Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Matematica
Corso di Laurea in Matematica

CRITTOGRAFIA A CHIAVE PRIVATA
Protocollo a Conoscenza Zero

Relatore:
Prof. Davide Aliffi

Presentata da:
Alberto Vecchione

Anno Accademico 2016/2017

Indice

Introduzione

La *crittografia* è la branca della crittologia che studia le tecniche di rappresentazione di messaggi in una forma tale che le informazioni in essi contenute possano essere lette solo dal destinatario. In particolare, queste tecniche, permettono a due soggetti (chiamati in genere *Alice* e *Bob*) di scambiarsi messaggi senza che vengano compresi da un eventuale ascoltatore (chiamato *Eva*).

Nel primo capitolo, dopo aver dato delle nozioni base di probabilità, daremo la definizione di *schema crittografico a chiave privata*, utilizzando l'approccio "classico", dove uno degli scopi fondamentali è quello di non trasmettere nessuna "informazione" ad un eventuale avversario. Quindi, ci concentreremo sulla sicurezza di questi schemi crittografici, in particolare sulla *sicurezza computazionale*.

Nel secondo capitolo, invece, daremo una definizione alternativa di schema crittografico a chiave privata, che dimostreremo essere equivalente a quella di schema crittografico sicuro. Questa definizione "traduce" la teoria descritta nel capitolo precedente in termini di *conoscenza*, cioè non ci preoccupiamo più di tener nascoste le informazioni, ma di non "trasmettere" conoscenza ad un eventuale ascoltatore. Questo nuovo approccio si basa su un metodo, chiamato *protocollo (o dimostrazione) a conoscenza zero*, che permette ad un soggetto di dimostrare, ad un altro soggetto, che una certa affermazione è vera, senza rivelare nient'altro oltre alla veridicità della stessa.
Informalmente, in una *dimostrazione a conoscenza zero* ci sono due parti, Alice e Bob. Alice vuole convincere Bob che una certa affermazione è vera; per esempio, Alice vuole convincere Bob che un numero N è prodotto di due primi p, q. Una possibile soluzione potrebbe essere quella di comunicare p e q a Bob. In questo modo, Bob può verificare da solo che p e q sono primi (utilizzando ad esempio un test di primalità), poi moltiplicando

i due numeri potrà verificare se effettivamente il loro prodotto è uguale ad N. Ma questa soluzione rivela p e q. È necessario? La risposta è no. Vedremo che, utilizzando una dimostrazione a conoscenza zero, Alice può convincere Bob che questa affermazione è vera senza rivelare i fattori p e q.

Infine, nel terzo capitolo, faremo un esempio di applicazione di questo protocollo. In particolare, descriveremo un'estensione di *Bitcoin*, chiamata *Zerocoin*, che, tramite l'utilizzo del protocollo a conoscenza zero, garantisce un certo livello di privacy ai suoi utenti.

Capitolo 1

Sicurezza computazionale

1.1 Richiami di probabilità

Introduciamo alcune nozioni base di probabilità, che saranno indspensabili per capire meglio quello che andremo a studiare in questo elaborato.
Sia (Ω, Pr) uno *spazio di probabilità discreto*, dove:

- Ω è un insieme non vuoto, finito o numerabile;
- Pr è una *probabilità*, cioè una funzione

$$Pr : \mathcal{P}(\Omega) \to [0, 1] \tag{1.1}$$

dove $\mathcal{P}(\Omega)$ è l'insieme delle parti di Ω, tale che $Pr[\Omega] = 1$ ed è σ-additiva.

Sia un evento $A \in \mathcal{P}(\Omega)$, con $Pr[A]$ denotiamo la probabilitá che si verifichi tale evento. Inoltre, diremo che A è un evento *non trascurabile* se $Pr[A] > 0$.

Diamo ora la definizione di *probabilità condizionata* che ci permetterà di analizzare come il verificarsi di un evento B influenzi la probabilità di un altro evento A.

Definizione 1.1. Siano A e B due eventi con B non trascurabile, allora:

$$Pr[A|B] := \frac{Pr[A \bigcup B]}{Pr[B]}, \qquad A, B \in \mathcal{P}(\Omega) \tag{1.2}$$

é detta *probabilità di A condizionata a B*.

Notiamo che, quando $Pr[B] = 0$, la probabilità condizionata non è definita. In questo elaborato, abusiamo leggermente della notazione e definiamo

$$Pr[A|B] = Pr[A] \tag{1.3}$$

quando $Pr[B] = 0$.

