinti tra loro non sia un intero Blum. La maggior parte degli studi però si concentrarono sul tentativo di dimostrare come le dimostrazioni a conoscenza zero si potessero applicare a diversi domini.
Alcuni dei più importanti risultati in questa direzione sono i seguenti:

- Goldreich, Micali e Wigderson dimostrarono che, sotto l'assunzione dell'esistenza di una crittografia invincibile, è possibile realizzare una dimostrazione a conoscenza zero per il problema NP-completo della colorazione di un grafo a tre colori. Ciò significa che tutti i problemi in NP possono avere dimostrazioni a conoscenza zero, in quanto ciascuno di essi può essere efficientemente ridotto a questo problema [6].
- Partendo dalla stessa assunzione, Impagliazzo e Yung [7], così come Goldreich e i suoi colleghi [8], mostrarono che esistono dimostrazioni a conoscenza zero per tutti i problemi in IP, ovvero tutto ciò che può essere provato con un sistema interattivo può esserlo anche da uno a conoscenza zero.
- Gli studi di Lund e i suoi colleghi [9] e di Shamir [10], invece, permisero di estendere la definizione dei sistemi di prova interattivi, dimostrando, rispettivamente, che ogni linguaggio nella gerarchia tempo-polinomiale ha un sistema di prova interattivo (in particolare anche co-NP), e che IP = PSPACE.
- Goldwasser e colleghi [11] trovarono un'alternativa all'uso delle *one-way function* (utilizzate per avere una crittografia invincibile) utilizzando i sistemi di dimostrazione interattivi multipli, i quali permettono al verificatore di esaminare correttamente il singolo dimostratore tramite diversi dimostratori indipendenti. Utilizzando questo sistema tutti i problemi in NP hanno dimostrazioni a conoscenza zero.

Parallelamente, vi furono ricerche che portarono alla definizione di varianti del protocollo pensato da Goldwasser, Micali e Rackoff; tra queste si citano i protocolli di prova a testimonianza indistinguibile (*witness-indistinguishable proof protocols*) definiti da Feige e Shamir [12], e le dimostrazioni a conoscenza zero non-interattive (*non-interactive zero-knowledge proofs*) introdotti da Blum, Feldman, e Micali [13].

Infine, per quanto concerne il campo della crittografia, ci furono numerosi studi sull'utilizzo delle dimostrazioni a conoscenza zero, i quali continuano ancora oggi. Tra i più rilevanti vi è lo studio di Goldreich [6], il quale introdusse le basi per la creazione di algoritmi per la generazione automatica di protocolli crittografici a due parti e multi-parti, di cui troviamo degli esempi nelle pubblicazioni di Yao [14] e di Goldreich e Micali [15]. Altrettanto importante, il protocollo di Fiat-Shamir [16] fu il primo schema di identificazione basato sulle dimostrazioni a conoscenza zero, al quale susseguirono numerosi altri, come gli schemi di identificazione di Schnorr e di Guillou-Quisquater [17] che ne hanno mantenuto la stessa struttura.

2.1.1 Sistemi a dimostrazione interattiva

Per capire al meglio il concetto di dimostrazione a conoscenza zero, è bene introdurre prima alcune definizioni sulla quale esso si basa.
Goldwasser, Micali e Rackoff [2] definiscono le dimostrazioni a conoscenza zero come sistemi di prova interattiva, ovvero protocolli nei quali due entità interagiscono tra loro al fine di dimostrare la validità di un'affermazione.

Un sistema a dimostrazione interattiva è composto da due parti: il *Prover* (dimostratore), una macchina con infinito potere computazionale con lo scopo di convincere il Verifier della validità della sua affermazione, ed il *Verifier* (verificatore), una macchina probabilistica con complessità polinomiale con lo scopo di verificare se l'affermazione del Prover sia vera o falsa, e che non può essere ingannato. L'interazione tra Prover e Verifier avviene tramite lo scambio di messaggi, e l'insieme di tutti i messaggi scambiati, chiamato *round*, può essere ripetuto più volte per aumentare la sicurezza dell'algoritmo.

Spesso il Verifier e il Prover vengono modellati come una macchina di Turing interattiva, un modello di calcolo introdotto nel 1936 da Alan Turing che viene spesso utilizzato per definire in modo formale un algoritmo.
Le macchine di Turing (*Turing Machines*) [18] [19] possono essere viste come astrazioni dei normali computer: esse utilizzano una lista di istruzioni elementari, le quali rappresentano degli algoritmi, per svolgere dei compiti, per esempio risoluzioni di funzioni matematiche. Esse sono composte da diversi nastri:

- Nastro di conoscenza o di input (*knowledge tape*): è la parte di memoria nella quale vengono inseriti tutti gli input privati.

- Nastro ausiliario (*auxiliary tape*): è utilizzato dalla macchina per scrivere e leggere risultati intermedi delle computazioni.
- Nastro di output (*output tape*): è il nastro nel quale vengono scritti i risultati delle computazioni una volta terminate.
- Nastro di comunicazione (*communication tape*): è utilizzato per permettere a più macchine di Turing di interagire tra loro scrivendo e leggendo i valori su questo nastro.
- Nastro casuale (*random tape*): è una parte di memoria utilizzata per immagazzinare bit casuali impiegabili dalla macchina per svolgere computazioni che richiedono valori random. Una macchina di Turing che possiede questo nastro è detta probabilistica.

Una macchina di Turing è detta essere a tempo polinomiale (o efficiente), se il numero totale delle operazioni di lettura e scrittura sui nastri è limitato da un polinomio nella lunghezza dell'input. I problemi che non possono essere risolti da una macchina di Turing a tempo polinomiale sono definiti intrattabili (*intractable* o *infeasible*) e, tipicamente, sono alla base di numerosi algoritmi crittografici. In particolare, la difficoltà nella loro risoluzione li rende perfetti per la realizzazione di funzioni che non devono essere invertibili, ovvero non è possibile risalire alle variabili dalle quali un determinato risultato è stato ottenuto.

Definite le macchine di Turing si può affermare che, tipicamente, il Verifier è modellato con una macchina di Turing interattiva probabilistica a tempo polinomiale (PPT), mentre il Prover è modellato con una macchina di Turing con infinito potere computazionale. Dimostratore e verificatore condividono lo stesso nastro di conoscenza in sola lettura (nessuno dei due può scrivere, modificare o eliminare dei dati di input scritti su di esso), il quale contiene l'affermazione da verificare, e lo stesso nastro di comunicazione (in modo tale che ogni macchina possa leggere e scrivere dati su di esso, ma non possa eliminare o modificare i dati dopo che vengono scritti).

Affinché un sistema sia definibile a dimostrazione interattiva deve soddisfare le seguenti proprietà:

- Completezza (*completeness*): la probabilità che un Verifier accetti una dimostrazione, se l'affermazione è vera, alla fine dell'interazione con il Prover è quasi pari a 1. Se la probabilità è esattamente 1 si parla di completezza perfetta (*perfect completeness*).

- Correttezza (*soundness*): la probabilità che un Verifier accetti una dimostrazione falsa, alla fine dell'interazione con un Prover disonesto è trascurabile[2]. Se la probabilità è esattamente zero si parla di correttezza perfetta (*perfect soundness*).

Un Prover si definisce disonesto se cerca di convincere il Verifier che un'affermazione è vera, anche se in realtà non lo è o non possiede alcuna conoscenza per dimostrarne la veridicità. Anche il Verifier può essere disonesto, e accade quando non segue il corretto svolgimento del protocollo e cerca di ottenere informazioni aggiuntive da parte del Prover. Dimostratore e verificatore che si comportano seguendo le specifiche nel protocollo sono detti onesti.

Con i sistemi di prova interattivi nasce la classe di complessità IP (*Interactive Polynomial-Time*), definita come l'insieme di tutti i problemi risolvibili tramite un sistema di prova interattivo [2].

2.2 Dimostrazioni a conoscenza zero interattive

Una dimostrazione a conoscenza zero (*Zero-Knowledge Proof*) è un protocollo nel quale un Prover (P) vuole convincere un Verifier (V) della validità di un'affermazione y, senza divulgare alcuna informazione addizionale oltre alla veridicità dell'affermazione.
Le dimostrazioni a conoscenza zero nascono come sistemi a dimostrazione interattiva [2] e, in quanto tali, rispettano le proprietà di completezza e correttezza descritte precedentemente, con l'aggiunta di una nuova proprietà:

- Conoscenza Zero (*zero-knowledge*): dall'esecuzione del protocollo interattivo il Verifier non è in grado, con nessuna strategia, di ricavare alcuna informazione che non possa essere calcolata anche senza l'esecuzione del protocollo, tranne la veridicità dell'affermazione.

[2] Con probabilità trascurabile si intende un evento molto raro. Formalmente [53], una funzione μ si considera trascurabile se svanisce più velocemente del reciproco di ogni polinomio: per ogni polinomio positivo p e un sufficientemente grande n, significa che $\mu(n) < 1/p(n)$.

> Detto diversamente, un sistema di prova interattivo (P, V) è a conoscenza zero se tutto ciò che può essere calcolato in modo efficiente sull'input y dopo aver interagito con P, può anche essere calcolato in modo efficiente da y senza alcuna interazione.

Si può notare che ciò corrisponde a quanto è stato descritto nell'esempio iniziale della "Caverna di Ali Baba", dove il reporter disonesto è stato in grado di ricreare l'esperimento di Peggy, senza conoscere il segreto ed interagire con Peggy, rendendo il risultato indistinguibile da quello originale.
La definizione formale delle dimostrazioni a conoscenza zero è la seguente [20]:

> Sia (P, V) un sistema di prova interattivo per un linguaggio[3] L. Si dice che (P, V) è a *conoscenza zero* se per ogni macchina probabilistica interattiva a tempo polinomiale V^* esiste un probabilistico algoritmo a tempo polinomiale S^* tale per cui, per ogni $y \in L$, le due seguenti variabili casuali sono distribuite in modo identico:
>
> - $\langle P, V^* \rangle (y)$ – ovvero l'output della macchina interattiva V^* dopo aver interagito con la macchina interattiva P sull'input comune y
> - $S^*(y)$ – ovvero l'output della macchina S^* sull'input y
>
> La macchina S^* è detta *simulatore* per l'interazione di V^* con P.

Un simulatore è una macchina di Turing probabilistica a tempo polinomiale che è in grado di simulare l'interazione tra V^* e P senza avere accesso a P. L'output del simulatore è la trascrizione del protocollo, ovvero la sequenza di messaggi scambiati tra P e V^*. Il simulatore è volutamente non onesto e non possiede alcun tipo di conoscenza, nonostante ciò, la trascrizione che produce risulta indistinguibile da quella reale. Di conseguenza, l'esistenza del simulatore è la dimostrazione che V^* non può guadagnare alcuna informazione aggiuntiva interagendo con P, in quanto tutto ciò che V^* può imparare dall'interazione può essere imparato anche eseguendo il simulatore sullo stesso input. Questo è vero in quanto qualunque informazione una macchina è in grado di ricavare da sola non può essere considerata un guadagno di informazione ottenibile dall'interazione con un altro soggetto.

[3] Con linguaggio si intende l'insieme di elementi costruiti utilizzando un determinato alfabeto. Nella crittografia si intende l'insieme di tutte le stringhe per le quali l'applicazione di una procedura decisionale produce una risposta affermativa. In questo caso specifico, il linguaggio L corrisponde al set delle affermazioni y che sono in relazione R con un segreto x, formalmente:

$$L = \{y \mid \exists x : (y, x) \in R\}$$

Per dimostrare, quindi, che un protocollo è a conoscenza zero, è sufficiente dimostrare che esiste un simulatore per quel protocollo.
Si osservi che la notazione V^* indica che il simulatore S^* esiste per ogni verificatore che interagisce con P, e non solo per V, questo significa che il simulatore esiste per qualsiasi modo efficiente di interagire con P, anche disonesto, non necessariamente il modo definito da V.

Per creare la trascrizione il simulatore deve essere in grado di ingannare il Verifier e fargli credere di possedere il segreto. Ciò sembrerebbe andare in contrasto con la proprietà di correttezza, ma non è così, in quanto, per poterlo fare, il simulatore deve avere accesso al codice del Verifier, in questo modo è in grado di ripetere parti dell'algoritmo o controllare i valori casuali utilizzati. Questo significa che, durante una normale esecuzione del protocollo, l'algoritmo usato dal simulatore per ingannare il Verifier non è utilizzabile e la proprietà di correttezza è preservata.

La definizione di dimostrazioni a conoscenza zero, tuttavia, risulta troppo restrittiva, ed esclude molti casi reali nei quali il simulatore ha probabilità, seppur ridotta, di fallire nel riprodurre l'interazione. Dunque, in base al grado di indistinguibilità della trascrizione che il simulatore riesce a riprodurre e quella reale si definiscono [19] tre diverse tipologie di conoscenza zero:

- Conoscenza zero perfetta (*perfect zero-knowledge*): la differenza tra la trascrizione del simulatore e quella reale è nulla, ovvero sono completamente indistinguibili. Corrisponde alla definizione data precedentemente, e nemmeno un Verifier con potenza computazionale illimitata è in grado di ricavare qualcosa dalla comunicazione.
- Conoscenza zero statica (*static zero-knowledge*): la differenza tra la trascrizione del simulatore e quella reale è statisticamente indistinguibile, ovvero esse non sono necessariamente uguali, ma statisticamente vicine. Un Verifier con potenza computazionale illimitata non può ricavare nulla dalla comunicazione eccetto con probabilità trascurabile.
- Conoscenza zero algoritmica o computazionale (*computational zero-knowledge*): la differenza tra la trascrizione del simulatore e quella reale è computazionalmente indistinguibile. Ciò significa che non vi è alcun algoritmo probabilistico efficiente (in tempo polinomiale all'input dato) capace di distinguere le due trascrizioni.

Nelle dimostrazioni a conoscenza zero interattive, il protocollo utilizzato consiste in una serie di messaggi scambiati tra le due parti; tipicamente il primo messaggio è inviato dal Prover, mentre l'ultimo è inviato dal Verifier, il quale dichiara se la prova è stata accettata o rifiutata. L'affermazione y che vuole dimostrare il Prover è legata alla testimonianza, o segreto x, tramite una relazione R, tipicamente un problema intrattabile che consente di calcolare y a partire da x, ma non viceversa. La dimostrazione a conoscenza zero viene dunque definita una dimostrazione per la relazione $R(y, x)$.

Di seguito viene riportato un esempio [21] riassuntivo dei concetti visti fino ad ora che consiste nel ricreare un semplice protocollo di identificazione a conoscenza zero, detto *Schnorr identification protocol*.

Peggy (il dimostratore) vuole dimostrare a Victor (il verificatore) che conosce l'esponente x di $y = g^x \ mod \ p$ senza rivelare x. Come primo approccio si può provare a nascondere il valore crittografandolo, ma allo stesso tempo bisogna anche dimostrare che il valore è presente. Il modo più semplice è aggiungere al segreto un valore k generato casualmente:

$$z = k + x$$

In questo modo, dopo che Peggy ha inviato z e k, Victor potrebbe confermare la presenza della testimonianza semplicemente verificando che le seguenti equazioni siano uguali:

$$g^z = y \times g^k = g^{x+k}$$

Tuttavia, il protocollo svolto in questo modo non è sicuro, infatti l'equazione che dovrebbe nascondere x può facilmente essere invertita per ricavarne la testimonianza:

$$x = z - k$$

Per evitare che Victor possa ricavare x invertendo l'equazione di k, prima di inviare il valore, Peggy deve nasconderlo come esponente, in modo che l'equazione di

Victor sia ancora valida, ma k non possa essere ricavato (problema del logaritmo discreto, ovvero un problema intrattabile):

$$u = g^k$$

In questo modo, però, Victor non ha il controllo sull'onestà di Peggy; infatti, Peggy potrebbe ricavare u senza conoscere x, ma utilizzando l'inverso della formula, ovvero generando prima z in modo casuale e poi calcolando u di conseguenza:

$$u = g^z \times y^{-1}$$

Per essere sicuro che Peggy calcoli z a partire da u e non il contrario, Victor deve rendere il protocollo interattivo:

1. Peggy deve inviare il proprio valore casuale k in modo che non possa essere cambiato in seguito:

$$u = g^k$$

2. Ricevuto il valore di Peggy, Victor può generare un valore casuale c (detto *challenge*) e inviarlo a Peggy
3. Peggy può ora calcolare il suo valore z da inviare, sulla base del valore casuale k e la challenge c:

$$z = k + c \times x$$

Victor può facilmente affermare che Peggy è in possesso di x verificando che:

$$g^z = y^c \times u$$

Il protocollo così ottenuto è il Protocollo di identificazione di Schnorr, una dimostrazione a conoscenza zero interattiva che segue uno schema a tre mosse (*commitment*, *challenge* e *proof*) definito protocollo Sigma (*Σ-Protocol*).

Questo protocollo, tuttavia, non soddisfa a pieno la proprietà di conoscenza zero; infatti, per essere a conoscenza zero bisogna assumere che il Verifier si comporti in modo onesto e scelga una challenge c in modo casuale, altrimenti potrebbe

essere in grado di ricavare delle informazioni su x. Protocolli di questo tipo devono soddisfare la proprietà di "*Honest-verifier zero-knowledge*" (HVZK), ovvero il Verifier deve comportarsi in modo onesto seguendo il protocollo, in questo caso specifico, scegliendo c casuale.

Se è presente la proprietà di HVZK la definizione di conoscenza zero diventa più debole e richiede l'esistenza del simulatore per un unico verificatore, quello onesto descritto nelle specifiche del protocollo V.
L'aggiunta di questa proprietà rende l'algoritmo meno realistico in quanto non vengono presi in considerazione i comportamenti malevoli atti ad ottenere informazioni da parte del Verifier. Tuttavia, questa semplificazione permette di studiare e perfezionare nuovi algoritmi in modo più semplice, i quali, spesso, risultano più efficienti rispetto alle altre dimostrazioni a conoscenza zero.
Inoltre, successivamente è possibile convertire il protocollo HVZK in uno a conoscenza zero in modo efficiente, a leggero discapito della comunicazione e della computazione, alle quali va aggiunto un costo costante [22].

2.2.1 Σ-Protocol

Un Σ-Protocol (protocollo sigma) è un caso speciale di dimostrazione a conoscenza zero interattiva costituito da tre messaggi scambiati tra Prover e Verifier:

1. Il Prover invia il primo messaggio u detto *commitment*, il quale è, tipicamente, ricavato dall'esecuzione di un algoritmo sui valori in possesso del Prover: il segreto o testimonianza x, l'affermazione y e un numero generato in modo casuale k.
2. Il Verifier genera un numero casuale c detto *challenge* e lo invia come secondo messaggio.
3. Il Prover genera una risposta z detta *proof* tramite un algoritmo che, tipicamente, prende in input il segreto x, l'affermazione y, il commitment u, il valore casuale k e la challenge c. La risposta è inviata come terzo e ultimo messaggio del protocollo.

Infine, è richiesto che il Verifier accetti o rifiuti la risposta del Prover.