1.2 Crittografia a chiave privata

Consideriamo il seguente problema: abbiamo due parti, Alice e Bob. Alice vuole mandare un messaggio, privato, che chiameremo *plaintext*, a Bob attraverso un canale insicuro. In particolare, Alice e Bob vorrebbero mantenere la loro privacy anche davanti ad un avversario, Eva, che ascolta tutti i messaggi inviati nel canale. In che modo possiamo risolvere questo problema?
Una possibile soluzione potrebbe essere questa: prima di iniziare la conversazione, Alice e Bob scelgono un "codice segreto" che poi useranno per comunicare.
Un codice segreto consiste in una *chiave* e in due algoritmi *Enc*, *Dec*, dove entrambi gli algoritmi utilizzano la chiave, il primo per cifrare il *testo in chiaro* e il secondo per decifrare il *testo cifrato*.
Quindi, adesso Alice può usare la chiave per cifrare il messaggio, poi manda il testo cifrato a Bob. Bob, riceve il testo cifrato, usa la chiave per decifrarlo e ottiene il messaggio originale.
Inoltre, dobbiamo considerare un altro algoritmo, *Gen*, che viene usato da Alice e Bob per generare la chiave k, che verrà poi usata per cifrare e decifrare i messaggi. La prima questione che ci poniamo è: quali informazioni devono essere pubbliche e quali private? Negli approcci classici, vengono tenuti nascosti tutti e tre gli algoritmi (Gen, Enc, Dec) e la chiave k; l'idea dietro a questo ragionamento è: meno informazioni diamo all'avversario, e più sarà complicato per lui rompere lo schema. Oggi, invece, ci basiamo sul seguente principio per creare sistemi crittografici a chiave privata:

Principio di Kerckhoffs. *La sicurezza di un crittosistema non deve dipendere dal tener celato il crittosistema stesso, ma deve dipendere solo dalla segretazza della chiave.*

Quindi, gli algoritmi *Gen*, *Enc* e *Dec* possono essere resi pubblici, mentre l'unica cosa che deve essere tenuta segreta è la chiave k.

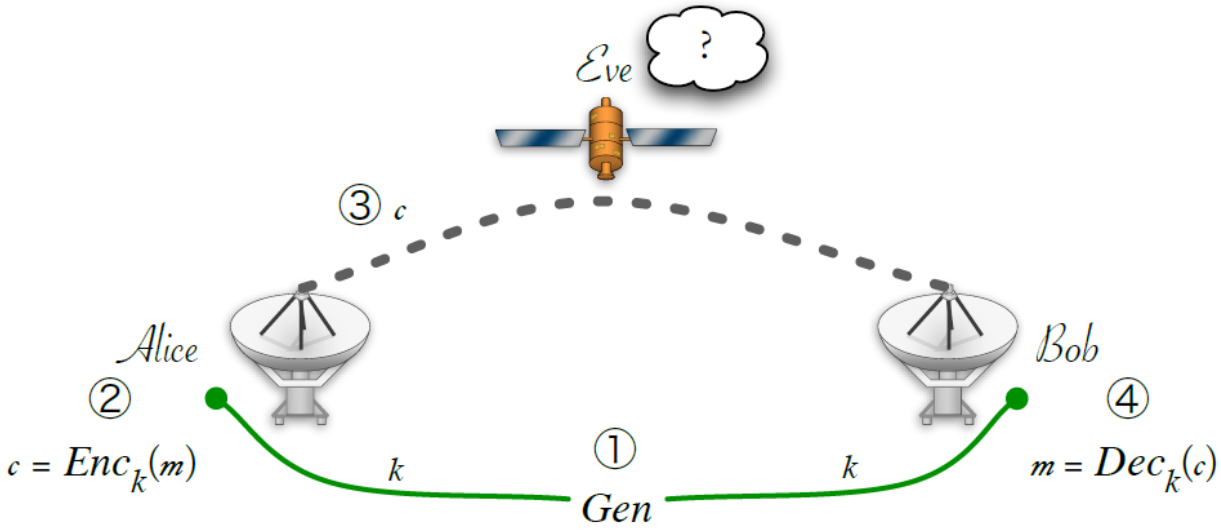


Figura 1.1: Illustrazione di uno schema crittografico a chiave privata. In primo luogo, *Gen* genera una chiave k che deve restare segreta, quindi verrà comunicata ad Alice e Bob tramite un canale sicuro (rappresentato con la linea inferiore in figura). Poi Alice codifica il messaggio m in un testo cifrato c e lo invia tramite il canale insicuro (linea superiore in figura). Bob riceve il messaggio codificato e per recuperare il messaggio originale m decodifica c usando la chiave k. Eva non impara nulla su m, eccetto forse la sua lunghezza.

Definizione 1.2. (Schema crittografico a chiave privata). La terna (Gen, Enc, Dec) si dice uno *schema crittografico a chiave privata* sullo spazio dei messaggi $\mathcal{M}$ e sullo spazio delle chiavi $\mathcal{K}$ se vale quanto segue:

1. *Gen* (algoritmo che genera le chiavi) è un algoritmo probabilistico che restituisce una chiave k tale che $k \in \mathcal{K}$. Denotiamo con $k \leftarrow Gen$ il processo che genera una chiave k;

2. *Enc* (algoritmo che codifica) è un algoritmo, potenzialmente probabilistico, che dati in input una chiave $k \in \mathcal{K}$ e un messaggio $m \in \mathcal{M}$, restituisce un testo cifrato c. Denotiamo con $c \leftarrow Enc_k(m)$ l'operazione di cifratura del testo m tramite l'algoritmo *Enc* con la chiave k;

3. *Dec* (algoritmo che decodifica) è un algoritmo deterministico che dati in input una chiave k e un testo cifrato c restituisce un messaggio $m \in \mathcal{M}$. Denotiamo con $m \leftarrow Dec_k(c)$ l'operazione di decifratura del testo c tramite l'algoritmo *Dec* con la chiave k.

4. Per ogni $m \in \mathcal{M}$,

$$Pr[k \leftarrow Gen : Dec_k(Enc_k(m)) = m] = 1 \tag{1.4}$$

Osservazione 1. Notiamo che la definizione, appena data, di *schema crittografico a chiave privata* non specifica alcuna proprietà di segretezza (o privacy), il solo requisito non banale è che l'algoritmo *Dec* recuperi in modo unico i messaggi codificati con *Enc* (se questi algoritmi vengono eseguiti con la stessa chiave $k \in \mathcal{K}$).

1.3 Efficienza di calcolo

Ci interessa formalizzare il significato di algoritmo che calcola una funzione, iniziamo con il caso deterministico, per poi estenderlo al caso probabilistico.

Definizione 1.3. (Algoritmo) Un *algoritmo* è una macchina di Turing deterministica che prende in input stringhe nell'alfabeto $\sum = \{0,1\}$, e restituisce stringhe sempre nell'alfabeto $\sum$.

Osservazione 2. Una *macchina di Turing* è una macchina ideale che manipola i dati contenuti su un nastro di lunghezza potenzialmente infinita, secondo un insieme prefissato di regole ben definite. In altre parole, è un modello astratto che definisce una macchina in grado di eseguire algoritmi e dotata di un nastro potenzialmente infinito su cui leggere o scrivere simboli.

Definizione 1.4. (Tempo di esecuzione) Un algoritmo $\mathcal{A}$ si dice eseguito in tempo $T(n)$ se per ogni $x \in \{0,1\}^*$, $\mathcal{A}(x)$ termina in massimo $T(|x|)$ passi[1]. In particolare, $\mathcal{A}$ si dice eseguito in *tempo polinomiale*, se esiste una costante c tale che $\mathcal{A}$ viene eseguito in tempo $T(n) = n^c$.

Definizione 1.5. (Calcolo deterministico) Un algoritmo $\mathcal{A}$ si dice che calcola una funzione $f : \{0,1\}^* \to \{0,1\}^*$ se per ogni $x \in \{0,1\}^*$, $\mathcal{A}$, preso in input x, restituisce $f(x)$.

[1] $|x|$ è la lunghezza della stringa x

Diciamo che un algoritmo è *efficiente* se viene eseguito in tempo polinomiale.

Osservazione 3. Prendere il tempo polinomiale come limite massimo per definire l'efficienza di un algoritmo potrebbe non essere corretto, poichè in caso di polinomio di grado elevato, il calcolo potrebbe non essere efficiente in pratica. Tuttavia, l'esperienza ci suggerisce che gli algoritmi che si eseguono in tempo polinomiale sono efficienti, cioè nella maggior parte dei casi tempo polinomiale significa "cubico o meglio ancora".

Una naturale estensione del cacolo deterministico è quella di consentire ad un algoritmo di accedere ad una fonte di "lanci di monete" casuali. Permettere questa libertà è plausibile (come è concepibile generare tali lanci in pratica), e questo ci garantisce una maggiore efficienza nel calcolo di alcuni compiti. Per esempio, come abbiamo già visto, uno dei principi di Kerckhoff dice che tutti gli algoritmi di uno schema devono essere pubblici; allora se l'algoritmo che genera le chiavi non usa una fonte di casualità, Eva potrebbe essere in grado di generare la stessa chiave che utilizzano Alice e Bob. Permettere questa ulteriore libertà agli algoritmi, estende la precedente definizione di calcolo come segue:

Definizione 1.6 (Algoritmo probabilistico)**.** Un *algoritmo probabilistico*, abbreviato in PPT (Probabilistic Polynomial-time Turing machine), è una macchina di Turing che ha un nastro supplementare contenente una stringa infinita di bit casuali. Ogni bit del nastro casuale viene scelto in modo uniforme e indipendente.

Equivalentemente, un algoritmo probabilistico è una macchina di Turing che ha accesso ad un oracolo casuale che, su richiesta, restituisce un bit random.
Per definire l'efficienza dobbiamo chiarire il concetto di *tempo di esecuzione* per un algoritmo probabilistico, poichè potrebbe dipendere dalle scelte del nastro casuale. Quindi, definiremo il tempo di esecusione di un algoritmo probabilistico come limite superiore rispetto a tutte le possibili sequenze casuali, generate dal nastro casuale.

Definizione 1.7 (Tempo di esecuzione)**.** Un algoritmo probabilistico $\mathcal{A}$, si dice eseguito in tempo $T(n)$ se per ogni $x \in \{0,1\}^*$, e per ogni nastro casuale, $\mathcal{A}(x)$ termina in al massimo $T(|x|)$ passi. In particolare, $\mathcal{A}$ si dice eseguito in tempo polinomiale se esiste una costante c tale che $\mathcal{A}$ viene eseguito in tempo $T(n) = n^c$.

Come prima, un algoritmo è *efficiente* se viene eseguito in tempo polinomiale.

Infine, dobbiamo estendere anche la definizione di calcolo per un algoritmo probabilistico.

Definizione 1.8 (Calcolo probabilistico)**.** Diciamo che un algoritmo probabilistico $\mathcal{A}$ calcola una funzione $f : \{0,1\}^* \to \{0,1\}^*$ se per ogni $x \in \{0,1\}^*$, $\mathcal{A}$ prende in input x e restituisce $f(x)$ con probabilità uguale a 1.