I Σ-Protocol vengono identificati tramite la terna (u, c, z) dei messaggi scambiati, e per essere tali devono soddisfare le seguenti proprietà:

- Completezza perfetta (*perfect completeness*): la probabilità che un Verifier accetti una dimostrazione, se l'affermazione è vera, alla fine dell'interazione con il Prover, è pari a 1.
- Correttezza speciale (*special soundness*): esiste un estrattore efficiente E che, dato un qualunque input y e una coppia di trascrizioni valide (u, c, z), (u, c', z'), con la challenge $c \neq c'$, è sempre in grado di calcolare una testimonianza x corrispondente a y.

  Un estrattore di conoscenza consiste in un algoritmo in grado di estrarre la prova x da due trascrizioni valide con il commitment u in comune, e due challenge differenti. Per poter funzionare l'estrattore deve avere accesso all'algoritmo del Prover durante la procedura di estrazione, la quale tipicamente consiste in una tecnica detta riavvolgimento[4] (*rewinding*). L'esistenza dell'estrattore dimostra che il Prover conosce realmente una testimonianza x che corrisponde all'affermazione y.
  Inoltre, questa proprietà implica che un Prover disonesto ha successo con una probabilità di al massimo $1/n$, dove n indica la cardinalità dello spazio delle challenge [23] [24].

  Potrebbe sembrare che l'esistenza dell'estrattore vada in contraddizione con la proprietà di conoscenza zero, in quando essa è basata sul principio che nessuna informazione può essere ottenuta dalla trascrizione del protocollo, mentre l'estrattore è in grado di estrarre la testimonianza dall'algoritmo. La differenza sostanziale è che l'estrattore ha accesso a più informazioni iniziali rispetto al Verifier, per esempio il codice del Prover, dunque, durante la normale esecuzione del protocollo, l'algoritmo dell'estrattore non è utilizzabile.
- *Special Honest-Verifier Zero-Knowledge* (SHVZK): esiste un simulatore M a tempo polinomiale, il quale prende come input y (lo stesso input di P e V) e il numero casuale c, e genera come output una trascrizione valida nella forma (u, c, z) con la stessa distribuzione di probabilità della conversazione reale tra P e il verificatore onesto V su y [23].

[4] Il riavvolgimento consiste nell'eseguire il codice del Prover utilizzando l'input che richiede il Verifier per ottenere una prima risposta, e poi riavvolgere l'algoritmo ad uno stato precedente per utilizzare un input differente ed ottenere una seconda risposta. In questo modo, l'estrattore è in grado di ottenere differenti output del Prover legati a differenti input del Verifier, ma con uno stesso stato iniziale. Questo permette di cancellare parte della casualità aggiunta dal Prover per nascondere la testimonianza e procedere così all'estrazione di x [54].

L'ultima proprietà significa che i Σ-Protocol possono essere a conoscenza zero solamente con un Verifier onesto, che segue correttamente il protocollo e non cerca di ottenere ulteriori informazioni dal Prover, per esempio utilizzando una challenge non realmente random. Nonostante questa limitazione, i protocolli sigma possono essere molto utili come strumenti per costruire altri protocolli, infatti, anche in questo caso possono essere convertiti in dimostrazioni a conoscenza zero generiche [22] [25].

Oltre alla proprietà di SHVZK, viene modificata anche la proprietà di correttezza, che diventa più restrittiva. Infatti, nei protocolli sigma, è richiesto che il Prover "conosca" un segreto per dimostrare il protocollo, mentre nella definizione di dimostrazioni a conoscenza zero veniva richiesta solamente l'appartenenza del segreto ad un linguaggio $(y \in L)$. Ciò significa che nelle dimostrazioni a conoscenza zero un dimostratore potrebbe provare che y appartiene al linguaggio anche senza conoscere la testimonianza correlata x, se è in grado di inviare i messaggi corretti che lo dimostrino. Invece, nei protocolli sigma il Prover deve dimostrare di conoscere la testimonianza x, ed è da questa necessità che nasce l'estrattore. Furono Bellare e Goldreich [26] i primi a introdurre l'estrattore e ad utilizzarlo per definire il concetto di conoscenza, concludendo che se un estrattore è in grado di estrarre il segreto x significa che il Prover lo conosce veramente.

I protocolli interattivi che possiedono un estrattore di conoscenza, e che quindi sono in grado di dimostrare ad un Verifier che il Prover conosce un segreto, sono detti prove di conoscenza (*Proof of Knowledge*), e sono indipendenti dalla proprietà di conoscenza zero (una prova di conoscenza può essere o meno una dimostrzione a conoscenza zero, e viceversa). Le prove di conoscenza risultano particolarmente utili nella costruzione dei protocolli di identificazione, nei quali la conoscenza di un segreto è la prova necessaria a identificare un utente.
I Σ-Protocol, dunque, sono sistemi di prova di conoscenza interattivi a conoscenza zero.

Per comprendere meglio le proprietà appena descritte e, soprattutto, il concetto di estrattore, si verifica di seguito la loro presenza nel protocollo di identificazione di Schnorr precedentemente descritto.

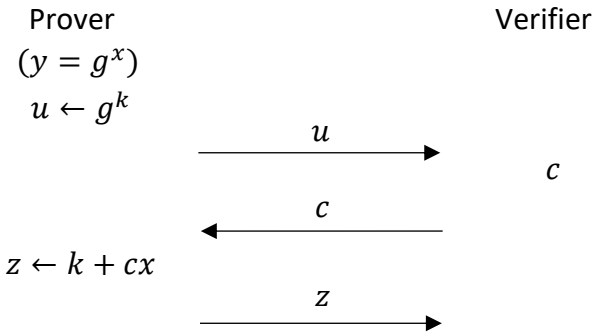

Figura 2.1: Protocollo di Schnorr

- Completezza perfetta:
  Se l'affermazione è vera, il Verifier accetta la dimostrazione. Questo corrisponde alla verifica che svolge il Verifier:

$$g^z = y^c \times u$$

  Con $z = k + c \times x,\ y = g^x, u = g^k$ abbiamo:

$$g^{k+c\times x} = g^{c\times x} \times g^k$$

  Le quali sono equivalenti.

- Correttezza speciale:
  Per verificare la proprietà, un estrattore deve essere in grado di estrarre la testimonianza, in questo caso la variabile x. L'estrattore è in possesso del codice del Prover, quindi è in grado di far eseguire i passaggi del protocollo più volte. Sfruttando questa possibilità esegue i seguenti passaggi:

  1. Il Prover genera il numero casuale k e lo invia al Verifier come

$$u = g^k$$

  2. Il Verifier genera a sua volta un numero casuale, la challenge c e la comunica al Prover
  3. Il Prover calcola $z = k + c \times x$ e lo invia al Verifier seguendo il protocollo

4. A questo punto l'estrattore può riavvolgere il protocollo e farlo tornare al punto 2 in modo che il Verifier generi un nuovo numero casuale c' e lo invii al Prover
5. Il Prover, dunque, segue il protocollo e genera un nuovo valore per la challenge c'

$$z' = k + c' \times x$$

6. L'estrattore prende i due valori appena generati e li utilizza per estrarre la testimonianza x, infatti, grazie alla procedura di riavvolgimento è riuscito ad ottenere due trascrizioni con lo stesso commitment, in questo caso u, e due challenge differenti: $(u, c, z), (u, c', z')$.
   Da queste si ottiene x nel seguente modo:

$$(z - z')/(c - c') =$$
$$((k + c \times x) - (k + c' \times x))/(c - c') =$$
$$x\,(c - c')/(c - c') =$$
$$x$$

L'esistenza dell'estrattore in grado di ricavare x è la dimostrazione che il Prover possiede realmente la testimonianza.
Per ricavare il segreto l'estrattore utilizza due trascrizioni con il commitment in comune e le challenge differenti tra loro. Questa coppia di trascrizioni è detta collisione (*collision*) e bisogna prestare attenzione al fatto che, come l'estrattore è stato in grado di ottenere la testimonianza da una collisione, chiunque potrebbe farlo. Dunque, è importante che lo stesso valore di k non venga mai usato in due diverse esecuzioni dell'algoritmo (con due challenge differenti); ciò significa, nella pratica, che il suo riutilizzo debba essere altamente improbabile.

La proprietà che permette di estrarre la testimonianza da una collisione del protocollo è detta proprietà di collisione (*collision-property*) ed è espressa in modo formale [18] come:

> Data una collisione per l'input y, è possibile calcolare in modo efficiente una testimonianza x tale che $(y, x) \in R$, ovvero esiste

una macchina di Turing PPT che, preso in input una collisione per y, restituisce in output il segreto x tale che $(y, x) \in R$.

Dunque, si può dire che un protocollo soddisfa la proprietà di correttezza speciale, se e solo se possiede la proprietà di collisione.

- SHVZK:
  Essendo in presenza di verificatore onesto, nella creazione del simulatore bisogna considerare che il Verifier segua il protocollo correttamente. In questo caso, il verificatore deve scegliere la challenge c in modo totalmente casuale. Come nel caso dell'estrattore, anche il simulatore possiede una conoscenza aggiuntiva, per esempio il codice del Prover, in modo da poter usare la tecnica del riavvolgimento:

  1. Il Prover genera un numero casuale e lo invia al Verifier $u = g^k$
  2. Il Verifier genera la challenge casuale c e la invia al Prover
  3. Il simulatore può riavvolgere l'algoritmo e tornare al punto 1, dunque il Prover genera un nuovo numero casuale z e un intero malevolo k' tale che:

     $$u' = g^{k'} = g^z \times g^{x(-c)}$$

     e lo invia al Verifier
  4. Il Verifier genera lo stesso numero casuale c e lo invia al Prover
  5. Il Prover, a questo punto, ha ingannato il Verifier e conosce già la risposta, avendo scelto k' in modo opportuno; quindi, può inviare direttamente z al Verifier
  6. Il Verifier verificherà che

     $$g^z = g^{xc} \times g^z \times g^{x(-c)}$$

  Di conseguenza, la trascrizione (u', c, z) risulta valida per il Verifier, al pari di qualunque altra trascrizione, anche se il simulatore non conosce la testimonianza. Dunque, la proprietà è rispettata.

La sicurezza del protocollo di identificazione di Schnorr si basa sulla difficoltà di calcolare il logaritmo discreto, il quale è un problema intrattabile con l'utilizzo di

una Macchina di Turing; tuttavia, l'evoluzione della tecnologia fa sì che potrebbe, in futuro, essere risolto in un tempo drasticamente minore, tanto da non essere più considerabile intrattabile. Infatti, in crittografia il concetto di sicurezza è mutevole: un algoritmo è considerato sicuro finché non si dimostra che i modelli matematici su cui si basa sono falsi. Per questo motivo è bene riconsiderare periodicamente la sicurezza degli algoritmi, per assicurarsi che la forza dei modelli matematici utilizzati non sia cambiata, per esempio perché è stato trovato un algoritmo risolutivo di tempo polinomiale, oppure i calcolatori sono diventati esponenzialmente più veloci nella computazione.

2.2.2 Composizione dei Σ-Protocol

Un'ulteriore proprietà dei Σ-Protocol è quella di invarianza nei confronti della composizione parallela, ovvero ripetendo un protocollo sigma due volte in parallelo si ottiene un nuovo Σ-Protocol con lunghezza della challenge doppia [23].

Questa proprietà permette di combinare tra loro i protocolli sigma per dimostrare più affermazioni contemporaneamente o aumentare la lunghezza della challenge. Per eseguire il protocollo correttamente l'esecuzione delle istanze parallele deve avvenire in modo sincrono, dunque il singolo messaggio i-esimo deve essere inviato nello stesso momento da entrambe le istanze. Si noti che la possibilità di utilizzare la composizione parallela non è comune a tutte le dimostrazioni a conoscenza zero; infatti, soltanto alcuni protocolli mantengono invariate le loro proprietà quando vengono eseguiti in parallelo.

Vi sono vari modi per combinare tra loro i protocolli, i quali dipendono dal risultato che si vuole ottenere. Mantenendo come protocollo di base il protocollo di Schnorr, si descrivono di seguito cinque tipologie di composizione dei protocolli sigma e si verificano le loro proprietà [24].

Composizione parallela

Eseguendo due istanze dello stesso Σ-Protocol in parallelo al fine di dimostrare un'affermazione x, il risultato è un Σ-Protocol con lo stesso scopo, ma con uno spazio delle challenge più grande.

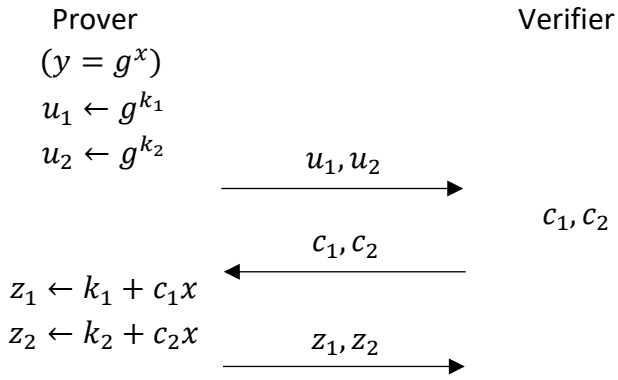

Figura 2.2: Composizione parallela del protocollo di Schnorr

Per verificare il successo dell'algoritmo e la presenza della proprietà di completezza, il Verifier esegue le seguenti verifiche:

$$g^{z_1} = g^{k_1 + c_1 x} = g^{k_1}(g^x)^{c_1} = u_1 y^{c_1}$$

$$g^{z_2} = g^{k_2 + c_2 x} = g^{k_2}(g^x)^{c_2} = u_2 y^{c_2}$$

Per dimostrare la proprietà di correttezza speciale, invece, è necessario l'uso di un estrattore, tramite il quale è possibile ricavare due trascrizioni valide con challenge differenti $(c_1, c_2) \neq (c_1', c_2')$, ma commitment in comune:

$$(u_1, u_2;\ c_1, c_2;\ z_1, z_2), (u_2, u_2;\ c_1', c_2';\ z_1', z_2')$$

Grazie alle due trascrizioni è possibile ricavare la testimonianza segreta x con due differenti metodi a seconda delle challenge, ovvero, se $c_1 \neq c'_1$:

$$g^{z_1} = u_1 y^{c_1}, \qquad g^{z_1'} = u_1 y^{c_1'}$$

$$g^{z_1 - z_1'} = y^{c_1 - c_1'}$$

$$y = g^{\frac{z_1 - z_1'}{c_1 - c_1'}} \implies x = \frac{z_1 - z_1'}{c_1 - c_1'}$$

Mentre, se $c_2 \neq c_2'$ la dimostrazione è speculare e il valore x si ottiene come:

$$x = \frac{z_2 - z_2'}{c_2 - c_2'}$$

Infine, la proprietà di SHVZK è verificata con l'uso di un simulatore, in grado di ricavare due trascrizioni del protocollo con la stessa coppia di challenge (c_1, c_2) e con esse dimostrare l'esistenza di una conversazione simulata con distribuzione identica a quella reale:

$$\{(u_1, u_2;\ c_1, c_2;\ z_1, z_2) :\ u_1 \leftarrow g^{k_1};\ u_2 \leftarrow g^{k_2};\ z_1 \leftarrow k_1 + c_1 x;\ z_2 \leftarrow k_2 + c_2 x\}$$

$$\{(u_1, u_2;\ c_1, c_2;\ z_1, z_2) :\ u_1 \leftarrow g^{z_1} y^{-c_1};\ u_2 \leftarrow g^{z_2} y^{-c_2}\}$$

Composizione AND

Date due coppie distinte di affermazione e testimonianza, è possibile ottenere un Σ-Protocol in grado di dimostrare la veridicità della congiunzione (AND) delle due affermazioni eseguendo i rispettivi Σ-Protocol in parallelo, ed usando una challenge comune ad entrambi.

La composizione AND risulta quindi un caso speciale di composizione parallela, nella quale entrambe le istanze eseguite in parallelo usano la stessa challenge c. Il protocollo verrà eseguito nel seguente modo: il Prover invierà due diversi commitment al Verifier, uno per ogni istanza, ed il Verifier risponderà con un'unica challenge. In seguito, dopo aver ricevuto la challenge, il Prover calcolerà ed invierà due risultati differenti, e solamente se entrambi risulteranno corretti la dimostrazione verrà accettata. In questo modo con un'unica challenge è possibile dimostrare due affermazioni.

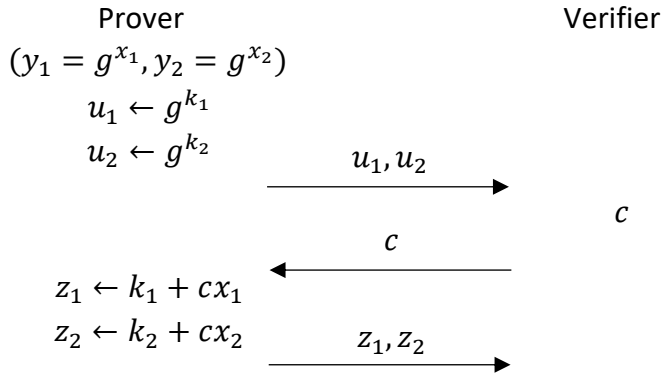

Figura 2.3: Composizione AND del protocollo di Schnorr

Di seguito si verificano le proprietà dei protocolli sigma per la composizione AND del protocollo di Schnorr. La proprietà di completezza è verificata in quanto:

$$g^{z_1} = g^{k_1+cx_1} = g^{k_1}(g^{x_1})^c = u_1 {y_1}^c$$

$$g^{z_2} = g^{k_2+cx_2} = g^{k_2}(g^{x_2})^c = u_2 {y_2}^c$$

La proprietà di correttezza speciale si verifica in modo similare al caso precedente, partendo da due trascrizioni del protocollo con i commitment in comune, e le challenge differenti tra loro $(u_1, u_2; c; r_1, r_2)$ e $(u_1, u_2; c'; r_1', r_2')$.

$$g^{z_1} = u_1 {y_1}^c,\ g^{z_1'} = u_1 {y_1}^{c'}$$

$$g^{z_1 - z_1'} = {y_1}^{c-c'}$$

$$y_1 = g^{\frac{z_1 - z_1'}{c-c'}} \implies x_1 = \frac{z_1 - z_1'}{c-c'}$$

In modo simmetrico è possibile estrarre x_2.

Infine, la proprietà di SHVZK è dimostrata dall'esistenza di due trascrizioni, una reale e l'altra simulata, con la stessa distribuzione:

$$\{(u_1, u_2;\ c;\ z_1, z_2) :\ u_1 \leftarrow g^{k_1};\ u_2 \leftarrow g^{k_2};\ z_1 \leftarrow k_1 + cx_1;\ z_2 \leftarrow k_2 + cx_2\}$$

$$\{(u_1, u_2;\ c;\ z_1, z_2) :\ u_1 \leftarrow g^{z_1} {y_1}^{-c};\ u_2 \leftarrow g^{z_2} {y_2}^{-c}\}$$

Composizione EQ

La composizione EQ corrisponde ad un caso speciale di composizione AND, nel quale la testimonianza delle due diverse affermazioni che si vogliono provare è la stessa. Ovvero, applicando questo concetto al protocollo di Schnorr si ottengono $y_1 = {g_1}^x$ e $y_2 = {g_2}^x$.