Notiamo che, nella pratica, alcuni degli algoritmi probabilistici comunemente utilizzati (per esempio, i test di primalità) possono commettere degli errori con probabilità molto basse. In questo elaborato non consideriamo questi rari casi e supponiamo che gli algoritmi probabilistici funzionino sempre correttamente.
Utilizzeremo gli algoritmi probabilistici eseguiti in tempo polinomiale come modello di efficienza. Quando scriveremo *probabilistic polynomial-time Turing machine (PPT)* oppure *algoritmo probabilistico efficiente*, faremo riferimento alla classe di algoritmi citati sopra. Date queste nozioni, possiamo dare la definizione di *schema crittografico a chiave privata efficiente*.

Definizione 1.9 (Schema crittografico a chiave privata efficiente)**.** Una terna di algoritmi (Gen, Enc, Dec) si dice uno *schema crittografico a chiave privata efficiente* se vale quanto segue:

1. $k \leftarrow Gen(1^n)$ è un **PPT** tale che per ogni $n \in \mathbb{N}$, genera una chiave k di lunghezza n.

2. $c \leftarrow Enc_k(m)$ è un **PPT** che dati k e $m \in \{0,1\}^n$ produce un testo cifrato c.

3. $m \leftarrow Dec_k(c)$ è un **PPT** che dati k e un testo cifrato c restituisce un messaggio $m \in \{0,1\}^n$.

4. Per ogni $n \in \mathbb{N}$, $m \in \{0,1\}^n$, data una chiave k, generata da $Gen(1^n)$, si ha:

$$Pr[Dec_k(Enc_k(m)) = m] = 1 \tag{1.5}$$

Notiamo che l'algoritmo Gen ha come input 1^n, chiamato *parametro di sicurezza*, che rappresenta una stringa di n copie di 1 (es. $1^4 = 1111$). Questo parametro di sicurezza è usato per garantire una certa "sicurezza" allo schema, parametri più grandi corrispondono a schemi più sicuri. Inoltre, il parametro di sicurezza stabilisce il tempo di esecuzione

dell'algoritmo *Gen*, e la massima lunghezza della chiave k, quindi di conseguenza anche il tempo di esecuzione degli algoritmi *Enc* e *Dec*.
Notiamo anche che nella definizione precedente non compaiono gli spazi dei messaggi $\mathcal{M}$ e delle chiavi $\mathcal{K}$, poichè supponiamo che entrambi siano stringhe di bit. In particolare, il parametro di sicurezza 1^n, definisce schemi che gestiscono messaggi di n bit. Nel seguito di questo elaborato, per ridurre la notazione, quando diremo *schema crittografico a chiave privata* ci riferiremo sempre ad uno schema crittografico efficiente nel senso della definizione precedente.

1.4 Avversario efficiente

Quando modelliamo gli avversari, usiamo una definizione più "debole" di calcolo efficiente. In particolare, non richiediamo che l'avversario sia una macchina a dimensione costante, cioè consentiamo all'avversario di essere *non uniforme*. Come prima, permettiamo all'avversario di usare "lanci casuali di monete" e richiediamo che il suo tempo di esecuzione sia limitato da un polinomio.

Definizione 1.10 (**PPT** non uniforme)**.** Un *algoritmo probabilistico non uniforme* (abbreviato in n.u. **PPT**) A è una sequenza di macchine probabilistiche $A = \{A_1, A_2, ...\}$ per le quali esiste un polinomio d tale che $|A_i| < d(i)$ e il tempo di esecuzione di A_i è minore di $d(i)$. Denotiamo con $A(x)$ la distribuzione ottenuta eseguendo $A_{|x|}(x)$.

Nel seguito, per ogni algoritmo $\mathcal{A}$ considereremo come avversario un *n.u. PPT*, anche se non specificato.

1.5 Funzioni unidirezionali

Queste funzioni sono alla base della crittografia, poichè uno schema crittografico deve essere costruito in modo tale che:

- deve essere facile generare c dati m e k, ma
- deve essere difficile trovare m e k dato c.

Quindi, abbiamo bisogno di funzioni facili da calcolare, ma difficili da invertire, chiamate *funzioni unidirezionali*. Esistono diversi modi per definire questa classe di funzioni, noi ci

accontenteremo della definizione di *funzione unidirezionale forte*. Prima di formalizzare questa definizione abbiamo bisogno di introdurre le *funzioni trascurabili*.

Definizione 1.11 (Funzione trascurabile)**.** Una funzione $\epsilon(n)$ è trascurabile se per ogni costante c, esiste n_0 tale che per ogni $n > n_0$, $\epsilon(n) \leq \frac{1}{n^c}$.

Definizione 1.12 (Funzione unidirezionale forte)**.** Una funzione $f : \{0,1\}^* \to \{0,1\}^*$ è una *funzione unidirezionale forte* se soddisfa le seguenti condizioni:

1. **(Facile da calcolare)** Esiste una PPT $\mathcal{C}$ che calcola $f(x)$ per ogni $x \in \{0,1\}^*$;
2. **(Difficile da invertire)** Ogni tentativo efficiente di invertire f, dato un input casuale, riesce solo con probabilità trascurabile[2]. Formalmente, preso un qualsiasi avversario A, esiste una funzione trascurabile $\epsilon()$ tale che $\forall x \in \{0,1\}^n$ di lunghezza $n \in \mathbb{N}$ si ha che:
$$Pr[f(A(1^n, f(x))) = f(x)] \leq \epsilon(n). \tag{1.6}$$

Notiamo che l'algoritmo A riceve l'input aggiuntivo 1^n, questo permette all'algoritmo di avere un tempo di esecuzione polnomiale in $|x|$.