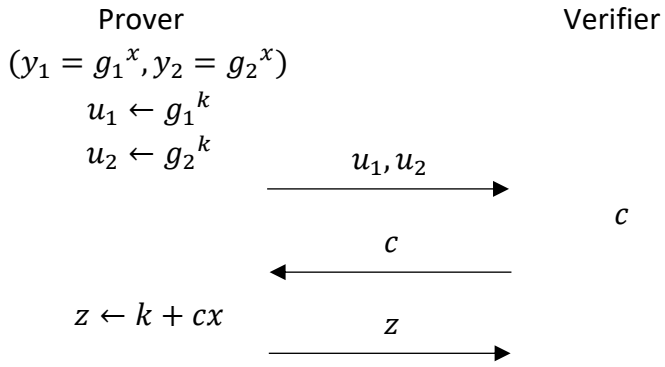

Figura 2.4: Composizione EQ del protocollo di Schnorr

La verifica delle proprietà si svolge in modo similare alle precedenti. La proprietà di completezza è dimostrata verificando che:

$$ {g_1}^z = {g_1}^{k+cx} = {g_1}^k ({g_1}^x)^c = u_1 {y_1}^c $$

$$ {g_2}^z = {g_2}^{k+cx} = {g_2}^k ({g_2}^x)^c = u_2 {y_2}^c $$

La proprietà di correttezza speciale si verifica a partire da due trascrizioni con i commitment in comune $(u_1, u_2; c; z)$ e $(u_1, u_2; c'; z')$:

$$ {g_1}^z = u_1 {y_1}^c,\ {g_2}^z = u_2 {y_2}^c,\ {g_1}^{z'} = u_1 {y_1}^{c'},\ {g_2}^{z'} = u_2 {y_2}^{c'} $$

$$ {g_1}^{z-z'} = {y_1}^{c-c'},\ {g_2}^{z-z'} = {y_2}^{c-c'} $$

$$ y_1 = {g_1}^{\frac{z_1 - z_1'}{c-c'}},\ y_2 = {g_2}^{\frac{z_1 - z_1'}{c-c'}} \implies x = \frac{z_1 - z_1'}{c-c'} $$

Infine, la proprietà di SHVZK si verifica con l'esistenza di una trascrizione simulata con distribuzione identica a quella reale:

$$ \{(u_1, u_2;\ c;\ z) :\ u_1 \leftarrow {g_1}^k;\ u_2 \leftarrow {g_2}^k;\ z \leftarrow k + cx\} $$

$$ \{(u_1, u_2;\ c;\ z) :\ u_1 \leftarrow {g_1}^z {y_1}^{-c};\ u_2 \leftarrow {g_2}^z {y_2}^{-c}\} $$

Composizione OR

La seguente composizione permette di dimostrare la congiunzione (OR) di due affermazioni con un unico protocollo. In questo caso il Prover vuole dimostrare di

conoscere almeno una delle due testimonianze correlate con le affermazioni (non è necessario che le conosca entrambe), senza rivelare quale sia delle due.

Ciò è possibile grazie all'uso dello stesso simulatore usato per verificare la proprietà di SHVZK. Il Prover esegue il simulatore sull'istanza del protocollo di cui non conosce la testimonianza, in questo modo è in grado di ottenere una trascrizione (u_1, c_1, z_1) che ha la stessa distribuzione di probabilità di una trascrizione reale accettata da un Verifier onesto. Ottenuta la simulazione il Prover può eseguire l'algoritmo della seconda istanza normalmente generando il commitment u_2 e inviandolo al Verifier insieme al primo commitment.
A questo punto il Verifier risponde con una challenge c e il Prover la usa per ricavare la seconda challenge $c_2 = c - c_1$ che utilizza per calcolare la risposta z_2 seguendo la normale esecuzione del protocollo. È proprio grazie alla possibilità di dividere la challenge in due che il Prover può rispondere ad entrambi i protocolli anche se non conosce una delle due testimonianze. Infine, il Prover invia entrambe le risposte (z_1, z_2) e il Verifier accetta la dimostrazione se entrambe le risposte sono corrette.

È importante che il Verifier non possa distinguere quali delle due risposte è quella ricavata dalla testimonianza in possesso del Prover, e ciò è garantito dall'utilizzo del simulatore che genera una trascrizione distribuita in modo identico a quella reale.
Formalmente si traduce in [23]:

Si assume che (y_1, y_2) siano gli input in comune tra Prover e Verifier, e x sia la testimonianza segreta in possesso del Prover corrispondente a uno dei due valori (y_1, y_2). Per semplicità si ipotizza che x corrisponda a y_1:

1. Il Prover genera il valore del primo commitment u_1, sceglie un valore casuale c_2 ed esegue un simulatore sull'input (x, c_2) ottenendo come output la trascrizione (u_2, c_2, z_2).
   Il Prover invia (u_1, u_2) al Verifier.
2. Il Verifier sceglie una challenge casuale c e la invia al Prover
3. Il Prover ricava la seconda challenge c_1 in modo tale che $c_1\ XOR\ c_2 = c$, e la utilizza insieme alla testimonianza per calcolare la risposta z_1.
   Il Prover invia (u_1, c_1, z_1) e (u_2, c_2, z_2) al Verifier
4. Il Verifier controlla che entrambe le trascrizioni siano corrette

Quanto appena descritto è il protocollo generale; per applicarlo all'esempio del protocollo di Schnorr si considera nuovamente che x sia la testimonianza dell'affermazione y_1; nel caso in cui, invece, la testimonianza corrisponda all'affermazione y_2 tutte le considerazioni rimarrebbero valide in modo speculare.

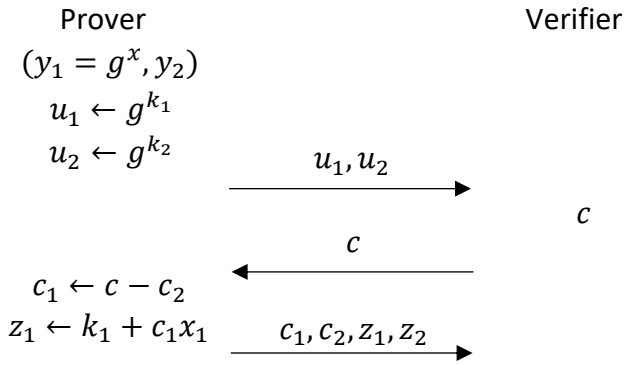

Figura 2.5: Composizione OR del protocollo di Schnorr nel caso in cui il Prover sia a conoscenza del segreto di y_1

La completezza è soddisfatta in quanto:

$$g^{z_1} = g^{k_1 + c_1 x_1} = g^{k_1}(g^{x_1})^{c_1} = u_1 {y_1}^{c_1}$$

$$g^{z_2} = u_2 {y_2}^{c_2}$$

La correttezza speciale si verifica tramite le due trascrizioni $(u_1, u_2;\ c;\ c_1, c_2, z_1, z_2)$ e $(u_1, u_2; c';\ c_1', c_2', z_1', z_2')$ con $c \neq c'$ e di conseguenza $c_1 + c_2 \neq c_1' + c_2'$, dunque può verificarsi che $c_1 \neq c_1'$, $c_2 \neq c_2'$ o entrambe le condizioni. Nel caso in cui siano presenti entrambe si hanno le equazioni:

$$g^{z_1} = u_1 {y_1}^{c_1}, \qquad g^{z_2} = u_2 {y_2}^{c_2}, \qquad g^{z_1'} = u_1 {y_1}^{c_1'}, \qquad g^{z_2'} = u_2 {y_2}^{c_2'}$$

Dalle quali si ricavano:

$$g^{z_1 - z_1'} = {y_1}^{c_1 - c_1'}, \qquad g^{z_2 - z_2'} = {y_2}^{c_2 - c_2'}$$

$$\Rightarrow x_1 = \frac{z_1 - z_1'}{c_1 - c_1'}, \qquad x_2 = \frac{z_2 - z_2'}{c_2 - c_2'}$$

Infine, la proprietà di SHVZK è dimostrata dall'esistenza delle trascrizioni indistinguibili:

$$\{(u_1, u_2; c; c_1, c_2, z_1, z_2) : u_1 \leftarrow g^{k_1}; u_2 \leftarrow g^{k_2} {y_2}^{-c_2};\ c_1 \leftarrow c - c_2; z_1 \leftarrow k_1 + c_1 x_1\}$$

$$\{(u_1, u_2; c;\ c_1, c_2, z_1, z_2) : u_1 \leftarrow g^{z_1} {y_1}^{-c_1};\ u_2 \leftarrow g^{z_2} {y_2}^{-c_2};\ c_2 \leftarrow c - c_1\}$$

Composizione NEQ

L'ultima composizione è la controparte della composizione EQ, in questo caso non solo g_1 e g_1 sono differenti, ma lo sono anche le testimonianze, dunque:

$$y_1 = {g_1}^{x_1}, \qquad y_2 = {g_2}^{x_2}$$

In questo caso il protocollo necessita di un numero maggiore di valori da scambiare tra le parti.

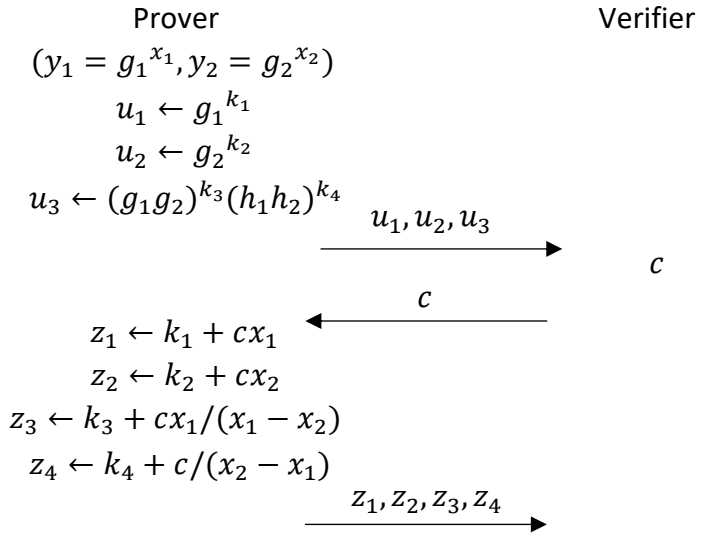

Figura 2.6: Composizione NEQ del protocollo di Schnorr

Per verificare la completezza è necessario ricavare una equazione di g_2 partendo da:

$$y_1 y_2 = {g_1}^{x_1} {g_2}^{x_2} = (g_1 g_2)^{x_1} {g_2}^{x_2 - x_1}$$

Dalla quale si ricava:

$$g_2 = (g_1 g_2)^{x_1/(x_1 - x_2)} (y_1 y_2)^{1/(x_2 - x_1)}$$

Utilizzando questa equazione è possibile dimostrare l'uguaglianza:

$$(g_1 g_2)^{z_3} (y_1 y_2)^{z_4} = u_3 {g_2}^c$$

La proprietà di correttezza speciale, in modo analogo alle precedenti dimostrazioni, si verifica utilizzando due trascrizioni con in comune i tre commitment e ricavando:

$$x_1 = \frac{z_1' - z_1}{c' - c}, \qquad x_2 = \frac{r_2' - r_2}{c' - c}$$

Infine, la proprietà di SHVZK si dimostra con l'esistenza della conversazione simulata con la stessa distribuzione di probabilità di quella reale:

$$\{(u_1, u_2, u_3; c;\ z_1, z_2, z_3, z_4) :\ u_1 \leftarrow {g_1}^{z_1} {y_1}^{-c};\ a_2 \leftarrow {g_2}^{z_2} {y_2}^{-c};$$

$$u^3 \leftarrow (g_1 g_2)^{z_3} (y_1 y_2)^{z_4} {g_2}^{-c}\}$$

Abbinando tra loro le tecniche di combinazioni appena descritte è possibile costruire dei Σ-Protocol per ogni combinazione booleana di affermazioni. Tuttavia, la lunghezza delle variabili utilizzate (commitment e risposta) aumenta di molto al crescere della complessità della formula booleana utilizzata.

2.3 Dimostrazioni a conoscenza zero non-interattive

Nelle varie applicazioni non sempre è possibile svolgere le interazioni necessarie ai protocolli appena descritti: lo scambio di messaggi tra le due parti necessita di tempo e che Prover e Verifier siano online contemporaneamente, vincoli che non sempre è possibile soddisfare. Per questi motivi, nel 1988 è stata introdotta da Blum, Feldman e Micali [13] la notazione di dimostrazione a conoscenza zero non-interattiva.
Nella loro pubblicazione gli autori mostrano come le interazioni in ogni dimostrazione a conoscenza zero possano essere sostituite da una stringa casuale, di lunghezza polinomiale e comune a dimostratore e verificatore. Nel 1991 venne

pubblicato un secondo articolo, da Blum, De Santis, Micali e Persiano [27], i quali ripresero la stessa trattazione per migliorarla.

Il modello sviluppato in questi articoli è detto a stringa di riferimento comune (*Common Reference String – CRS*), e si contrappone ad un altro modello detto a oracolo casuale (*random oracle*) utilizzato da Fiat e Shamir [16] nel 1987 per definire una tecnica in grado di convertire una dimostrazione a conoscenza zero interattiva in una non-interattiva.

2.3.1 Modello con stringa di riferimento comune

Nei loro articoli, Blum e gli altri autori [13] spiegano che le dimostrazioni a conoscenza zero sono composte da tre principali elementi:

1. Interazione: avviene uno scambio di messaggi tra Prover e Verifier
2. Randomizzazione nascosta: i valori casuali generati dal Verifier sono impredicibili per il Prover
3. Difficoltà computazionale: il Prover basa la sicurezza della sua prova sulla difficoltà computazionale di problemi matematici

Tuttavia, gli autori spiegano anche che la difficoltà computazionale da sola è in grado di rendere non necessaria l'interazione ed eliminare la segretezza della randomizzazione, tramite l'utilizzo di un'unica stringa random condivisa. In questo modo la proprietà di conoscenza zero dipende da quanto la stringa è realmente casuale, e non più dalla sua segretezza e impredicibilità.

Il modello così introdotto è composto da tre entità: il Prover, il Verifier e la stringa di riferimento che può essere letta da entrambi. Lo scambio di messaggi è sostituito da un unico messaggio inviato dal Prover al Verifier, il quale deve poi prendere una decisione: se accettare o meno la dimostrazione del Prover.
Come nel caso di dimostrazioni a conoscenza zero interattive, è bene prima definire [20] cos'è un sistema di prova non-interattivo nel modello CRS:

> Una coppia di macchine probabilistiche (P, V) è chiamata sistema di prova non-interattivo per un linguaggio L, se V è a tempo polinomiale e si verificano le seguenti condizioni:

- Completezza (*completeness*): per ogni x appartenente al linguaggio L, la probabilità che il Verifier accetti la dimostrazione del Prover, basata su x e sulla stringa casuale R, è prossima a 1.
- Correttezza (*soundness*): per ogni x non appartenente al linguaggio L e ogni algoritmo B, la probabilità che il Veirfier accetti la dimostrazione del Prover, basata su x e sulla stringa casuale R utilizzando l'algoritmo B, è prossima a zero.

R corrisponde alla stringa comune di riferimento.

Anche in questo caso, la probabilità di queste due condizioni può essere rispettivamente aumentata e diminuita con la ripetizione del processo, usando una sequenza di stringhe di riferimento scelte indipendentemente, a discapito della velocità e del carico computazionale.

Per rendere a conoscenza zero il sistema, deve essere soddisfatta anche la proprietà di conoscenza zero (la quale risulta essere più semplice rispetto alle prove interattive, in quanto il Verifier non può influenzare le azioni del Prover):

> Un sistema di prova non-interattivo (P, V) per un linguaggio L, e per ogni x appartenente a L, è a conoscenza zero se esiste una macchina probabilistica S (simulatore) tale per cui l'output di S (composto dalla stringa casuale in comune simulata e la simulazione della prova non-interattiva inviata da P) e la coppia formata dalla stringa casuale reale e la prova originale risultano computazionalmente indistinguibili [28].

In questo caso, dunque, il simulatore deve essere in grado di generare la stringa casuale di riferimento, e non solo di simulare il Prover.

Come accennato precedentemente, diventa fondamentale che il Verifier sia sicuro che la stringa sia realmente casuale, altrimenti la proprietà di correttezza verrebbe meno: se il Prover ha la possibilità di scegliere il *seed* del generatore pseudo-casuale[5], può sceglierlo in modo tale che il Verifier, durante la verifica, chieda proprio quel tipo di domande che permettano al Prover di produrre una prova convincente dell'appartenenza di x a L, anche se in realtà x non vi appartiene.

[5] Un generatore pseudo-casuale (*Pseudo-Random Number Generator* – PRNG) è un algoritmo deterministico con lo scopo di generare numeri pseudocasuali, i quali non sono realmente casuali, bensì è possibile risalire alle condizioni che li hanno generati: la sequenza generata è determinata dal valore iniziale chiamato *seed*.

La generazione della CRS è detta "*trusted setup*" e necessita di una fase iniziale di comunicazione per lo scambio di un parametro, detto *public parameter*, generato da un "*trusted party*" che ne garantisca l'affidabilità. La stringa di riferimento così generata utilizzando il parametro affidabile è valida per tutta la sessione. La trusted party, quindi, deve essere disponibile all'inizio della sessione, generando un ulteriore vincolo al modello che non sempre può essere rispettato. Per questo esistono vari studi con l'obbiettivo di trovare metodi alternativi che permettano la creazione della challenge affidabile.
Un esempio è l'utilizzo di un protocollo multi-parti per la generazione del parametro pubblico, che richiede di interagire con molti elementi, dei quali è sufficiente che anche solo uno sia onesto per garantire la proprietà di correttezza [29] [30] [31].

Se la challenge è creata da una terza entità affidabile, resa pubblica, e non vi è più una parte interattiva tra Prover e Verifier, la veridicità dell'affermazione del Prover diventa verificabile da chiunque sia interessato alla sua validità e non più solo dal Verifier coinvolto. Questo è vero anche per i protocolli non-interattivi che si basano sul modello a oracolo casuale.

2.3.2 Modello a oracolo casuale

Una delle prime applicazioni dell'oracolo casuale la si trova nel lavoro di Fiat e Shamir [16], mentre una metodologia esplicita è stata descritta da Bellare e Rogaway nella pubblicazione "*Random Oracles are Practical: a Paradigm for Designing Efficient Protocols*" [32].
Si definisce il modello sulla base della pubblicazione di Bellare e Rogaway:

> Un oracolo casuale è una funzione che produce un valore casuale in base a una stringa data in input. Se la stringa in input è la stessa il valore prodotto è lo stesso.
> Più nello specifico, un oracolo casuale R consiste in una funzione di mappatura da $\{0,1\}^*$ a $\{0,1\}^\infty$, che sceglie ogni bit di $R(x)$ uniformemente e indipendentemente, per ogni x.

Con il simbolo di infinito si intende che la stringa restituita dall'oracolo è infinita, ovviamente questo nella realtà risulta impossibile. Quindi, nella metodologia di

Bellare e Rogway, un protocollo viene prima sviluppato nel modello di oracolo casuale, dopodiché l'oracolo viene sostituito da una funzione di hash[6] adeguata.