1.6 Indistinguibilità computazionale

Ci interessa la nozione di *indistinguibilità computazionale*, perchè ci permetterà di formalizzare il significato dell'affermazione: "due distribuzioni di probabilità sembrano le stesse dal punto di vista di un avversario computazionalmente limitato". Prima di dare definizioni formali, illustriamo l'idea alla base di questo concetto con il seguente esempio:

Esempio 1.1. Guardate la Figura 1.2 per non più di due secondi. I due rettangoli contengono lo stesso numero di cerchi e posizionati allo stesso modo?
Adesso supponiamo di ripetere l'esperimento, ma questa volta guardando la figura per dieci secondi, invece che due. Immaginiamo ora di spendere dieci minuti, invece che dieci secondi. Se viene dato solo un piccolo lasso di tempo i due rettangoli sembrano indistinguibili l'uno dall'altro. Prendendo sempre più tempo per analizzare la figure, è

[2]In particolare, è esclusa la possibilità di invertire f mediante una ricerca esaustiva, ovvero per "forza bruta"

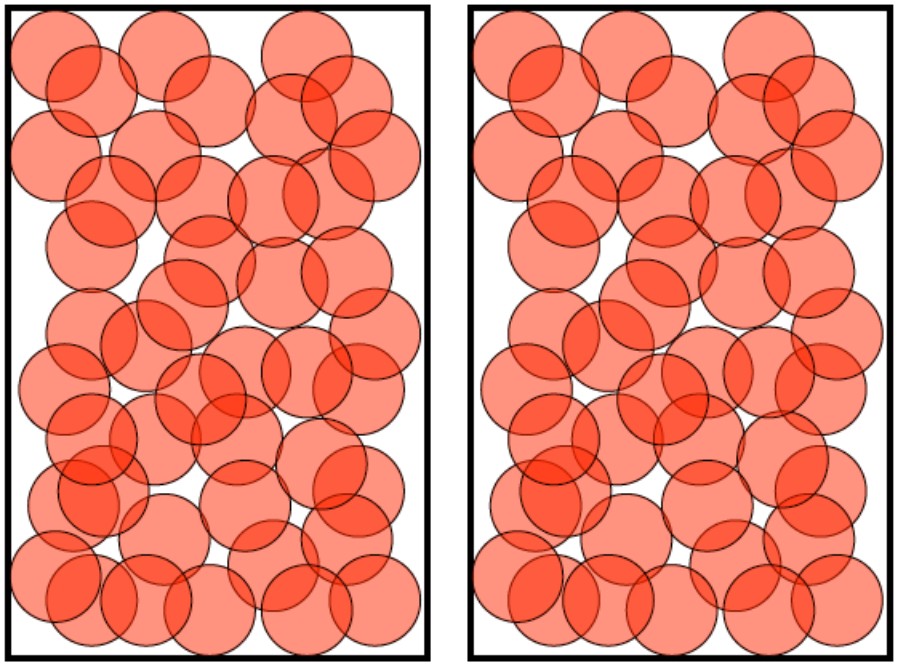


Figura 1.2: I cerchi sono posizionati allo stesso modo?

possibile trovare alcune piccole differenze tra destra e sinistra. In generale, anche se due distribuzioni di probabilità sono completamente disgiunte, può essere che un osservatore (con un tempo di elaborazione limitato) non sappia distinguerle.

La definizione formale considera sequenze, chiamate insiemi, di distribuzioni di probabilità.

Definizione 1.13 (Insiemi di distribuzioni di probabilità)**.** Una sequenza $\{X_n\}_{n\in\mathbb{N}}$ è chiamata *insieme di distribuzioni di probabilità* se per ogni $n \in \mathbb{N}$. X_n è una distribuzione di probabilità su $\{0,1\}^n$.

Definizione 1.14 (Indistinguibilità computazionale)**.** Siano $\{X_n\}_{n\in\mathbb{N}}$ e $\{Y_n\}_{m\in\mathbb{N}}$ due insiemi dove X_n, Y_n sono distribuzioni su $\{0,1\}^n$. Diciamo che $\{X_n\}_{n\in\mathbb{N}}$ e $\{Y_n\}_{n\in\mathbb{N}}$ sono *indistinguibili computazionalmente* (abbreviato con $\{X_n\}_{n\in\mathbb{N}} \approx \{Y_n\}_{n\in\mathbb{N}}$) se per ogni n.u. PPT D (chiamata *distinguisher*), esiste una funzione trascurabile $\epsilon()$ tale che per ogni $n \in \mathbb{N}$

$$|Pr[D(X_n) = 1] - Pr[D(Y_n) = 1]| < \epsilon(n). \tag{1.7}$$

Per semplificare la notazione, diremo che D *distingue le distribuzioni* X_n e Y_n con probabiltà ϵ se

$$|Pr[D(X_n) = 1] - Pr[D(Y_n) = 1]| > \epsilon. \tag{1.8}$$

Inoltre, diremo che D *distingue gli insiemi* $\{X_n\}_{n\in\mathbb{N}}$ e $\{Y_n\}_{n\in\mathbb{N}}$ con probabilità $\mu()$ se per ogni $n \in \mathbb{N}$, D distingue X_n e Y_n con probabilità $\mu(n)$.
Analizziamo ora due importanti proprietà dell'indistinguibilità computazionale.
La prima proprietà dice che se due distribuzioni sono indistinguibili, rimangono indistinguibili anche dopo aver applicato su di loro una PPT.

Lemma 1.1 (Chiusura rispetto ad operazioni efficienti)**.** *Se due insiemi* $\{X_n\}_{n\in\mathbb{N}} \approx \{Y_n\}_{n\in\mathbb{N}}$*, allora per ogni n.u. PPT* M*,* $\{M(X_n)\}_{n\in\mathbb{N}} \approx \{M(Y_n)\}_{n\in\mathbb{N}}$.

Dimostrazione. Supponiamo che esista una n.u. PPT D e una funzione non trascurabile $\mu(n)$ tale che D distingue $\{M(X_n)\}_{n\in\mathbb{N}}$ da $\{M(Y_n)\}_{n\in\mathbb{N}}$ con probabilità $\mu(n)$. Cioè,

$$|Pr[D(M(X_n)) = 1] - Pr[D(M(Y_n)) = 1]| > \mu(n) \tag{1.9}$$

ne segue che, presi x e y, rispettivamente da X_n e Y_n, si ha

$$|Pr[D(M(x)) = 1] - Pr[D(M(y)) = 1| > \mu(n). \tag{1.10}$$

Allora, la n.u. PPT $D'() = D(M())$, distingue $\{X_n\}_{n\in\mathbb{N}}$ da $\{Y_n\}_{n\in\mathbb{N}}$ con probabilità $\mu(n)$, questo contraddice l'ipotesi $\{X_n\}_{n\in\mathbb{N}} \approx \{Y_n\}_{n\in\mathbb{N}}$. □

La seconda proprietà dice che la nozione di indistingubilità computazionale è transitiva.

Lemma 1.2 (Lemma ibrido)**.** *Sia* $X^1, X^2, ..., X^m$ *una sequenza di distribuzioni di probabilità. Supponiamo che la macchina* D *distingua* X^1 *e* X^m *con probabilità* ϵ*. Allora esiste* $i \in [1, ..., m-1]$ *tale che* D *distingue* X^i *e* X^{i+1} *con probabilità* $\frac{\epsilon}{m}$*.*

Dimostrazione. Supponiamo che D distingua X^1, X^m con probabilità ϵ. Cioè,

$$|Pr[D(X^1) = 1] - Pr[D(X^m) = 1]| > \epsilon \tag{1.11}$$

Chiamiamo $g_i = Pr[D(X^i) = 1]$. Allora, $|g_1 - g_m| > \epsilon$. Ne segue che,

$$|g_1-g_2|+|g_2-g_3|+...+|g_{m-1}-g_m| \geq |g_1-g_2+g_2-g_3+...+g_{m-1}-g_m| = |g_1-g_m| > \epsilon. \tag{1.12}$$

Allora, deve esistere i tale che $|g_i - g_{i+1}| > \frac{\epsilon}{m}$. □

1.7 Generatori pseudocasuali

Utilizziamo la nozione, appena data, di indistinguibilità computazionale per definire i *generatori pseudocasuali*.
Prima però ci serve la definizione di *distribuzione pseudocasuale*. Denotiamo con U_n la distribuzione uniforme su $\{0,1\}^n$. Diremo che una distribuzione è pseudocasuale se è indistinguibile dalla distribuzione uniforme.

Definizione 1.15 (Insiemi pseudocasuali)**.** L'insieme di probabilità $\{X_n\}_{n\in\mathbb{N}}$ dove X_n è una distribuzione di probabilità su $\{0,1\}^n$, è detta *pseudocasuale* se $\{X_n\}_{n\in\mathbb{N}} \approx \{U_n\}_{n\in\mathbb{N}}$.

Adesso possiamo dare la definizione di *generatore pseudocasuale*, abbreviato in **PRG** (*pseudo-random generator*).

Definizione 1.16 (Generatore pseudocasuale)**.** Una funzione $G : \{0,1\}^* \to \{0,1\}^*$ è un *generatore pseudocasuale (PRG)* se ha le seguenti proprietà:

1. (efficienza): G può essere eseguito in p.p.t.
2. (espansione): $|G(x)| > |x|$
3. l'insieme $\{G(U_n)\}_{n\in\mathbb{N}}$ è pseudocasuale.

Dove con la notazione $G(U_n)$ intendiamo che, dopo aver scelto x secondo la distribuzione U_n, viene eseguito $G(x)$.

1.8 Sicurezza computazionale

La definizione di indistinguibilità computazionale ci serve, anche, per definire uno *schema crittografico computazionalmente sicuro*. Inoltre, utilizzeremo i **PRG** per dimostrare che un particolare schema crittografico è sicuro secondo la definizione di sicurezza che daremo a breve.
L'idea che sta dietro alla definizione di sicurezza computazione è semplice: richiede solamente che la cifratura di due messaggi sia indistinguibile.

Definizione 1.17. Uno schema crittografico (Gen, Enc, Dec), si dice $single-message\ secure$ se per ogni PPT D, esiste una funzione trascurabile $\epsilon()$ tale che per ogni $n \in \mathbb{N}$ e per ogni coppia di messaggi m_1,$m_2 \in \{0,1\}^n$, D distingue le seguenti distribuzioni con probabilità al più $\epsilon(n)$:

- $\{Enc_k(m_1),\ con\ k\ data\ da\ Gen(1^n)\}$;
- $\{Enc_k(m_2),\ con\ k\ data\ da\ Gen(1^n)\}$.