La definizione di dimostrazione a conoscenza zero rimane molto simile alla stessa nel modello a stringa di riferimento casuale, dove la differenza sostanziale consiste nel sostituire alla CRS l'output dell'oracolo casuale. Dunque, il simulatore S dovrà essere in grado di costruire anche la simulazione dell'oracolo. Per quanto riguarda la lunghezza della stringa simulata, idealmente dovrebbe essere infinita, ma nella pratica consiste in una piccola (di grandezza polinomiale) parte dell'oracolo. Sulla base di questo modello Fiat e Shamir hanno presentato un metodo per trasformare un protocollo di identificazione interattivo in uno non-interattivo.

2.3.3 Trasformata Fiat-Shamir

La trasformata di Fiat-Shamir (*Fiat-Shamir transform – FS*) [16], detta anche euristica, consiste in una conversione di un Σ-Protocol interattivo in una dimostrazione a conoscenza zero non-interattiva, che è sicura nel modello dell'oracolo casuale.
L'idea è quella di sostituire la fase di creazione della challenge da parte del Verifier con l'interazione con un oracolo casuale; in questo modo il Prover è in grado di ottenere da solo la challenge, senza dover interagire con il Verifier, ma in un modo che non può controllare, così che non possa comportarsi in modo disonesto.

Dunque, per dimostrare un'affermazione, il Prover può richiedere la stringa casuale all'oracolo sulla base del commitment u, calcolare il valore z e inviare al Verifier i valori (u, z). Il Verifier a sua volta deve contattare l'oracolo e farsi restituire la stringa casuale corrispondente all'input u, in modo da poter così verificare la correttezza di z come farebbe normalmente. In questo modo si elimina anche la possibilità che il Verifier si comporti in modo disonesto, in quanto non è lui a generare la challenge, ma l'oracolo, il quale per definizione restituisce solo valori realmente casuali.

[6] Una funzione di hash è una funzione che mappa un valore in ingresso di qualsiasi lunghezza in un valore di uscita con lunghezza predefinita. La funzione deve essere non invertibile, ovvero non deve essere possibile dall'output risalire all'input, e deve essere computazionalmente intrattabile la ricerca di una coppia di valori in input che diano lo stesso valore in output (collisione).

In modo più formale si può affermare che:

> Dato un Σ-Protocol (P, V) e una funzione di hash h: $\{0,1\}^* \rightarrow C$, che rappresenta l'oracolo casuale, la trasformata di Fiat-Shamir sostituisce il secondo passaggio costituito dalla generazione di c da parte del Verifier, con $c = h(x, u)$, mentre il calcolo di z e la verifica del Verifier rimangono invariati.

Il protocollo così definito risulta essere uguale a un Σ-Protocol, ma senza l'interazione tra Prover e Verifier, ed è sicuro se la funzione di hash h è modellata come un oracolo casuale. Vi è però anche un'altra differenza: l'utilizzo dell'oracolo fa sì che il Prover possa richiedere la challenge più volte, nella speranza che l'oracolo restituisca una challenge casuale fortunata, alla quale il Prover sappia rispondere e possa così ingannare il Verifier. Per questo motivo risulta fondamentale la lunghezza della stringa restituita dall'oracolo, in quanto maggiore è la lunghezza e minore è la probabilità che il Prover riesca ad ottenere la challenge che cerca (maggiore il tempo necessario alla ricerca). Per questo motivo l'oracolo casuale è definito come una funzione che restituisce una stringa infinita.

Tuttavia, nella realtà la funzione usata dall'oracolo non può restituire una stringa infinita: normalmente la funzione utilizzata è una funzione di hash che restituisce un output con una lunghezza fissata t. Se t è sufficientemente grande, il protocollo mantiene la proprietà di correttezza computazionale e il livello di sicurezza che si ottiene può essere sufficiente a molte applicazioni reali. Infatti, se t è sufficientemente grande, il Prover è in grado di convincere il Verifier con una sola challenge, e la probabilità che il Prover riesca a far accettare un'affermazione falsa con una sola challenge è trascurabile.

Il calcolo dell'hash permette di aggiungere informazioni in più al protocollo, per esempio un messaggio m, trasformando la prova di conoscenza in uno schema di firma digitale[7]. Infatti, aggiungendo un messaggio m come valore opzionale della funzione di hash, si può ottenere una firma su m, la quale può essere prodotta solamente da qualcuno che conosce la chiave segreta x, e dunque è una prova dell'appartenenza del messaggio m.

[7] Uno schema di firma è un protocollo crittografico in grado di dimostrare l'autenticità di un messaggio o un documento. In particolare, permette di verificare l'origine del messaggio e la sua integrità (se è stato modificato o meno).

Di seguito è riportato l'esempio dell'applicazione della trasformata di Fiat-Shamir al protocollo di identificazione di Schnorr, convertendolo in un protocollo non-interattivo, e, in particolare, in un protocollo di firma digitale.

Si definisce una funzione di hash $H : \{0,1\}^{\lambda} \rightarrow \mathbb{Z}_q$ comune tra le parti, il segreto x di cui provare la conoscenza, e il valore comune y tra Prover e Verifier:

$$y = g^x \, mod \, p$$

Peggy genera un numero intero casuale k, e lo nasconde utilizzandolo come esponente di g. Il valore u corrisponde al commitment.

$$u = g^k \, mod \, p$$

A questo punto il dimostratore dovrebbe contattare il verificatore per ottenere la challenge da utilizzare, ma questa interazione viene sostituita con l'uso della funzione di hash definita precedentemente:

$$c = H(u \,||\, m)$$

Il valore m è un messaggio opzionale e "||" rappresenta l'operatore concatenazione. Il Prover calcola ora z e invia al Verifier la coppia (z, c) che costituisce la firma del messaggio.

$$z = -xc + k \, mod \, p$$

Il Verifier può verificare la correttezza della firma ricalcolando u e c per poi confrontare i valori ottenuti

$$u_V = y^c \cdot g^z$$

$$c_V = H(u_V \,||\, m)$$

Verificare se $c = c_V$

Infatti, se il protocollo è eseguito correttamente, si ha che:

$$u_V = g^{xc} \cdot g^{-xc+k} \, mop \, p = g^k \, mod \, p = u$$

Il protocollo così ottenuto è sicuro nel modello dell'oracolo casuale, nel quale H, agendo come oracolo casuale restituisce una challenge c per ogni messaggio m, completamente casuale, seguendo la stessa distribuzione di quando il Prover interagisce con un Verifier onesto.

In alcuni casi, tuttavia, può diventare insicuro. Infatti, come visto per il corrispondente protocollo interattivo, se lo stesso valore casuale k viene utilizzato in due firme distinte è possibile risalire alla chiave privata:

$$z_2 - z_1 = (k_2 - k_1) - s(c_2 - c_1)$$

$$con\ k_2 = k_1 \ \rightarrow \ s = \frac{z_1 - z_2}{c_2 - c_1}$$

Il valore k, quindi, deve essere diverso per ogni firma e viene chiamato *nonce*. L'unico in grado di sfruttare questa insicurezza deve essere l'estrattore, il quale avendo accesso al codice del Prover e grazie alla tecnica del riavvolgimento è in grado di ottenere due trascrizioni con lo stesso k, estrarre la testimonianza x e dimostrare così la proprietà di correttezza. Nell'esecuzione normale del protocollo bisogna prestare molta attenzione a non utilizzare mai lo stesso k, e ciò dipende dal generatore di numeri casuali utilizzato dal Prover.

La trasformata di Fiat-Shamir non è il solo modo per trasformare una dimostrazione a conoscenza zero interattiva in una non interattiva. Studi successivi si concentrarono nel trovare un'alternativa all'utilizzo dell'oracolo casuale: un esempio è lo studio di Damgård, Fazio e Nicolosi [33], il quale descrive un protocollo basato su una forma di cifratura chiamata "*homomorphic encryption*" per trasformare una classe di protocolli sigma in dimostrazioni a conoscenza zero non-interattive, sicure al di fuori del modello ad oracolo casuale.

2.4 Esempi di dimostrazioni a conoscenza zero

Al fine di una migliore comprensione dei concetti trattati finora vengono, di seguito, riportati due esempi di dimostrazioni a conoscenza zero.

2.4.1 ZKP per la colorazione di un grafo con tre colori

La dimostrazione a conoscenza zero per la colorazione di un grafo con tre colori è uno dei protocolli più importanti da un punto di vista didattico. Infatti, è tramite questo protocollo che Goldreich, Micali e Wigderson [6] dimostrarono l'esistenza delle dimostrazioni a conoscenza zero per tutti i problemi in NP, aumentando notevolmente l'interesse nei loro confronti.

I problemi NP sono definiti come i problemi decisionali la cui soluzione può essere trovata in tempo polinomiale con una macchina di Turing non deterministica. Di questi fanno parte i problemi NP-completi, come la colorazione di un grafo, i quali rappresentano i problemi NP più difficili, e possiedono un'importante proprietà:

> Per ogni linguaggio in NP esiste una riduzione polinomiale che lo riduce ad un linguaggio in NP-completo.

Questo significa che ogni problema in NP è riducibile in tempo polinomiale ad un problema in NP-completo. Per questo motivo, dimostrare l'esistenza di una dimostrazione a conoscenza zero per il problema NP-completo della colorazione di un grafo con tre colori significa anche dimostrare l'esistenza di una dimostrazione a conoscenza zero per tutti i problemi in NP, in quanto tutti sono riducibili ad esso.

Un grafo si dice essere colorabile con tre colori se è possibile colorare i suoi vertici utilizzando tre colori differenti, in modo tale che ad ogni coppia composta da vertici adiacenti siano assegnati due colori diversi tra loro. La colorazione del grafo è indicata tramite tre partizioni dei vertici del grafo, ognuna corrispondente ad un colore, mentre il linguaggio corrispondente, chiamato "*graph 3-colorability*", è indicato come G3C ed è composto da tutti i grafi che sono colorabili con tre colori.

La dimostrazione a conoscenza zero corrispondente si basa sull'assunzione di esistenza di una *one-way function* f, la quale viene utilizzata per fare il commitment dei valori durante l'esecuzione del protocollo. L'input in comune tra Prover e Verifier è il grafo, composto dai suoi vertici e archi $G(V, E)$, il quale si assume essere semplice[8] e connesso[9].

[8] Un grafo è detto semplice se non comprende cappi (ovvero un arco con i due vertici coincidenti) ed archi multipli (più di un arco che collega gli stessi due vertici).

[9] Un grafo è detto connesso se per ogni coppia di vertici esiste un cammino che li collega.

Il valore segreto in possesso del Prover è la colorazione del grafo indicata come

$$\phi: V \to \{1, 2, 3\}$$

Altri valori utili al protocollo sono:

- $n = |V|$, ovvero il numero di vertici, identificati come $V = \{1, 2, \dots, n\}$
- $m = |E|$, ovvero il numero di archi. Siccome il grafo è connesso si ha

$$n - 1 \leq m < n^2/2$$

- $S_3 = Sym(\{1, 2, 3\})$, ovvero l'insieme delle permutazioni di $\{1, 2, 3\}$

Viene riportato in seguito la descrizione di un round del protocollo, il quale è ripetibile t volte per aumentarne la sicurezza.

1. Il Prover sceglie una permutazione casuale $\pi \in S_3$, e n numeri casuali $r_i \in \{0, 1\}$. Successivamente, calcola i valori $F_i = f\big(\pi\big(\phi(v_i)\big), r_i\big)$ per ogni elemento dell'insieme dei vertici, e invia la sequenza risultante al Verifier. Questa sequenza corrisponde al commitment della colorazione del grafo.
2. Il Verifier sceglie in modo casuale un arco $e \in E$ e lo invia al Prover
3. Il Prover invia la colorazione corrispondente ai vertici dell'arco $e = (u, v)$, ovvero invia $\big(\pi\big(\phi(u)\big), r_u\big)$ e $\big(\pi\big(\phi(v)\big), r_v\big)$.
4. A questo punto il Verifier può controllare la correttezza del protocollo, e solamente se tutte le condizioni seguenti sono verificate la dimostrazione del Prover è accettata:
    - $F_u = f\big(\pi\big(\phi(u)\big), r_u\big)$
    - $F_v = f\big(\pi\big(\phi(v)\big), r_v\big)$
    - $\pi\big(\phi(u)\big) \neq \pi\big(\phi(v)\big)$
    - $\pi\big(\phi(u)\big),\ \pi\big(\phi(v)\big) \in \{1, 2, 3\}$

Le verifiche svolte dal Verifier consistono nel controllare che i colori appena ricevuti, e corrispondenti ai vertici dell'arco scelto dal Verifier, coincidano ai colori di cui è stato svolto il commitment precedentemente e che i due colori siano diversi tra loro. Queste verifiche sono anche la dimostrazione della proprietà di completezza del protocollo.

La proprietà di correttezza, invece, è verificata in quanto un Prover disonesto verrà sempre rigettato, ad ogni round, con una probabilità di $1/m$. Questo perché un Prover disonesto avrà a disposizione una colorazione del grafo con almeno una coppia di vertici (che equivale ad un arco) con lo stesso colore; perciò, il Verifier ha probabilità di $1/m$ di chiedere proprio quell'arco. Ciò significa che la probabilità che il Prover disonesto riesca a completare l'intero protocollo con t rounds è di $(1 - 1/m)^t$. In particolare, Goldreich, Micali e Wigderson suggeriscono di usare $t = m^2$, in modo da rendere tale probabilità trascurabile. Infatti, in questo caso è possibile scrivere la probabilità come:

$$\left(1 - \frac{1}{\sqrt{t}}\right)^t$$

la quale per valori di t molto grandi ($t \to \infty$) tende a zero.

L'ultima proprietà da dimostrare è la proprietà di conoscenza zero: essa risulta più complessa delle precedenti, in quanto richiede la definizione di un simulatore S del protocollo per ogni V^*.
Il simulatore ha accesso al codice del Verifier e svolge i t rounds nel seguente modo:

1. Il simulatore sceglie un arco $e' = (u, v) \in E$ e una coppia di interi (a, b) che corrispondono a due dei tre colori del grafo presi casualmente.
   Successivamente, genera n valori casuali $r_i \in \{0, 1\}$ e calcola i valori di commitment con la funzione f per tutti i vertici di V, ovvero $F_i = f(c_i, r_i)$, dove $c_u = a$, $c_v = b$ e tutti i restanti valori di c sono uguali a zero.
   Il simulatore invia i valori così ottenuti al Verifier.
2. Il Verifier risponde con un arco $e \in E$ scelto casualmente.
3. A questo punto possono verificarsi due situazioni differenti:
   - $e = e' = (u, v)$ ovvero l'arco scelto dal Verifier e quello scelto dal simulatore coincidono, dunque il simulatore può rispondere con i colori scelti precedentemente e che risulteranno corretti, ovvero invia (a, r_u) e (b, r_v) e può procedere con l'iterazione successiva.
   - $e \neq e'$ il simulatore non può rispondere al Verifier, dunque ripete l'intero round a partire dal punto 1 finché i due archi non coincidono.

Quando tutti i round sono stati completati correttamente il simulatore può produrre una trascrizione valida del protocollo e indistinguibile in tempo polinomiale da

quella reale. L'unica differenza nelle due trascrizioni è nella fase di commitment, dove nella trascrizione reale i commitment sono tutti vertici colorati, mentre nel protocollo simulato i commitment consistono principalmente in vertici associati ad un valore zero. Siccome questi commitment non vengono mai aperti, dal punto di vista esterno essi risultano identici a commitment reali.
Una dimostrazione rigorosa dell'indistinguibilità delle due trascrizioni si può trovare nella pubblicazione di Goldreich, Micali e Wigderson [6].

Il protocollo appena visto può essere usato per costruire una dimostrazione a conoscenza zero per ogni problema in NP tramite l'utilizzo di una riduzione, ma solo sotto l'assunzione di una one-way function che permetta l'uso del commitment. Studi successivi si sono impegnati a trovare un'alternativa all'uso della one-way function, per poter generare dimostrazioni a conoscenza zero più generiche per tutti i problemi in NP; un risultato positivo in tal senso è stato ottenuto da Goldwasser e i suoi colleghi in [11].

2.4.2 ZKP per l'appartenenza ad un set

Le dimostrazioni a conoscenza zero per l'appartenenza ad un set, detti *Zero-Knowledge Set Membership* (ZKSM), sono un gruppo di dimostrazioni a conoscenza zero con lo scopo di provare se un valore appartiene o meno ad un determinato insieme discreto. Per esempio, potrebbero dimostrare l'appartenenza o meno di una città all'insieme delle città europee, oppure l'appartenenza di un valore all'intervallo numerico $[18, 200]$; si noti che tale verifica corrisponde anche a dimostrare la maggiore età di una persona, problema molto frequente ed importante.
I problemi che utilizzano un intervallo numerico come insieme di appartenenza sono dei casi speciali delle prove di appartenenza ad un set e sono chiamati *range proofs*, mentre le corrispondenti dimostrazioni a conoscenza zero sono dette *Zero-Knowledge Range Proof* (ZKRP).
Entrambi questi problemi, e i rispettivi protocolli, sono molto importanti nei sistemi di moneta elettronica e nelle blockchain.

Uno dei primi protocolli ZKRP risale al 2000 ed è stato sviluppato da Boudot [34], mentre l'esempio che si è scelto di descrivere è stato sviluppato da Camenish, Chaabouni e Shelat nel 2008 [35], il quale risulta essere un'alternativa più efficiente del precedente e che può essere utilizzato sia come ZKMS che come ZKRP.

Il protocollo ZKMS di Camerish permette ad un Prover di dimostrare ad un Verifier l'appartenenza di un valore, di cui è stato eseguito un commitment, ad un insieme pubblico e discreto Φ. Il set Φ può essere considerato un input comune al dimostratore e verificatore, dunque non è necessario esplicitarlo.

L'approccio utilizzato dagli autori in [35] consiste nel codificare il set Φ in modo tale che i suoi elementi vengano firmati utilizzando una chiave del Verifier vk, in modo che l'affermazione che deve dimostrare il Prover diventi nella forma "Il Prover conosce la firma con la chiave vk per l'elemento di cui è stato svolto il commitment".
Dunque, il Verifier invia al Prover la firma di ogni elemento del set Φ, e tra questi vi è anche quella dell'elemento σ di cui è stato fatto il commitment. In questo modo il Prover può utilizzare la firma di σ ricevuta per dimostrare di conoscere il commitment.

Il protocollo ZKMS consiste in un Σ-Protocol e, come tale, è un Honest-Verfier Zero-Knowledge, ma può sempre essere trasformato in dimostrazione a conoscenza zero generale [25]. Nel caso in cui il Verifier non sia più onesto è fondamentale inserire all'interno del protocollo, ad ogni passo, le verifiche della correttezza dei parametri (omesse nel caso seguente con Verifier onesto), per esempio che un parametro inviato appartenga realmente al gruppo prestabilito.

Per poter considerare il protocollo una dimostrazione a conoscenza zero devono verificarsi alcune assunzioni:

- Assunzione computazionale: il protocollo richiede l'utilizzo dei gruppi bilineari, ovvero, sia PG un generatore di gruppi tale che, preso in input 1^k, restituisca la descrizione di gruppi moltiplicativi[10] $\mathbb{G}_1$ e $\mathbb{G}_T$ di ordine primo p con $|p| = k$. Sia $\mathbb{G}_1^* = \mathbb{G}_1 \backslash \{1\}$ e $g \in \mathbb{G}_1^*$.
  I gruppi generati sono tali che esiste una mappa bilineare $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, ciò significa che

  1) Per ogni $a, b \in \mathbb{Z}_p$ si ha che $e(g^a, g^b) = e(g, g)^{ab}$
  2) $e(g, g) \neq 1$
  3) La mappa bilineare è calcolabile in modo efficiente

[10] Un gruppo moltiplicativo è un gruppo la cui operazione binaria (operazione che si svolge completamente su elementi dello stesso insieme) è rappresentata dalla moltiplicazione.

- Assunzione forte di Diffie-Hellman (q-SDH): si dice che l'assunzione q-SDH associata al generatore PG si verifica se, per ogni macchina PPT A, la probabilità che $A\left(g, g^x, \dots, g^{x^q}\right)$, con $x \leftarrow \mathbb{Z}_p$, dia come output una coppia $\left(c, g^{1/(x+c)}\right)$ con $c \in \mathbb{Z}_p$ è trascurabile in k.

Lo schema utilizza come funzione di commitment quella utilizzata nello schema di Pederson:

> Siano g, h elementi di un gruppo G di ordine primo q; il commitment svolto dal dimostratore consiste in $C = g^s h^r$, dove s è il segreto in possesso del dimostratore e r è un elemento random appartenente a $\mathbb{Z}_q$.

Ricapitolando, gli input in comune tra le parti sono: gli elementi g, h, il commitment C e il set Φ. Invece, gli input segreti del Prover sono i valori σ, r tali che $C = g^\sigma h^r$ e $\sigma \in \Phi$. Di seguito vengono riportati i passaggi del protocollo.

1. Il Verifier seleziona un valore $x \in \mathbb{Z}_p$ corrispondente alla propria chiave privata dello schema di firma, ed invia la chiave pubblica $y = g^x$ al Prover. Inoltre, calcola ed invia al Prover la firma A_i per ogni elemento $i \in \Phi$ dove

$$A_i = g^{\frac{1}{x+i}}$$

2. Il Prover seleziona un valore $v \in \mathbb{Z}_p$ ed invia al Verifier $V = A_\sigma^v$
3. Il Prover seleziona $s, t, m \in \mathbb{Z}_p$ necessari per oscurare i tre valori segreti σ, v, r rispettivamente, e invia i due valori seguenti

$$a = e(V, g)^{-s}\, e(g, g)^t$$

$$D = g^s h^m$$

4. Il Verifier seleziona una challenge casuale $c \in \mathbb{Z}_p$ e la invia al Prover
5. Il Prover invia tre valori in risposta

$$z_\sigma = s - \sigma c$$

$$z_v = t - vc$$

$$z_r = m - rc$$

A questo punto il Verifier deve svolgere i seguenti controlli per approvare o meno la dimostrazione del Prover:

$$D = C^c h^{z_r} g^{z_\sigma}$$

$$a = e(V, y)^c \cdot e(V, g)^{-z_\sigma} \cdot e(g, g)^{z_v}$$

Lo schema di firma utilizzato nel primo passaggio per calcolare gli A_i è lo schema di firma di Boneh-Boyen [36], il quale risulta sicuro sotto l'assunzione q-SDH. Gli ultimi tre passaggi, invece, corrispondono alla prova di conoscenza vera e propria.

Per quanto riguarda le proprietà delle dimostrazioni a conoscenza zero, la proprietà di completezza si può dimostrare sostituendo i valori opportuni nelle equazioni che deve controllare il Verifier e verificando che esse risultino vere se il Prover ha seguito il protocollo.

- $D = C^c h^{z_r} g^{z_\sigma} \quad \Rightarrow \quad g^s h^m = (g^\sigma h^r)^c \cdot h^{m-rc} \cdot g^{s-\sigma c}$

$$= g^{\sigma c + s - \sigma c} \cdot h^{rc + m - rc}$$

$$= g^s h^m$$

- $a = e(V, y)^c \cdot e(V, g)^{-z_\sigma} \cdot e(g, g)^{z_v}$

$$\Rightarrow \quad e(V, g)^{-s}\, e(g, g)^t = e\left(g^{\frac{v}{x+\sigma}}, g^x\right)^c \cdot e\left(g^{\frac{v}{x+\sigma}}, g\right)^{-s+\sigma c} \cdot e(g, g)^{t-vc}$$

$$= e(g, g)^{\frac{vxc}{x+\sigma}} \cdot e(g, g)^{\frac{v(-s+\sigma c)}{x+\sigma}} \cdot e(g, g)^{t-vc}$$

$$= e(g, g)^{\frac{vxc+(-vs+v\sigma c)+(t-vc)(x+\sigma)}{x+\sigma}}$$

$$= e(g, g)^{\frac{-vs+tx+t\sigma}{x+\sigma}}$$

$$= e(g, g)^{\frac{-vs}{x+\sigma}} \cdot e(g, g)^t$$

$$= e(V, g)^{-s} \cdot e(g, g)^t$$

Invece, la proprietà di correttezza è verificata tramite l'estrazione del segreto (σ, r, v) con l'uso di un estrattore. In particolare, grazie a due trascrizioni con in comune gli elementi iniziali $(y, \{A_i\}, V, a, D, c, z_\sigma, z_v, z_r)$ e $(y, \{A_i\}, V, a, D, c', z'_\sigma, z'_v, z'_r)$, è possibile ricavare

$$\sigma = \frac{z_\sigma - z'_\sigma}{c' - c};\ \ r = \frac{z_r - z'_r}{c' - c};\ \ v = \frac{z_v - z'_v}{c' - c}$$

Infine, per verificare la proprietà di conoscenza zero è necessaria l'esistenza di un simulatore:

1. Il simulatore riceve y e $\{A_i\}$ dal Verifier, sceglie $\sigma \in \Phi$ in modo casuale (non essendo in possesso del valore segreto) e $v \in \mathbb{Z}_p$, ed esegue il protocollo normalmente inviando al Verifier $V = A_\sigma^v$.
   Il simulatore invia due valori casuali a, D.
2. Il Verifier restituisce una challenge casuale $c \in \mathbb{Z}_p$
3. Il simulatore sceglie z_σ, z_v, z_r in modo casuale e calcola i seguenti valori di conseguenza

$$a = e(V, g)^{-z_\sigma - \sigma c} \cdot e(g, g)^{z_v + vc}$$

$$D = C^c h^{z_r} g^{z_\sigma}$$

A questo punto, il simulatore ha tutti gli elementi per creare la trascrizione simulata

$$(y, \{A_i\}, V, a, D, c, z_\sigma, z_v, z_r)$$

la quale ha distribuzione identica a una normale trascrizione tra Prover e Verifier.

Si può notare che la complessità comunicativa dipende dalla cardinalità dell'insieme Φ, in quanto il primo messaggio è il più lungo ed è costituito dalla firma di tutti gli elementi di Φ. Tuttavia, il primo messaggio può essere riutilizzato anche in altre prove di appartenenza, quindi non è necessario rinviarlo ogni volta.

Per quanto riguarda la sicurezza dell'algoritmo, invece, è necessario fare una precisazione. Cheon nella pubblicazione "*Security analysis of the strong Diffie-Hellman problem*" [37] pose l'attenzione sulla forza dell'assunzione q-Strong Diffie-Hellman evidenziando come la complessità computazionale per ricavare il segreto x

sia legata al rapporto di p e q. In particolare, per mantenere vera l'assunzione, il valore q deve essere di molto minore rispetto a p: più nello specifico, con un valore tipico di $p \geq 256$ bits, il valore di q deve essere minore di 15 bits. Tuttavia, per ovviare a questo problema, in caso di necessità è possibile adattare lo schema descritto ad un diverso schema di firma.

Il protocollo così sviluppato è utilizzabile anche come ZKRP, ovvero applicandolo ad un intervallo numerico essendo esso un caso speciale di appartenenza. Ciononostante, gli autori implementano alcune modifiche per rendere il protocollo più efficiente quando utilizzato per dimostrare l'appartenenza ad un range. In particolare, la modifica consiste nello scrivere il segreto σ come un u-array, ovvero:

$$\sigma = \sum_{j}^{l} \sigma_j \cdot u^j$$

E, di conseguenza, l'affermazione da dimostrare diventa $\sigma \in [0, u^l)$, dove u, l sono due input aggiuntivi comuni tra le parti.

Questo protocollo rappresenta un ottimo esempio di ZKSM e contemporaneamente ZKRP. Inoltre, vista l'importanza che questi problemi rivestono, soprattutto se si considerano le prove di appartenenza ad un intervallo nell'ambito delle monete digitali, vi furono numerosi studi successivi atti a trovare protocolli alternativi e più efficienti. Tra questi si cita uno dei più importanti e più utilizzati: il protocollo *Bulletproof,* realizzato da Bünz e i suoi colleghi nel 2018 e descritto all'interno della pubblicazione "*Bulletproofs: Short Proofs for Confidential Transactions and More*" [38].

Capitolo 3

Dimostrazioni a conoscenza zero nei protocolli di identificazione

3.1 Protocolli di identificazione

Un protocollo di identificazione è un protocollo tra due parti dove la prima, il Prover, deve convincere la seconda, il Verifier, di essere chi dichiara. Un tipico esempio è un utente che desidera accedere al proprio account su un sito internet. Nel tempo sono stati sviluppati molti protocolli di identificazione, e l'obbiettivo che tutt'ora si cerca di raggiungere è quello di minimizzare i costi computazionali delle parti interessate ed aumentare la sicurezza degli algoritmi.

In particolare, con sicurezza di un protocollo di identificazione, si intende la sua abilità di bloccare gli attacchi atti ad impersonificare un altro utente, ovvero non deve essere possibile per un attaccante identificarsi con successo usando l'identità di un'altra entità.
Un esempio di attacco di questo tipo è l'utilizzo di un Verifier disonesto (*cheating verifier attack*) che volutamente non segue il protocollo in modo corretto, cercando così di estrarre informazioni utili dal Prover per potersi in seguito identificare ad un altro Verifier con la sua identità.

Come per i protocolli precedenti, anche quelli di identificazione devono rispettare delle proprietà; in questo caso sono necessarie la proprietà di completezza e correttezza che vengono definite nel seguente modo:

- Completezza (*completeness*): se un utente che desidera identificare sé stesso segue il protocollo correttamente allora l'identificazione ha successo e l'utente riesce a identificarsi.

- Correttezza (*soundness*): se un utente cerca di identificarsi utilizzando l'identità di qualcun altro il protocollo di identificazione non ha successo e l'utente non riesce a identificarsi.

Prima di poter usare un protocollo di identificazione è necessario che sia stato svolto un protocollo di registrazione, ovvero il dimostratore e il verificatore devono essersi scambiati delle informazioni che permettano in seguito al verificatore di riconoscere il dimostratore, per esempio una chiave simmetrica o la password del dimostratore. Il protocollo di registrazione insieme al protocollo di identificazione definiscono uno schema di identificazione. [24]

Un esempio di schema di identificazione è il modello basato su password. Esso è lo schema di identificazione più semplice, ma anche il più insicuro. In questo caso dopo il protocollo di registrazione il Prover sarà in possesso di un *user_id*, ovvero un valore univoco che identifica il Prover, e di una password.
Il protocollo di identificazione a questo punto consiste nell'invio, da parte del Prover, della password al Verifier. L'invio della password è molto rischioso in quanto, se essa viene intercettata, qualcun altro può impersonare il Prover. Per questo motivo l'invio della password in chiaro non deve mai essere svolto, ma bisogna utilizzare algoritmi che la mascherino in modo tale che se viene intercettata non sia comunque possibile ricavare il valore originale.

Gli algoritmi sviluppati di questo tipo sono numerosi, un semplice esempio è lo schema di identificazione di Lamport, il quale utilizza una funzione di hash [39]:

- Durante la fase di registrazione il Prover sceglie $x_0 \in \{0,1\}^k$, il quale corrisponde alla password, e definisce una catena di hash, ovvero una sequenza di valori x_i, con $0 \leq i \leq l$, tali che $x_{i+1} = H(x_i)$, dove $H : \{0,1\}^* \to \{0,1\}^k$ è una funzione di hash.
  In seguito, calcola il valore $x_l = H^l(x_o)$ e lo invia al Verifier, il quale memorizza il valore all'interno di una variabile v.
  Prover e Verifier mantengono un contatore i di valore iniziale $i = 0$.
- Durante la fase di identificazione il Prover incrementa il contatore i, calcola $x_{l-i} = H^{l-i}(x_o)$ e lo invia al Verifier.
  Il Verifier controlla che $H(x_{l-i}) = v$ e in caso affermativo il protocollo di identificazione ha successo, il Verifier incrementa a sua volta i e salva in v il nuovo valore ricevuto x_{l-i}.

In questo modo anche se un attaccante riesce a intercettare la comunicazione e ad ottenere il valore x_{l-i} non è in grado di ricavare la password e quindi di eseguire il protocollo di identificazione. Tuttavia, il procedimento può essere svolto solamente l volte, dopo le quali è necessario ripetere le operazioni svolte durante la fase di registrazione generando un nuovo x_0 e una nuova catena di hash.

Lo schema di identificazione basato sulla password, però, consente anche altri tipi di attacchi; infatti, l'utilizzo della password è di per sé problematico. Tipicamente la password è scelta dall'utente, il quale la sceglie per essere semplice da ricordare, ovvero non molto lunga e spesso con parole comuni. Questo permette a terze parti di sviluppare attacchi con lo scopo di indovinarla.
Per questo motivo, negli anni sono state sviluppate numerose altre tipologie di schemi di identificazione, uno dei più utilizzati è lo schema di identificazione a chiave pubblica, ovvero che utilizza la crittografia a chiave pubblica (*public-key cryptography*), detta anche crittografia asimmetrica.

In questo protocollo, durante la fase di registrazione ogni utente deve generare una coppia di chiavi dette chiave pubblica e chiave privata, di cui la prima, come indicato dal nome, deve essere resa pubblica a chiunque, mentre la seconda deve rimanere segreta. Le due chiavi devono essere legate tra loro per permettere la risoluzione di un qualche tipo di challenge, ma deve essere impossibile risalire alla chiave privata tramite la chiave pubblica.
Un tipico esempio di challenge è l'utilizzo di due differenti algoritmi in grado, rispettivamente, di firmare e verificare un valore. In questo caso l'algoritmo di firma utilizza la chiave segreta sk, mentre l'algoritmo di verifica utilizza la chiave pubblica pk.
Il protocollo di identificazione corrispondente si svolge nel seguente modo [24]:

1. Il Verifier invia una challenge $c \in \{0,1\}^k$
2. Il Prover calcola e invia al Verifier la risposta $z = sign_{sk}(c)$
3. Il Verifier controlla il valore ricevuto calcolando $verf_{pk}(c, z)$ e dichiara se l'identificazione ha avuto successo o è stata rifiutata

Anche utilizzando protocolli a chiavi asimmetriche, tuttavia, ci sono ancora dei rischi: per esempio, chiunque riesca ad intercettare la conversazione e ad ottenere il messaggio firmato dal Prover sa per certo che il Prover ha svolto l'autenticazione con il Verifier, in quanto solamente il Prover è in possesso della chiave privata per firmare la challenge. Inoltre, siccome i protocolli a chiave pubblica richiedono che il

Prover esegua un'operazione con la chiave privata su un valore inviato dal Verifier, è possibile che il Verifier usi valori non casuali per cercare di ottenere informazioni sulla chiave privata del Prover. I rischi generali dei protocolli di identificazione sono la perdita di privacy e il rischio di esposizione dell'identità del Prover.
Con l'applicazione delle dimostrazioni a conoscenza zero nei protocolli di identificazione è possibile aumentare ulteriormente la sicurezza e garantire la privacy del Prover.

3.2 Protocolli di identificazione a conoscenza zero

I protocolli di identificazione a conoscenza zero sono, come dice il nome, dei protocolli di identificazione che rispettano la proprietà di conoscenza zero, ovvero, non importa in che modo un Verifier cerchi di ingannare il Prover, comunque non sarà in grado di ottenere alcuna informazione utile che non sia già in grado di generare da solo.
Questa proprietà aggiuntiva permette di ottenere un livello di sicurezza maggiore rispetto ai normali protocolli di identificazione. In particolare, la proprietà di conoscenza zero risulta fondamentale nei casi in cui l'autenticazione avviene tramite un canale insicuro, garantendo che anche nel caso in cui una terza parte legga il contenuto dei messaggi scambiati non sia in grado di acquisire alcuna informazione utile.

L'uso delle dimostrazioni a conoscenza zero fa sì che, grazie alla trasformata di Fiat-Shamir, ogni schema di identificazione possa essere facilmente trasformato in uno schema di firma, il cui uso, però, presenta delle differenze a livello concettuale. I protocolli di identificazione si dividono in tre differenti livelli, a seconda di ciò che si vuole dimostrare e qual è il livello di sicurezza desiderato [16]:

- Schema di autenticazione (*Authentication scheme*): il Prover dimostra al Verifier di essere il Prover, e nessun'altro può dimostrare al Verifier di essere il Prover.
- Schema di identificazione (*Identification scheme*): il Prover dimostra al Verifier di essere il Prover, e il Verifier non può dimostrare ad alcuno di essere il Prover.

- Schema di firma (*Signature scheme*): il Prover dimostra al Verifier di essere il Prover, e il Verifier non può dimostrare ad alcuno (nemmeno a sé stesso) di essere il Prover.

Gli schemi di identificazione possono risultare molto simili ai protocolli di firma digitale, ma nel primo caso il Verifier è in grado di generale una trascrizione credibile di una comunicazione immaginaria, mentre, nel secondo caso, solamente un dialogo reale con il Prover può generare una trascrizione valida. Infatti, la grande differenza è che negli schemi di firma sussiste la proprietà del non ripudio (ovvero il Prover non può rinnegare di aver svolto il protocollo), mentre non è presente negli schemi di identificazione.
Normalmente, per l'identificazione di un'entità, la proprietà del non ripudio non è necessaria, bensì l'opposto: utilizzare uno schema di firma per identificare l'utente consiste in una violazione della privacy dell'utente. Di fatto, se l'utente produce una firma, essa può essere utilizzata per mostrare a chiunque che l'utente si è autenticato con un determinato Verifier violando la privacy dell'utente.

Si può notare che un protocollo di identificazione è già stato presentato, ed è il protocollo di identificazione di Schnorr, il quale può essere trasformato anche in un protocollo di firma digitale grazie alla trasformata di Fiat-Shamir.
Il protocollo di Schnorr è un protocollo di identificazione a chiave pubblica, in quanto i valori definiti inizialmente dal Prover corrispondono ad una chiave pubblica e la corrispettiva chiave privata. La chiave pubblica è $y = g^x \ mod \ p$ e viene fornita al Verifier durante la fase di registrazione, mentre x è la chiave privata.
Il fatto che il Prover voglia dimostrare al Verifier di conoscere il valore x significa che il Prover vuole dimostrare di conoscere la chiave privata associata a quella pubblica y, provando così la sua identità.