Questa definizione è basata sull'indistinguibilità delle due distribuzioni di testi cifrati creati cifrando due messaggi differenti.
Adesso daremo un esempio di schema crittografico *single-message secure*. Vedremo che, utilizzando un PRG, si possono cifrare messaggi lunghi con una chiave corta.

Esempio 1.2. Prendiamo un generatore pseudo casuale $G(s)$ e consideriamo il seguente schema crittografico per messaggi di n bit:

- $Gen(1^n) : U_{n/2} \to k$
- Enc_k: Output $m \oplus G(k)$
- Dec_k: Output $c \oplus G(k)$

Teorema 1.3. *Lo schema* (Gen, Enc, Dec) *descritto nell'esempio precedente è single-message secure.*

Dimostrazione. Supponiamo per assurdo che esista un distinguisher D e un polinomio $p(n)$ tale che per infiniti n, esistono messaggi m_n^0, m_n^1 tali che D distingue le seguenti distribuzioni con probabilitá $1/p(n)$:

- $\{k \leftarrow Gen(1^n) : Enc_k(m_n^0)\}$
- $\{k \leftarrow Gen(1^n) : Enc_k(m_n^1)\}$.

Poi consideriamo le seguenti distribuzioni ibride:

- H_n^1 (cifratura di m_n^0): $\{s \leftarrow Gen(1^n) : m_n^0 \oplus G(s)\}$
- H_n^2: $\{r \leftarrow U_n : m_n^0 \oplus r\}$
- H_n^3: $\{r \leftarrow U_n : m_n^1 \oplus r\}$
- H_n^4 (cifratura di m_n^1): $\{s \leftarrow Gen(1^n) : m_n^1 \oplus G(s)\}$.

Per costruzione D distingue H_n^1 e H_n^4 con probabilità $1/p(n)$ per infiniti n. Allora per il *lemma ibrido* D distingue due distribuzioni ibride consecutive con probabilità $\frac{1}{4p(n)}$. Mostriamo che questo è assurdo.
$\{H_n^1\}_{n\in\mathbb{N}} \approx \{H_n^2\}_{n\in\mathbb{N}}$ e $\{H_n^3\}_{n\in\mathbb{N}} \approx \{H_n^4\}_{n\in\mathbb{N}}$ per definizione di PRG, inoltre, per la sicurezza di Shannon $\{H_n^2\}_{n\in\mathbb{N}} \approx \{H_n^3\}_{n\in\mathbb{N}}$.
Quindi, tutte le distribuzione ibride consecutive sono indistinguibili e questo contraddice l'ipotesi. □

Capitolo 2

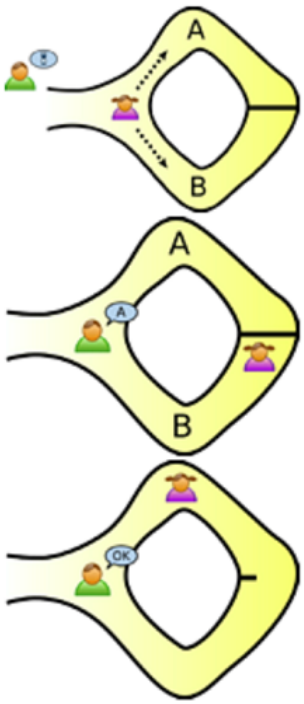
Conoscenza Zero

Come già indicato nell'introduzione, in questo capitolo daremo un'altra definizione di schema crittografico a chiave privata, utilizzando un approccio differente da quello usato nel capitolo precedente, che si basa sul *protocollo* (*o dimostrazione*) *a conoscenza zero*. Inoltre, dimostreremo che questo schema crittografico ha una sicurezza equivalente a quella definita sopra.
Vediamo un esempio che spiega abbastanza bene l'idea che è alla base di questo protocollo:

Esempio 2.1.
Ci sono due parti *Peggy* (che dimostra l'affermazione) e $Victor$ (che verifica l'affermazione).

Peggy ha scoperto la parola segreta usata per aprire la porta magica in una caverna. La caverna ha la forma di un cerchio, con l'entrata su un lato e la porta magica che blocca l'altro lato. Victor dice che la pagherà per il segreto, ma non prima di essere sicuro che lei lo conosca davvero. Peggy si dice d'accordo a rivelargli il segreto, ma non prima di aver ricevuto i soldi. Pianificano allora uno schema con il quale Peggy può dare prova a Victor di conoscere la parola senza rivelargliela.

Victor aspetta fuori dalla caverna mentre Peggy entra. Etichettiamo il sentiero sinistro e quello destro partendo dall'entrata con A e B. Peggy sceglie a caso uno dei due sentieri. Quindi, Victor entra nella caverna e grida il nome del sentiero che Peggy dovrà utilizzare per ritornare indietro, fra A e B, preso a caso. Se si ipotizza che lei conosca veramente la parola magica, è facile: apre la porta, se necessario, e ritorna attraverso il sentiero desiderato. È da notare che Victor non conosce il sentiero per il quale Peggy è



entrata.