Come accennato precedentemente, i protocolli di identificazione a conoscenza zero sono spesso implementati come prove di conoscenza, in quanto l'identificazione richiede un vincolo più stringente rispetto alle normali dimostrazioni a conoscenza zero, ovvero che l'utente conosca il segreto che lo identifica. Per esempio, è richiesto che il Prover sia in grado di dimostrare di conoscere la chiave privata corrispondente a una precisa chiave pubblica. Si ricorda che, invece, nelle dimostrazioni a conoscenza zero generali è richiesto che il Prover dimostri l'appartenenza del segreto ad un linguaggio, e ciò è possibile, in alcuni casi, anche se il Prover non conosce il segreto.

Il primo protocollo di identificazione a conoscenza zero è stato definito da Fiat e Shamir [16] nel 1987.

3.2.1 Il protocollo di Fiat-Shamir

Il protocollo di Fiat-Shamir [16] si basa sul seguente teorema:

> Sia $n = p \cdot q$, con p, q due numeri primi e distinti tra loro; trovare la radice quadrata di un numero appartenente all'aritmetica modulare, ovvero risolvere $x^2 = y \ mod \ n$, è computazionalmente equivalente a trovare la fattorizzazione del numero n.

Se n è un numero grande e non primo, trovare la sua fattorizzazione è esponenzialmente difficile, dunque lo è anche trovare la radice quadrata. In particolare, trovare la radice quadrata di y è considerato un problema intrattabile.

L'inizio del protocollo si basa sul modello a chiavi asimmetriche, pertanto corrisponde alla definizione delle chiavi.
Dato $n = p \cdot q$, dove p e q sono due numeri primi distinti, il Prover genera un numero intero $1 \leq x < n$ e lo mantiene segreto insieme alla fattorizzazione di n, in modo che la tripletta (p, q, x) rappresenti la chiave privata. La chiave pubblica invece è costituita dal numero n insieme a $y = x^2 \ mod \ n$, ovvero (n, y), e viene resa pubblica in modo tale che il Verifier possa utilizzarla.

A questo punto il protocollo è costituito da una serie di messaggi scambiati tra il Prover (P) e il Verifier (V), e per aumentarne la sicurezza è possibile ripetere la procedura t volte. La procedura consiste in un Σ-Protocol, quindi i messaggi scambiati sono esattamente tre:

1. P genera un numero casuale intero $1 \leq r < n$ e il corrispondente

$$u = r^2 \ mod \ n$$

   Dopo di che invia u (il quale corrisponde al commitment) a V.

2. V genera un numero intero casuale c tra $\{0, 1\}$ che rappresenta la challenge e lo comunica a P.

3. P calcola

$$z = r \cdot x^c \; mod \; n$$

come risposta (*proof*) alla challenge e lo invia a V. Dove:

- Se $c = 0$ si ha $z = r \; mod \; n$
- Se $c = 1$ si ha $z = r \cdot x \; mod \; n$

V a questo punto può verificare che $z^2 \; mod \; n = u \cdot y^c \; mod \; n$, ovvero:

- Se $c = 0$ si ha: $r^2 \cdot x^{0 \cdot 2} = r^2 \cdot y^0$
- Se $c = 1$ si ha: $r^2 \cdot x^2 = r^2 \cdot y = r^2 \cdot x^2$

La verifica che fa il Verifier è anche la dimostrazione che il protocollo soddisfa la proprietà di completezza.
Invece, per quanto riguarda la proprietà di correttezza, si procede nel seguente modo:
Un Prover disonesto può convincere il Verifier se:

1. Sceglie r e il numero casuale $\tilde{c} \in \{0, 1\}$, calcola e invia a V:

$$u = r^2 \cdot y^{-\tilde{c}}$$

2. V genera il numero casuale $c \in \{0, 1\}$ e lo invia a P
3. P invia r al Verifier

Se $c = \tilde{c}$ il Verifier accetta la prova nonostante il Prover non conosca realmente x. Ciò succede con una probabilità di $1/2$, quindi questa è la probabilità massima con cui il Verifier può essere truffato.

Se un Prover disonesto vuole aumentare questa probabilità deve conoscere un u per il quale essere in grado di rispondere ad entrambe le challenge, ovvero:

$$z_1^2 = u \; e \; z_2^2 = u \cdot y$$

Ma questo vuol dire che sarebbe in grado di calcolare anche il segreto s:

$$z_1^2 = {z_2^2}/{y} \rightarrow y = x^2 = {z_2^2}/{z_1^2} \rightarrow x = {z_2}/{z_1}$$

Ciò va in contrasto con l'assunzione di intrattabilità su cui si basa l'algoritmo e la proprietà di correttezza è preservata [40].

L'ultima proprietà da verificare è la proprietà di conoscenza zero, la quale richiede l'esistenza di un simulatore S in grado di creare una trascrizione valida del protocollo. Ciò è possibile grazie al seguente algoritmo:

1. S seleziona $\tilde{c} \in \{0, 1\}$ e $\tilde{z}$ in modo casuale
2. S calcola $\tilde{u} = \tilde{z}^2 \cdot y^{-\tilde{c}}$ e lo invia al Verifier
3. Il Verifier invia il valore casuale $c \in \{0, 1\}$
4. Se $c = \tilde{c}$ il simulatore ha avuto successo e può restituire la trascrizione $(\tilde{u}, \tilde{c}, \tilde{z})$, altrimenti ripete il procedimento finché i due valori non coincidono

Un Verifier che vede la trascrizione del simulatore non è in grado di distinguere se sia reale o falsa, dunque, la proprietà di conoscenza zero è verificata.

Per quanto riguarda la sicurezza generale del protocollo ci sono delle considerazioni da fare. Se il protocollo è svolto solo una volta, un Prover che non possiede il segreto ha il 50% di probabilità di ingannare il Verifier. La probabilità è troppo alta per poter usare il protocollo, per questo il procedimento deve essere ripetuto t volte indipendenti tra loro, così da ridurre la probabilità a 2^{-t}. Inoltre, il protocollo appena presentato è una versione semplificata del protocollo originale.
Infatti, per ridurre ulteriormente la probabilità che un Prover disonesto riesca a convincere il Verifier, il protocollo originale pensato da Fiat e Shamir utilizza i vettori al posto delle singoli variabili:

- Il segreto x diventa il vettore $x = (x_1, \dots, x_i)$
- Il corrispettivo pubblico y diventa $y = (y_1^2, \dots, y_i^2)$

Di conseguenza l'algoritmo diventa:

1. P genera $1 \leq r < n$, calcola $u = r^2$ e lo invia a V
2. V genera un vettore di interi casuali $c = (c_1, \dots, c_i) \in \{0, 1\}^i$ e lo invia a P
3. P calcola $z = r \prod_{j=1}^{i} x_j^{c_j}$ e lo invia a V
4. V verifica che $z^2 = u \prod_{j=1}^{i} y_j^{c_j}$

In questo modo se l'algoritmo viene ripetuto per t volte la probabilità di riuscire a ingannare il Verifier diventa di 2^{-it}. Lo svantaggio che ne consegue è l'aumento del tempo necessario a svolgere l'intero protocollo.

Lo schema di identificazione di Fiat-Shamir riveste grande importanza in quanto risulta semplice e leggero dal punto di vista computazionale, tanto da poter essere utilizzato nei dispositivi a microprocessore come le Smart-Cards. Nella definizione del protocollo Fiat e Shamir ipotizzano l'esistenza di una organizzazione affidabile (*trusted*) che si occupa di controllare l'identità del Prover nella fase di registrazione, e in caso positivo firmare la Smart-Cards. È l'associazione, dunque, a scegliere il numero n e a renderlo pubblico per tutti gli utenti che desiderano registrarsi, mentre la fattorizzazione di n rimane segreta. In questo modo si ottengono anche altri vantaggi, ovvero:

- Non è necessario memorizzare i dati dell'utente, per esempio la chiave privata
- In un sistema è possibile identificare un numero molto elevato di utenti, dato dal numero di possibili chiavi pubbliche
- Le Smart-Cards possono essere usate un numero illimitato di volte, in quanto il Verifier non ha possibilità di modificare i dati al loro interno

Lo svantaggio principale del protocollo riguarda il numero di interazioni tra il Prover e il Verifier al fine di provare l'identità, il quale, in base al livello di sicurezza desiderato, possono aumentare notevolmente.

3.2.2 Il protocollo di Feige-Fiat-Shamir

Vista l'importanza del protocollo di Fiat-Shamir, pubblicazioni successive si concentrarono a migliorare il protocollo da molti punti di vista. Un primo esempio di protocollo derivante da quello di Fiat-Shamir è il protocollo di Feige-Fiat-Shamir [41], il quale si pone come obbiettivo quello di ridurre il tempo e la complessità della comunicazione.

In questa versione del protocollo i due numeri che compongono la fattorizzazione di n vengono scelti nella forma $4r + 3$ (congrui a $3\ mod\ 4$). In questo modo n è un intero Blum, la cui caratteristica principale consiste nel fatto che -1 è un non-residuo quadratico con simbolo di Jacobi $+1$ $(mod\ n)$. Il valore di n è scelto, anche in questo caso, da un'autorità fidata, la quale lo distribuisce pubblicamente in modo che chiunque possa utilizzarlo.
Le chiavi del Prover, invece, vengono scelte nel seguente modo:

1. Il Prover sceglie s numeri random $x_1, \dots, x_s$ tali che $1 \leq x_i \leq n-1$ e s bit casuali b_i
2. Il Prover calcola $y_i = (-1)^{b_i}/{x_i}^2 \ mod\ n$ con $1 \leq i \leq s$
3. Il Prover pubblica $y = y_i, \dots, y_s$ come chiave pubblica, e mantiene segreto $x = x_1, \dots, x_s$ come chiave privata

I segreti x_i sono protetti dalla difficoltà di estrarre la radice quadrata $mod\ n$, come nel protocollo di Fiat-Shamir.

Il protocollo di identificazione è composto dai seguenti passaggi, i quali possono essere ripetuti t volte per ottenere la sicurezza desiderata:

1. Il Prover seleziona un valore casuale $1 \leq r \leq n-1$ e invia al Verifier

$$u = \pm r^2 \ mod\ n$$

2. Il Verifier genera ed invia al Prover un vettore di valori booleani casuali

$$c = (c_1, \dots, c_s)$$

3. Il Prover calcola e invia la risposta

$$z = r \cdot \prod_{c_i=1} x_i \ mod\ n$$

4. Il Verifier controlla la corretta esecuzione del protocollo verificando che

$$u = \pm z^2 \cdot \prod_{c_i=1} y_i \ mod\ n$$

La verifica svolta dal Verifier è corretta in quanto, per ogni $1 \leq i \leq s$:

$$y_i {x_i}^2 = \pm 1 \ mod\ n$$

Dunque, la dimostrazione dell'equazione è la seguente:

$$z^2 \cdot \prod_{c_i=1} y_i = \left(r \cdot \prod_{c_i=1} x_i\right)^2 \cdot \prod_{c_i=1} y_i = r^2 \prod_{c_i=1} ({x_i}^2 y_i) = \pm r^2 = \pm u$$

Nel protocollo di Fiat-Shamir, suo predecessore, veniva divulgato un bit di informazione legato al fatto che il Prover dimostrava l'appartenenza del segreto al linguaggio. In questo protocollo, invece, gli autori riescono ad impedire la divulgazione di quel bit grazie all'uso delle prove di conoscenza. Infatti, è in questa pubblicazione [41] che viene evidenziato come l'aggiunta della proprietà di conoscenza alle normali dimostrazioni a conoscenza zero permette di rafforzare la loro sicurezza, soprattutto quando vengono utilizzati come protocolli di identificazione.

3.2.3 Il protocollo di Guillou-Quisquater

Un secondo esempio di miglioramento del protocollo di Fiat-Shamir è quello sviluppato da Guillou-Quisquater [17], il quale richiede la memorizzazione di un solo numero di autenticazione in ogni microprocessore, a discapito di un piccolo aumento del numero di computazioni necessarie.

Il protocollo si basa sull'algoritmo RSA, dove il problema principale consiste nel calcolare $x = y^{1/e}\ mod\ n$ dato y, il quale è un problema computazionalmente infattibile per valori p e q fattorizzazioni di n abbastanza grandi. In questo caso x corrisponde alla chiave privata, mentre $y = x^e\ mod\ n$ corrisponde alla chiave pubblica.

Dato $n = p \cdot q$ chiamato modulo, dove p e q sono due numeri primi distinti tra loro e di lunghezza k sufficientemente grande. Sia e un numero positivo intero e sufficientemente grande, tale che $MCD(e, \phi(n)) = 1$, dove $\phi(n) = (p-1)(q-1)$. Il protocollo è il seguente:

1. Il Prover seleziona un numero casuale $1 < r < n - 1$ ed invia al Verifier il commitment

$$u = r^e\ mod\ n$$

2. Il Verifier genera una challenge $0 \leq c \leq e - 1$ e la invia al Prover
3. Il Prover calcola la risposta z e la invia al Verifier, dove

$$z = r \cdot x^c\ mod\ n$$

Il Verifier per verificare la corretta esecuzione del protocollo deve controllare che

$$z^e \; mod \; n = u \cdot y^c \; mod \; n$$

L'equazione è vera in quanto, tralasciando il modulo n:

$$z^e = (r \cdot x^c)^e = r^e \cdot (x^e)^c = u \cdot y^c$$

La proprietà di completezza è dimostrata allo stesso modo, invece la proprietà di correttezza e quella di conoscenza zero sono verificate in quanto la probabilità di avere successo di un Prover disonesto è limitata da $1/e$, e non è possibile distinguere una trascrizione reale da una simulata. Segue la dimostrazione [24].

Utilizzando un simulatore al posto del Prover è possibile rispondere a due challenge distinte dopo aver inviato lo stesso commitment u, ottenendo così due trascrizioni valide (u, c, z) e (u', c', z') con $c \neq c'$.
Dalle due trascrizioni è possibile ricavare le seguenti uguaglianze:

$$z^e = u \cdot y^c, \;\; z'^e = u \cdot y^{c'}$$

Le quali implicano:

$$({}^{z}/_{z'})^e = y^{c-c'}$$

Si può notare che $c - c'$ è un numero positivo intero minore di e dunque, visto che e è un numero primo, $MCD(e, c - c') = 1$. Grazie all'algoritmo esteso di Euclide[11] è possibile calcolare due numeri interi a, b tali che soddisfano l'identità di Bézout[12]:

$$ae + b(c - c') = MCD(e, c - c') = 1$$

[11] L'algoritmo esteso di Euclide permette di calcolare il massimo comun divisore tra due interi e, contemporaneamente calcolare i coefficienti a e b dell'identità di Bézout su x e y.

[12] L'identità di Bézout afferma che se x e y sono interi e il loro massimo comun divisore è d, allora esistono due interi a e b tali che: $ax + by = d$

Ottenuti i due valori è possibile elevare a b l'equazione precedente e ricavare la chiave privata con i seguenti passaggi:

$$\left({}^{z}/_{z'}\right)^{be} = y^{b(c-c')} = y^{1-ae}$$

$$y = \left(y^{a}\left({}^{z}/_{z'}\right)^{b}\right)^{e}$$

$$x = y^{a}\left({}^{z}/_{z'}\right)^{b}$$

Ad ogni esecuzione del protocollo un Prover disonesto ha una possibilità su e di ingannare il Verifier.
Mentre, dal punto di vista del Verifier, dopo l'esecuzione del protocollo non è possibile imparare nulla riguardo alla chiave segreta x perché non è possibile distinguere una conversazione reale da quella simulata usando la chiave pubblica:

$$\{(u, c, z) : u \leftarrow r^{e}; z \leftarrow rx^{c}\},$$

$$\{(u, c, z): u \leftarrow r^{e}y^{-c}\}$$

Siccome la probabilità di ingannare il Verifier è legata a e è possibile diminuirla semplicemente aumentando e e non è necessario, a differenza del protocollo di Fiat-Shamir, ripetere il protocollo più volte, riducendo drasticamente il numero di interazioni tra il Prover e il Verifier.

3.3 Protocolli di identificazione a conoscenza zero basati su problemi NP-difficili

I protocolli di identificazione a conoscenza zero visti fino ad ora sono tutti basati sulla difficoltà computazionale della fattorizzazione in numeri primi e sul problema del logaritmo discreto (protocollo di Schnorr), ovvero problemi legati alla teoria dei numeri. Questi problemi sono considerati sicuri in quanto la loro computazione richiede molto tempo, se vengono usati valori abbastanza grandi. Tuttavia, la costante evoluzione della tecnologia ha portato a domandarsi quanto essi siano realmente sicuri, soprattutto in vista della creazione di computer quantistici.

I computer quantistici sono calcolatori che si basano sui fenomeni della fisica quantistica e che permettono di raggiungere una velocità computazionale di molto superiore rispetto a quella dei normali computer. Essi sono oggetto di studio da molti anni e nell'ultimo periodo si è riusciti a crearli, benché siano ancora con potenzialità ridotte. Una pubblicazione di Shor del 1994 [42] ha posto in evidenza come, davanti all'uso di un computer quantistico, i problemi di logaritmo discreto e fattorizzazione non sono più sicuri. Shor, infatti, sviluppa un algoritmo da utilizzare con un computer quantistico in grado di risolvere tali problemi in tempo polilogaritmico rispetto all'input iniziale.

Per questo motivo, studi successivi si concentrarono sulla definizione di algoritmi basati su problemi alternativi a quelli della teoria dei numeri. In particolare, per resistere all'algoritmo di Shor, i nuovi protocolli utilizzarono i problemi della classe NP-difficili[13] e NP-completi, i quali, allo stato attuale, risultano sicuri contro l'utilizzo dei computer quantistici.
Di seguito vengono riportati tre esempi di protocolli di identificazione a conoscenza zero basati su problemi appartenenti a queste classi.

3.3.1 Il protocollo di identificazione basato su MinRank

Uno dei primi protocolli resistenti ai computer quantistici fu proposto da Stern [43] e si basa sul problema del "*syndrome decoding*" (SD); esso è una dimostrazione a conoscenza zero che utilizza lo schema a tre mosse dei protocolli sigma e possiede un errore di correttezza di 2/3. Il problema SD rappresenta la formulazione più semplice di un problema di decoding, un problema NP-difficile che consiste nel trovare un vettore di pesi piccolo in un sottospazio affine dello spazio lineare.

Nel 2001 Nicolas T. Courtois [44] presentò un protocollo di identificazione a conoscenza zero basato su un problema molto simile, il problema dell'algebra lineare MinRank, a sua volta un problema NP-difficile. Il problema MinRank consiste nel trovare una combinazione lineare (o affine) di alcune matrici date. La definizione formale del problema MinRank è la seguente:

[13] I problemi NP-difficili sono la classe dei problemi considerati difficili almeno quanto i problemi NP-completi.

Siano $M_0;\ M_1, \dots, M_m$ delle matrici $\eta\ \times n$ su un anello R. Il problema MinRank(η, *n, m, r, R*) corrisponde a trovare una soluzione $\alpha\ \in\ R^m$ tale che:

$$Rank\left(\sum_i \alpha_i\, M_i\ -\ M_0\right) \leq r$$

Ovvero, trovare un α tale che il rango[14] della matrice tra parentesi sia minore di un valore prestabilito r. Con anello R, in generale, si indica un insieme su cui sono definite la somma e il prodotto, nel caso specifico del protocollo, R corrisponde ad un campo finito[15] $GF(q)$.

Il seguente protocollo risulta interessante in quanto uno dei primi ad utilizzare questo problema, il quale risulta essere una generalizzazione del problema SD.
Il protocollo utilizza come base lo schema a chiavi asimmetriche; dunque, necessita di una chiave pubblica e una privata [44].

- La chiave pubblica è composta da $1 + m$ matrici $\eta\ \times n$ su un campo finito $GF(q)$, $M_0;\ M_1, \dots, M_m$.
  Sia $r < n$, tramite un generatore pseudo-casuale con *seed* di 160 bits, vengono generate $1 + m - 1$ matrici $M_0;\ M_1, \dots, M_{m-1}$ e una matrice casuale M di rango r. Tale matrice è utilizzata insieme ad un $\alpha\ \in GF(q)^m$, casuale e diverso da zero, per costruire la matrice mancante M_m:

$$M_m = \left(M +\ M_0 - \sum \alpha_i\, M_i\right) / \alpha_m$$

- La chiave privata associata è la soluzione $\alpha\ \in GF(q)^m$ tale che:

$$Rank\left(\sum \alpha_i \cdot\ M_i -\ M_0\right) =\ r$$

[14] Il rango di una matrice è l'ordine massimo dei minori non nulli che si possono estrarre dalla matrice, ovvero il numero di righe (o colonne) della sottomatrice quadrata più grande che si può estrarre dalla matrice iniziale con determinante diverso da zero. Dunque, il rango sarà sempre compreso tra zero e il minimo tra il numero di colonne e quello delle righe della matrice.

[15] Un campo finito *GF(q)* è un insieme non vuoto composto da un numero finito di elementi *q,* dove *q* è il risultato di un numero primo elevato a potenza.

Il numero m deve essere scelto in modo tale che ci sia una bassa probabilità che il problema così definito abbia più di una soluzione. Per farlo si sceglie m minore o uguale ad un valore massimo definito come:

$$m_{max} = \eta n + r^2 - (\eta + n)r + 1$$

Definite le chiavi è possibile procedere con il protocollo. Lo scopo del Prover è convincere il Verifier di conoscere α, per farlo è necessario avere a disposizione una one-way function di hash H, la quale non deve avere collisioni (*collision-intractable*) e si suppone si comporti come un oracolo casuale.

La parte iniziale del protocollo è costituita dalla definizione di alcune variabili, in seguito vi è l'algoritmo vero e proprio che può essere ripetuto più volte, come nei protocolli precedenti.
Nella prima fase il Prover sceglie due matrici invertibili casuali U, T rispettivamente di dimensioni $\eta \times \eta$ e $n \times n$, e una matrice casuale di dimensione $\eta \times n$ chiamata X. Le tripletta composta dalle seguenti matrici è definita $\overline{UTX}$.
In seguito, il Prover genera una combinazione casuale β^{-1} di M_i e calcola:

$$N_1 = \sum \beta_{1i} \cdot M_i$$

Poi ricava $N_2 = M + M_0 + N_1$, dove N_2 risulta così essere:

$$N_2 = \sum \beta_{2i} \cdot M_i$$

E $\beta_2 - \beta_1 = \alpha$; in questo modo ogni β_i, preso separatamente, è casuale e uniformemente distribuito, e ogni N_i è solamente una combinazione casuale di M_i.

A questo punto l'algoritmo procede con la sequenza di operazioni che definiscono un round del protocollo di identificazione:

1. Il Prover invia al Verifier:

$$H(\overline{UTX}), \qquad H(TN_1U + X), \qquad H(TN_2U + X - TM_0U)$$

2. Il Verifier sceglie una domanda $Q \in \{0, 1, 2\}$ e la invia al Prover

3. Se $Q = 0$ il Prover invia i seguenti valori:

$$(TN_1U + X), \qquad (TN_2U + X - TM_0U)$$

Se $Q = 1, 2$ il Prover rivela i valori:

$$\overline{UTX}, \qquad \beta_Q$$

Per verificare i valori e decidere se accettare, il Verifier esegue i seguenti controlli:

- Se $Q = 0$ i valori $H(TN_1U + X)$ e $H(TN_2U + X - TM_0U)$ devono essere corretti, e la seguente matrice deve essere di rango r:

$$(TN_2U + X - TM_0U) - (TN_1U + X) = TMU$$

- Se $Q = 1, 2$ le matrici U e T devono essere invertibili, $H(\overline{UTX})$ deve essere corretto, a seconda del valore di Q, uno tra $H(TN_1U + X)$ e $H(TN_2U + X - TM_0U)$ deve essere corretto, e infine:

$$TN_QU = \sum \beta_{Qi}\ TM_iU$$

Le verifiche eseguite dal Verifier sono anche la dimostrazione che se il Prover conosce α il protocollo ha successo, e quindi il protocollo possiede la proprietà di completezza.

La successiva proprietà che deve possedere il protocollo è la proprietà di correttezza; essa si verifica dimostrando che un Prover disonesto, non in possesso di α, riesce a convincere il Verifier con una probabilità di base di 2/3. Inizialmente, si assume che il Prover sia in grado di rispondere a qualunque domanda Q.

Il Prover disonesto invia i valori $TN_1U + X$ e $TN_2U + X$ con la funzione H. Nel caso di $Q = 0$ il Verifier verifica il rango della matrice

$$\left(T\left(\sum \beta_{2i}\, M_i\right)U - TM_0U + X\right) - \left(T\left(\sum \beta_{1i}\, M_i\right)U + X\right) =$$
$$= \sum_{i=1}^{m} (\beta_{2i} - \beta_{1i}) \cdot TM_iU\ - TM_oU$$

Nel caso di $Q = 1, 2$ il Verifier verifica che i valori inviati siano nella forma $X + T\left(\sum \beta_{1,2i} M_i\right)U$ e che U e T siano invertibili, così la matrice seguente è di rango r:

$$\sum_{i=1}^{m} (\beta_{2i} - \beta_{1i}) \cdot M_i - M_o$$

Ma, in questo modo, il Prover conosce una soluzione al problema MinRank $\alpha = \beta_2 - \beta_1$, ovvero il segreto α.

Dunque, si può vedere che la probabilità che un Prover disonesto riesca ad ingannare il Verifier è $2/3$. Se l'algoritmo centrale viene ripetuto per più rounds la probabilità diminuisce ed è:

$$Pr_{disonesto} = \left({}^{2}/_{3}\right)^{\#rounds}$$

L'ultima proprietà da verificare è la proprietà di conoscenza zero; per farlo è sufficiente dimostrare l'esistenza di un simulatore S con accesso ad oracolo sul Verifier V:

1. $S(V)$ sceglie una domanda casuale $Q = 1, 2$ e si prepara a rispondere alle domande 0 e Q.
2. $S(V)$ sceglie $N = \sum \delta_i M_i$ con un δ casuale
3. Prende $\overline{UTX} = (U, T, X)$ con U e T invertibili
4. E prende una matrice casuale R di rango r
5. $N_Q = N$ e $N_{3-Q} = N + (-1)^{Q+1}(R + M_o)$. Ora $N_2 - N_1 = R + M_0$
6. $S(V)$ chiede la query del Verifier sul commitment:

$$Q' = V\left(H(\overline{UTX}), H(TN_1U + X), H(TN_2U - TM_oU + X)\right) \in \{0, 1, 2\}$$

7. Il simulatore a questo punto ripete i seguenti passi finché il Verifier non risponde con una delle due domande al quale si è preparato $Q' = \{0, Q\}$
8. Se $Q' = 0$ il simulatore $S(V)$ rivela $(TN_2U + X - TM_0U)$ e $(TN_1U + X)$ con una differenza di TRU di rango r
   Se $Q' = Q$ il simulatore $S(V)$ rivela $\overline{UTX}$ e δ che sono stati usati per costruire il valore $TN_QU + X\ [-TM_0U]$

La seguente trascrizione risulta indistinguibile da quella autentica, quindi anche la proprietà di conoscenza zero è verificata.

Il protocollo di Courtois così definito, tuttavia, si è visto avere una probabilità di successo per un Prover disonesto di $2/3$, la quale comporta un numero abbastanza alto di ripetizioni necessarie per ottenere un livello di sicurezza accettabile. Nello specifico, Courtois suggerisce un numero di rounds superiore a 35.

Per questo motivo, studi successivi si sono concentrati nel diminuire la probabilità di successo del Prover disonesto. Nella stessa pubblicazione con cui è stato presentato il protocollo, Courtois presenta una variante dell'algoritmo che diminuisce tale probabilità a $1/2$, tuttavia essa presenta molte assunzioni che ne limitano l'utilizzo e non vi è un'accurata analisi della sua sicurezza.

Infine, è bene sottolineare che nel tempo si sono susseguiti anche studi volti a trovare algoritmi risolutivi al problema MinRank più efficienti di quelli già conosciuti ed esposti in [44]. Per esempio nella pubblicazione "*Cryptanalysis of MinRank*" di Faugère, Levy-dit-Vehel e Perret [45], gli autori riescono a risolvere il protocollo di Courtois con due set di parametri da lui proposti ($m = 10,\ n = \eta = 6,\ r = 3$) e ($m = 10,\ n = \eta = 7,\ r = 4$), i quali, dunque, non sono più sicuri, mentre per gli altri parametri la sicurezza diminuisce (nel caso di $m = 11,\ n = \eta = 9,\ r = 8$ la complessità passa da 2^{138} a 2^{88}), ma è ancora sufficiente per essere utilizzati. Ciò significa che la sicurezza del protocollo va rivalutata periodicamente e i valori dei parametri cambiati se diventano insicuri.

3.3.2 Il protocollo di identificazione basato su PKP

Nel 1990 Shamir [46] sviluppò un protocollo di identificazione a conoscenza zero basato sul problema algebrico del nucleo permutato (*Permuted Kernel*). Il punto forte di questo schema riguarda il ridotto numero di bit coinvolti in tutte le fasi: ridotto numero di bit scambiati durante la comunicazione, operazioni aritmetiche a 8-bit, e ridotta grandezza delle chiavi, pubblica e privata. Anche in questo caso il protocollo, grazie alle sue caratteristiche di leggerezza, è implementabile utilizzando le smart cards.
Il problema matematico su cui si basa il protocollo è il seguente:

Dati:

- Una matrice A con dimensione $m \times n$
- Un vettore V con dimensione n
- Un numero primo p

Trovare una permutazione π tale che $V_\pi \in K(A)$

Con permutazione si intende lo scambio dell'ordine degli elementi del vettore V. Il risultato di tale permutazione è indicato come V_π ed è definito come il vettore W tale che:

$$w_j = v_{\pi(j)} \ \ per \ \ 1 \leq j \leq n$$

Mentre, $K(A)$ identifica il nucleo, o *kernel*, della matrice rettangolare A, ovvero il set degli n-vettori W tali che $AW = 0 \ mod \ p$ (ciò significa anche che $AV_\pi = 0$).
Un'ulteriore nozione che sarà utile in seguito riguarda la composizione di permutazioni: le permutazioni sono composte come le funzioni, ovvero, il vettore $V_{\pi\sigma}$ è definito come il vettore W tale che:

$$w_j = v_{\pi(\sigma(j))} \ \ per \ \ 1 \leq j \leq n$$

Trovare la permutazione π è un problema NP-completo, questo lo rende più sicuro rispetto agli algoritmi presentati precedentemente che utilizzavano la fattorizzazione come base della sicurezza del protocollo.

La fase iniziale del protocollo è composta dalla definizione delle chiavi e dei valori necessari alla dimostrazione successiva, in particolare:

- Vengono scelti come valori universali e comuni una matrice A e un numero primo p
- Viene definita una funzione di hash H che permette al Prover di svolgere il commitment dei valori senza rivelarli prematuramente al Verifier
- L'utente che vuole autenticarsi sceglie:
  - Una permutazione casuale π che corrisponde alla chiave privata
  - Un vettore casuale V tale che $V_\pi \in K(A)$, il quale corrisponde alla chiave pubblica dell'utente

Lo scopo del Prover è quello di dimostrare tramite il protocollo la conoscenza della permutazione segreta π, la quale corrisponde alla prova della sua identità.
Il protocollo viene svolto come segue:

1. Il Prover sceglie un vettore casuale R e una permutazione casuale σ, dopodiché svolge il commitment dei valori tramite la funzione di hash, inviando al Verifier:

$$H(\sigma, AR), H(\pi\sigma, R_\sigma)$$

2. Il Verifier sceglie un valore casuale $0 \leq c \leq p$ e lo invia al Prover
3. Il Prover risponde con il valore

$$W = R_\sigma + cV_{\pi\sigma}$$

4. Dopo aver ricevuto il valore W, il Verifier chiede al Prover di rivelare uno tra σ oppure $\pi\sigma$.
   A seconda del valore che è stato rivelato, il Verifier può verificare il protocollo nei seguenti modi:
   - Nel primo caso verifica la correttezza di $H(\sigma, A_\sigma W)$
   - Nel secondo caso verifica la correttezza di $H(\pi\sigma, W - cV_{\pi\sigma})$

Le verifiche svolte dal Verifier permettono ad un Prover onesto che conosce π di essere sempre accettato (completezza), infatti:

$$A_\sigma W = A_\sigma(R_\sigma + cV_{\pi\sigma}) = AR + cAV_\pi = AR$$

$$W - cV_{\pi\sigma} = R_\sigma$$

Invece, un Prover disonesto è in grado di farsi accettare dal Verifier solamente con una probabilità trascurabile (correttezza). La dimostrazione di tale affermazione è la seguente.

Quando il Prover disonesto deve scegliere i valori iniziali nel passo 1 deve prepararsi a rispondere a $2p$ possibili domande del Verifier. Se è in grado di rispondere correttamente a $p + 2$ domande, allora per lo stesso commitment (σ, X) e (τ, Y), ci

sono almeno due valori distinti c', c'' con le rispettive risposte W', W'' che soddisfano entrambe le condizioni. Quindi si ha il seguente sistema di equazioni:

$$A_\sigma W' = X$$

$$A_\sigma W'' = X$$

$$W' - c'\, V_\tau = Y$$

$$W'' - c''\, V_\tau = Y$$

Questo implica che:

$$(W' - W'') \in K(A_\sigma)$$

$$(W' - W'') = (c' - c'')V_\tau$$

Siccome:

$$c' - c'' \neq 0$$

$$V_{\tau\sigma^{-1}} \in K(A)$$

Significa che la permutazione segreta $\pi = \pi\sigma^{-1}$ può essere estratta ogni $p + 2$ risposte corrette. Di conseguenza, la probabilità di successo per un Prover disonesto che non conosce π è di $(p + 1)/2p$ che è approssimabile ad $1/2$. Ciò significa che con 20 iterazioni è possibile ridurre la probabilità a 10^{-6}, ovvero abbastanza piccola per poter usare l'algoritmo in molte situazioni pratiche.

Nel documento di presentazione dell'algoritmo, Shamir fornisce anche dei possibili valori implementativi per il protocollo. In particolare, propone per n i valori 32 e 64 che permettono di ottenere $32! = 2^{120}$ e $64! = 2^{296}$ permutazioni disponibili, in questo modo un attacco al protocollo, nel migliore dei casi, richiederebbe in media 2^{76} e 2^{184} passaggi. Mentre come valore ideale per p viene presentato 251, e il valore di m viene calcolato in modo tale che $p^m \approx n!$.

Tuttavia, nelle successive pubblicazioni è stato dimostrato che esistono modi più efficienti di ricavare la permutazione π. In particolare, nella pubblicazione del 1995 scritta da Baritaud, Campana, Chauvaud e Gilbert [47] si dimostra che con $n =$

$32,\ m = 16,\ p = 251$ si è in grado di ottenere la permutazione dopo un tempo di 2^{56}, un valore che bisogna considerare attentamente prima di implementare il protocollo. Infatti, gli autori sconsigliano l'uso di questi parametri in applicazioni che richiedono un alto livello di sicurezza. Anche i parametri $n = 64, m = 37, p = 251$ portano ad un livello di sicurezza inferiore a quello stimato da Shamir, ovvero 2^{137}, tuttavia questo valore è ancora abbastanza alto da poter essere considerato in molte applicazioni reali.

Viste le potenzialità di questo algoritmo, anche a distanza di anni dalla sua pubblicazione vi sono ancora studi in merito e, nel 2019, è stato utilizzato come base per ricavare uno schema di firma tramite l'applicazione della trasformata di Fiat-Shamir nella pubblicazione "*PKP-based signature scheme*" [48].

3.3.3 Il protocollo basato sui reticoli (lattice-based)

Un reticolo (*lattice*) è una notazione matematica di recente interesse nel campo della crittografia, e in particolare nelle dimostrazioni a conoscenza zero, che permette di sviluppare algoritmi resistenti all'uso dei computer quantistici. Vista la loro importanza in letteratura vi sono numerosi studi a riguardo: una buona fonte di informazione a proposito di questi studi è il sito web di Daniele Micciancio [49].

Come esempio esplicativo delle potenzialità dell'uso dei reticoli si è scelto di esporre il protocollo di identificazione a conoscenza zero di Cayrel, Lindner, Rückert e Silva del 2010 [50]. Questo protocollo utilizza chiavi pubbliche di dimensione ridotta e risulta essere più efficiente dei suoi predecessori.
Prima di esporre il protocollo, tuttavia, si forniscono le nozioni matematiche necessarie alla sua comprensione.

Un reticolo (anche detto sottogruppo discreto di $\mathbb{R}^m$) è una griglia infinita di punti generata da n vettori linearmente indipendenti nello spazio $\mathbb{R}^m$. Gli $n \leq m$ vettori costituiscono una base B e il reticolo corrisponde a tutte le combinazioni dei vettori in B con coefficienti appartenenti a $\mathbb{Z}$, ovvero $L = B\mathbb{Z}^n$.
I protocolli crittografici possono basarsi su diversi problemi che fanno uso dei reticoli, tra i più importanti vi sono: *Shortest Vector Problem* (SVP), *Shortest Integer Solution* (SIS), *Closest Vector Problem* (CVP) e *Learning With Errors* (LWE).
Il protocollo in questione utilizza i primi due problemi, le cui definizioni sono le seguenti.

> Data una base di un reticolo $B \in Z^{m \times n}$, lo *Shortest Vector Problem* consiste nel trovare un vettore del reticolo diverso da zero Bx tale che $\|Bx\| \leq \|By\|$ per ogni altro $y \in \mathbb{Z}^n \backslash \{0\}$.

La notazione Bx indica un elemento del reticolo generato moltiplicando la base B per un vettore di coefficienti $x \in \mathbb{Z}^m$. Mentre con $\|Bx\|$ si intende la lunghezza, o norma, del vettore, ovvero dato il vettore $v = (v_1, v_2, \dots, v_n)$

$$\|v\| = \sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}$$

Dunque, lo SVP consiste nel trovare il vettore appartenente al reticolo di lunghezza minima. Il secondo problema è il seguente:

> Data una matrice $A \in {\mathbb{Z}_q}^{n \times m}$, il problema *Short Integer Solution* consiste nel trovare un vettore non nullo $x \in \mathbb{Z}^m$ tale che $Ax = 0 \; mod \; q$ e che la sua lunghezza sia $\|x\| \leq b$, con $b < q$.

Il protocollo di identificazione seguente prende spunto da due pubblicazioni sviluppate precedentemente, delle quali cerca di sfruttare i punti di forza. In particolare, utilizza lo stesso schema del protocollo sviluppato da Cayrel e Veron [51], il quale si basa sul problema di *syndrome decoding* (introdotto per la prima volta da Stern [43]), ma lo trasforma in un protocollo basato sul problema SIS.
In questo modo, gli autori illustrano come passando da un dominio all'altro sia possibile aumentare la sicurezza dell'algoritmo e renderlo più efficiente. Questo passaggio di dominio è un concetto sviluppato precedentemente da Kawachi e altri all'interno della seconda pubblicazione di riferimento [52], nella quale si attua il passaggio di dominio con il protocollo di Stern, trasformandolo in un protocollo basato sui reticoli. Cayrel utilizza la stessa versione del problema SIS di Kawachi per migliorare ulteriormente il proprio algoritmo.

Lo schema così ottenuto risulta essere più efficiente in termini di numero di rounds necessari rispetto a quello sviluppato da Kawachi, e resistente contro gli attacchi attivi atti a estrapolare informazioni al Prover per poterlo impersonare. Infatti, grazie alla proprietà di conoscenza zero, un attaccante non riesce ad ottenere alcuna informazione utile.

La sicurezza del seguente protocollo di identificazione si basa sull'esistenza di una funzione adatta a svolgere il commitment; in particolare, tale funzione è definita

come un algoritmo deterministico che, preso come secondo input un *nonce* scelto in modo casuale, garantisce la segretezza del commitment.
La prima parte del protocollo è costituita dalla generazione delle chiavi.

Il Prover sceglie in modo casuale il vettore $x \in \{0,1\}^m$ con peso di Hamming[16] uguale a $m/2$, il quale costituisce la chiave privata, e una matrice $A \in {\mathbb{Z}_q}^{n \times m}$. La chiave pubblica corrisponde è il vettore y calcolato come

$$y = Ax \; mod \; q$$

Dopo aver definito una funzione f adeguata ad eseguire il commitment, un round del protocollo si svolge come segue.

1. Il Prover genera quattro valori in modo casuale, ovvero

$$k \in {\mathbb{Z}_q}^m$$

$$\sigma \in S_m$$

$$r_o, r_1 \in \{0,1\}^n$$

S_m indica il set delle permutazioni degli m elementi, ogni permutazione $\sigma \in S_m$ è un'operazione lineare e la matrice binaria associata $m \times m$ è indicata come P_σ.
Il Prover ora calcola i seguenti valori

$$z = P_\sigma x$$

$$u_o = f(\sigma \parallel Ak;\, r_o)$$

$$u_1 = f(z \parallel P_\sigma k;\, r_1)$$

E invia u_o, u_1 come commitment.

2. Il Verifier genera un valore causale $\alpha \in \mathbb{Z}_q$ e lo invia al Prover.
3. Il Prover utilizza il valore appena ricevuto per calcolare $\beta = P_\sigma(k + \alpha x)$ e lo invia al Verifier.

[16] Il peso di Hamming indica il numero di elementi diversi da zero.

4. Il Verifier genera un nuovo valore casuale $c \in \{0, 1\}$ che costituisce la challenge vera e propria del protocollo e lo invia al Prover.
5. Il Prover invia due diversi messaggi a seconda del valore di c, ovvero:
    - Se $c = 0$ invia σ, r_o
    - Se $c = 1$ invia z, r_1

Anche i controlli eseguiti dal Verifier si dividono in due casi diversi a seconda del valore di c, e confermano l'esistenza della proprietà di completezza.

- Se $c = 0$ il Verifier controlla che $\sigma \in S_m$ e che la seguente equazione sia vera

$$u_0 = f(\sigma \parallel AP_\sigma{}^{-1}\beta - \alpha y;\ r_o)$$

- Se $c = 1$ il Verifier controlla che $z \in \{0, 1\}^m$, che il suo peso di Hamming sia uguale a $m/2$ e che la seguente equazione sia vera

$$u_1 = f(z \parallel \beta - \alpha z;\ r_1)$$

Per verificare la proprietà di correttezza si dimostra che un Prover disonesto riesce ad ingannare il Verifier con una probabilità di $(q + 1)/2q \approx 1/2$.
Infatti, le interrogazioni a cui il Prover deve essere in grado di rispondere sono date dalla combinazione di tutti i possibili $c \in \{0, 1\}$ e $\alpha \in \{0, \dots, q - 1\}$, ovvero $2q$. Tra queste, il Prover disonesto è in grado di rispondere correttamente a tutte le challenge con $b = 0$, utilizzando una pseudo-chiave x' tale che $Ax' = y$, e contemporaneamente alle challenge $(\alpha = 0, c = 1)$ utilizzando uno z casuale, e $(\alpha = \alpha_0, c = 1)$ con α_0 un valore scelto prima di fare il commitment. Queste combinazioni corrispondono a $q + 1$ delle interrogazioni possibili e sono il numero massimo che è in grado di rispondere.
Infatti, si può dimostrare [50] che un Prover in grado di risolvere un numero maggiore di interrogazioni è anche in grado di risolvere il problema SIS, oppure rompere la funzione di commitment f, la quale però è corretta per definizione. Perciò ne consegue che $(q + 1)/2q \approx 1/2$ è la probabilità massima di successo per un dimostratore disonesto.
È bene notare che questa probabilità corrisponde all'esecuzione di un round del protocollo, per questo è necessario eseguire l'algoritmo ripetutamente, per più rounds, in modo da diminuire la probabilità ad un livello di sicurezza ottimale.

La proprietà di conoscenza zero, infine, è dimostrata tramite l'uso di un simulatore in grado di restituire una trascrizione identica a quella di una normale conversazione tra Prover e Verifier. In particolare, il simulatore genera i valori r_0, r_1 seguendo il protocollo, e poi genera i valori successivi in base a un valore casuale $c' \in \{0, 1\}$.

- Se $c' = 0$ il simulatore seleziona k e σ come da protocollo ed estrae una pseudo chiave x dalla seguente equazione

  $$Ax = y \; mod \; q$$

  Con la pseudo chiave appena calcolata genera i valori u_o e u_1 e li invia al Verifier. Ricevuto il valore α può calcolare $\beta = P_\sigma(k + \alpha x)$.
  A questo punto il simulatore può rispondere alla challenge $c = c' = 0$ con i valori (σ, r_0) che risulteranno corretti per il Verifier.

  Nel caso in cui il Verifier invii la challenge $c = 1$ il simulatore dovrà tornare al punto iniziale e ripetere il protocollo finché le challenge non coincidono.

- Se $c' = 1$ il simulatore seleziona un valore binario x con peso di Hamming uguale a $m/2$ e un σ come da protocollo. Successivamente, ricevuto α il simulatore calcola u_0, u_1 e $\beta = P_\sigma(k + \alpha x)$.
  A questo punto il simulatore può rispondere in caso di challenge $c = c' = 1$ rivelando $z = P_\sigma x$ che verrà accettato.

  Anche in questo caso, se le challenge non coincidono il simulatore ripeterà il protocollo da capo.

La trascrizione così ottenuta risulta indistinguibile da quella reale.

Per quanto riguarda la matrice SIS A, normalmente la sua memorizzazione richiederebbe uno spazio dell'ordine di n^2, ma grazie all'utilizzo di una particolare tipologia di reticoli, chiamata reticoli ideali[17], è possibile ridurre questo spazio nell'ordine di n e contemporaneamente aumentare la velocità computazionale.

Questo protocollo è un importante esempio dell'utilizzo dei reticoli per migliorare un algoritmo già esistente cambiandone le basi matematiche, e aumentando così la sicurezza.

[17] I reticoli ideali sono una particolare classe di reticoli chiusi rispetto alla moltiplica di anelli.

Conclusioni

Con la presente tesi si è cercato di fornire una panoramica generale delle dimostrazioni a conoscenza zero, con particolare enfasi sui loro utilizzi nei protocolli di identificazione. Nel farlo, sono stati presentate le dimostrazioni a conoscenza zero con le loro proprietà generali ed alcuni esempi esplicativi. La definizione di dimostrazione a conoscenza zero è stata introdotta tramite i sistemi di prova interattivi, per poi passare ad una formulazione più rigorosa. Tra le dimostrazioni interattive a conoscenza zero è stata posta particolare attenzione ai protocolli sigma, i quali sono particolarmente adatti ad essere utilizzati come protocolli di identificazione, vista la loro proprietà di conoscenza e la loro semplicità. Sono stati poi presentate le dimostrazioni a conoscenza zero non-interattive, con una metodologia per trasformare i protocolli interattivi in protocolli non-interattivi.

Dopo aver definito le basi teoriche delle dimostrazioni a conoscenza zero la trattazione si è spostata sui protocolli di identificazione, descrivendoli sommariamente, per poi fondere i due concetti. Infatti, lo scopo della tesi è quello di mostrare al lettore le potenzialità delle dimostrazioni a conoscenza zero e di come essi possano migliorare i protocolli di identificazione, soprattutto da un punto di vista della garanzia della privacy. In particolare, la proprietà di conoscenza zero permette di ridurre al minimo le informazioni scambiate durante il protocollo proteggendo così l'identità del Prover. Nessuna informazione viene divulgata (se non quella strettamente necessaria ad autenticare l'utente), neanche l'informazione che l'utente si è autenticato ad un determinato Verifier.

Il seguente elaborato vuole anche essere un buon punto di partenza per chi desidera studiare in modo più approfondito le dimostrazioni a conoscenza zero, fornendo conoscenze di base e riferimenti a pubblicazioni più complete.

Bibliografia

[1] J.-J. Quisquater, L. Quillou e T. Berson, «**How to Explain Zero-Knowledge Protocols to Your Children**» in *CRYPTO '89: Proceedings of conference on Advances in cryptology*, 1989.

[2] S. Goldwasser, S. Micali e C. Rackoff, «**The Knowledge Complexity of Interactive Proof-Systems**» in *STOC '85: Proceedings of the 17th annual ACM symposium on Theory of computing*, 1985.

[3] L. Babai, «**Trading Group Theory for Randomness**» in *STOC '85: Proceedings of the 17th annual ACM symposium on the Theory of Computing*, 1985.

[4] S. Goldwasser e M. F. Sipser, «**Private Coins versus Public Coins in Interactive Proof Systems**» in *STOC '86: Proceedings of the 18th annual ACM symposium on Theory of computing*, 1986.

[5] O. Goldreich, «**A zero-knowledge proof that a two-prime moduli is not a Blum integer**» 1985.

[6] O. Goldreich, S. Micali e A. Wigderson, «**Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems**» in *Journal of the ACM*, 1991.

[7] R. Impagliazzo e M. Yung, «**Direct Minimum-Knowledge Computations**» in *CRYPTO '87: Conference on Advances in cryptology*, 1987.

[8] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. Kilian, S. Micali e P. Rogaway, «**Everything Provable is Provable in Zero-Knowledge**» in *CRYPTO '88: Proceedings of conference on Advances in cryptology*, 1990.

[9] C. Lund, L. Fortnow, H. Karloff e N. Nisan, «**Algebraic Methods for Interactive Proof**» in *Journal of the ACM*, 1992.

[10] A. Shamir, «**IP = PSPACE**» in *Journal of the ACM*, 1992.

[11] M. Ben-Or, S. Goldwasser, J. Kilian e A. Wigderson, «**Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions**» in *STOC '88: Proceedings of the 20th annual ACM symposium on Theory of computing*, 1988.

[12] U. Feige e A. Shamir, «**Witness Indistinguishable and Witness Hiding Protocols**» in *STOC '90: Proceedings of the 22nd annual ACM symposium on Theory of Computing*, 1990.

[13] M. Blum, P. Feldman e S. Micali, «**Non-Interactive Zero-Knowledge and Its Applications**» in *STOC '88: Proceedings of the 20th annual ACM symposium on Theory of computing*, 1988.

[14] A. C.-C. Yao, «**How to generate and exchange secrets**» in *27th Annual IEEE Symposium on Foundations of Computer Science*, 1986.

[15] O. Goldreich, S. Micali e A. Wigderson, «**How to play ANY mental game or a completeness theorem for protocols with honest majority**» *19th Annual ACM Symposium on Theory of Computing,* pp. 218-229, 1987.

[16] A. Fiat e A. Shamir, «**How to Prove Yourself: Practical Solution to Identification and Signature Problems**» in *CRYPTO '86: Proceedings on Advances in cryptology*, 1987.

[17] L. C. Guillou e J.-J. Quisquater, «**A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory**» in *EUROCRYPT 1988: International conference on Andvances in cryptology*, 1988.

[18] R. Cramer, **Modular Design of Secure yet Practical Cryptographic Protocols**, 1997.

[19] A. Farneti, **I protocolli Zero-Knowledge**, 2021.

[20] O. Goldreich, **Foundations of Cryptography - Basic Tecniques**, Cambridge: Cambridge University Press, 2001.

[21] D. Wong, **Real-World Cryptography**, Manning Publications Co., 2021.

[22] J. A. Garay, P. MacKenzie e K. Yang, «**Strengthening Zero-Knowledge Protocols Using Signatures**» *Journal of Cryptology,* pp. 169-209, 2006.

[23] I. Damgård, «**On Σ-protocols**» in *CPT 2010*, 2010.

[24] B. Schoenmakers, **Lecture Notes - Cryptographic Protocols**, 2022.

[25] R. Cramer, I. Damgård e P. MacKenzie, «**Efficient Zero-Knowledge Proofs of Knowledge without Intractability Assumptions**» in *PKC 2000: International workshop on Public key cryptography*, 2000.

[26] M. Bellare e O. Goldreich, «**On Defining Proofs of Knowledge**» in *CRYPTO '92: Proceedings of the 12th annual international cryptology conference on Advances in cryptology*, 1992.

[27] M. Blum, A. De Santis, S. Micali e G. Persiano, «**Noninteractive Zero-Knowledge**» in *SIAM Journal on Computing*, 1991.

[28] U. Feige, D. Lapidot e A. Shamir, «**Multiple Noninteractive Zero Knowledge Proofs Under General Assumptions**» in *SIAM Journal on Computing*, 1999.

[29] O. Goldreich, S. Micali e A. Wigderson, «**How to Play any Mental Game or a Completeness Theorem for Protocols with Hones Majority**» in *STOC '87: Proceedings of the 19th annual ACM symposium on Theory of computing*, 1987.

[30] M. Ben-Or, S. Goldwasser e A. Wigderson, «**Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation**» in *STOC '88: Proceedings of the 20th annual ACM symposium on Theory of computing*, 1988.

[31] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer e M. Virza, «**Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs**» in *IEEE Symposium on Security and Privacy*, 2015.

[32] M. Bellare e P. Rogaway, «**Random Oracles are Practical: a Paradigm for Designing Efficient Protocols**» in *CCS '93: Proceedings on the 1st ACM conference on Computer and communications security*, 1993.

[33] I. Damgård, N. Fazio e A. Nicolosi, «**Non-interactive Zero-Knowledge from Homomorphic Encryption**» in *TCC 2006: Theory of cryptography conference*, 2006.

[34] F. Boudot, «**Efficient Proofs that a Committed Number Lies in an Interval**» in *EUROCRYPT 2000: International conference on Advances in cryptology*, 2000.

[35] J. Camenisch, R. Chaabouni e A. Shelat, «**Efficient Protocols for Set Membership and Range Proofs**» in *ASIACRYPT 2008: International conference on Advances in Cryptology*, 2008.

[36] D. Boneh e X. Boyen, «**Short Signatures Without Random Oracles**» in *EUROCRYPT 2004: International conference on Advances in cryptology*, 2004.

[37] J. H. Cheon, «**Security Analysis of the Strong Diffie-Hellman Problem**» in *EUROCRYPT 2006: International conference on Advances in cryptology*, 2006.

[38] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille e G. Maxwell, «**Bulletproofs: Short Proofs for Confidential Transactions and More**» in *IEEE Symposium on Security and Privacy*, 2018.

[39] L. Lamport, «**Password Authentication with Insecure Communication**» in *Communications of the ACM*, 1981.

[40] H. Delfs e H. Knebl, **Introduction to Cryptography: Principles and Applications**, Seconda edizione a cura di, Springer, 2007.

[41] U. Feige, A. Fiat e A. Shamir, «**Zero-Knowledge Proofs of Identity**» in *Journal of Cryptology*, 1988.

[42] P. W. Shor, «**Polynomial time algorithms for discrete logarithms and factoring on a quantum computer**» in *ANTS 1994: Algorithmic Number Theory*, 1994.

[43] J. Stern, «**A new Identification Scheme Based on Syndrome Decoding**» in *CRYPTO '93: International cryptology conference on Advances in cryptology*, 1993.

[44] N. T. Courtois, «**Efficient Zero-Knowledge Authentication Based on a Linear Algebra Problem MinRank**» in *ASIACRYPT 2001: International conference on Advances in cryptology*, 2001.

[45] J.-C. Faugère, F. Levy-dit-Vehel e L. Perret, «**Cryptanalysis of MinRank**» in *CRYPTO 2008: International cryptology conference on Advances in cryptology*, 2008.

[46] A. Shamir, «**An Efficient Identification Scheme Based on Permuted Kernels (extended abstract)**» *Advanced in Cryptology - CRYPTO' 89,* pp. 606-609, 1990.

[47] T. Baritaud, M. Campana, P. Chauvaud e H. Gilbert, «**On the Security of the Permuted Kernel Identification Scheme**» in *CRYPTO '92: Proceedings of the 12th annual international cryptology conference on Advances in cryptology*, 1992.

[48] W. Beullens, J.-C. Faugère, E. Koussa, G. Macario-Rat, J. Patarin e L. Perret, «**PKP-Based Signature Scheme**» in *INDOCRYPT 2019: International conference on Progress in Cryptology*, 2019.

[49] D. Micciancio. [Online]. Available: https://cseweb.ucsd.edu/~daniele/LatticeLinks/index.html. [Consultato il giorno 12-03-2023].

[50] P.-L. Cayrel, R. Lindner, M. Rückert e R. Silva, «**Improved Zero-Knowledge Identification with Lattices**» in *ProvSec 2010: International conference on Provable Security*, 2010.

[51] P.-L. Cayrel e P. Véron, «**Improved code-based identification scheme**» *ArXiv,* 2010.

[52] A. Kawachi, K. Tanaka e K. Xagawa, «**Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems**» in *ASIACRYPT 2008: International conference on Advances in cryptology*, 2008.

[53] O. Goldreich, «**A Short Tutorial of Zero-Knowledge**» 2010.

[54] G. Couteau, **Zero-Knowledge Proofs for Secure Computation***,* 2017.