Comunque, supponiamo che Peggy non conosca la parola. Allora, sarebbe in grado di tornare attraverso il sentiero chiamato se Victor avesse scelto il nome dello stesso sentiero per cui lei era entrata. Poichè Victor ha scelto a caso tra A e B, avrebbe il 50% di probabilità di indovinare correttamente. Se i due ripetessero questo procedimento molte volte (ad esempio, venti volte una dopo l'altra), la possibilità per Peggy di adempiere in modo corretto, senza conoscere davvero la parola magica, a tutte le richieste di Victor, diventerebbe statisticamente molto piccola.

Perciò, se Peggy appare in modo affidabile all'uscita chiamata da Victor, questo può concludere che molto probabilmente conosca davvero la parola segreta.

Traducendo l'esempio in modo formale, abbiamo un possessore e un verificatore; entrambi devono dimostrare all'altro di essere onesti, il verificatore seguendo un preciso schema di operazioni (un protocollo) e il possessore dando la risposta corretta alle domande del verificatore. Per avere una dimostrazione a conoscenza zero di un'affermazione, occorrerà che siano verificate tre condizioni:

1. **Completezza:** se l'affermazione è vera, un possessore onesto potrà convincere un verificatore onesto.

2. **Correttezza:** se l'affermazione è falsa, nessun possessore disonesto potrà convincere (sufficientemente) il verificatore onesto che essa è vera.

3. **Conoscenza zero:** se l'affermazione è vera, un verificatore potrà ricavare solo tale conoscenza, e nulla più.

Notiamo che questa non è una dimostrazione in senso matematico: non abbiamo mai la certezza, ma solo una probabilità di errore che possiamo rendere piccola a piacere.

La differenza sostanziale che c'è tra l'approccio descritto nel capitolo precedente e quello che andremo a descrivere qui sotto è tra $informazione$ e $conoscenza$. Nell'approccio "classico", ci si preoccupa di non trasferire nessuna informazione ad un eventuale ascoltatore. Mentre, qui non si vuole trasferire conoscenza, neppure al $verificatore$ stesso.

2.1 Cosa significa trasmettere conoscenza?

[1] Per conoscenza intendiamo la capacità di completare un certo compito. In particolare, un messaggio m trasmette conoscenza quando permette al destinatario di completare un nuovo compito, che prima non era in grado di svolgere. Quindi, per quantificare la conoscenza trasmessa da un messaggio, "misuriamo" quanto è più facile svolgere un compito se si è in possesso del messaggio m.

Per illustrare l'idea, consideriamo il semplice caso in cui Alice manda a Bob un unico messaggio. Immaginiamo che Alice mandi lo stesso messaggio 0^n a Bob. Il messaggio di Alice è deterministico e ha una breve descrizione; Bob può facilmente generare il messaggio 0^n da solo. Allora, questo messaggio non trasmette nessuna conoscenza a Bob.

Immaginiamo ora, che f sia una funzione unidirezionale, consideriamo il caso in cui Alice manda a Bob un messaggio della forma "controimmagina della controimmagine ... (n volte) di 0" ($f^{-n}(0)$). Ancora una volta, la stringa inviata da Alice è deterministica e ha una breve descrizione. Tuttavia, in questo caso, non è chiaro come Bob possa generare

[1] Stabilire cosa sia la "conoscenza" è sicuramente un problema molto difficile. Quella che segue è una definizione operativa, adatta al contesto della crittografia.

il messaggio da solo, poiché il costo computazionale sarebbe molto elevato. Perciò il messaggio trasmette effettivamente conoscenza a Bob.

Finora, abbiamo considerato solamente messaggi deterministici, ma in crittografia dobbiamo considerare anche il caso in cui Alice utilizza la casualità per scegliere il suo messaggio. In questo caso il messaggio inviato da Alice dipenderà da una distribuzione di probabilità. Allora, per quantificare la conoscenza trasmessa da un messaggio, possiamo considerare la dimensione (*size*) di una macchina di Turing probabilistica che è in grado di produrre la stessa distribuzione di messaggi prodotta da Alice. Possiamo anche accontentarci di richiedere che la macchina campioni messaggi da una distribuzione computazionalmente indistinguibile da quella dei messaggi prodotti da Alice.

2.2 Conoscenza Zero

Abbiamo visto quando una conversazione trasmette conoscenza, adesso invece ci interessa capire quando una conversazione NON trasmette conoscenza.Intuitivamente, quando partecipare alla conversazione **non** permette a Bob di completare un nuovo compito, che non era in grado di svolgere prima. Dato che in matematica i compiti da svolgere sono, di solito, calcolare funzioni, potremmo dire che Bob non impara a calcolare funzioni del messaggio in chiaro che non era già in grado di calcolare prim.
Una definizione più formale potrebbe essere: "Alice non trasmette conoscenza a Bob se Bob non riesce a distinguere (computazionalmente) una distribuzione di messaggi "simulata" dalla distribuzione di messaggi inviata da Alice."

Riesaminiamo, quindi, la teoria della sicurezza in termini di conoscenza; diciamo che uno schema crittografico è sicuro quando non trasmette conoscenza ad un ascoltatore. In altre parole, diciamo che uno schema crittografico è sicuro se il testo cifrato non consente ad un ascoltatore di calcolare una nuova funzione sul messaggio in chiaro, rispetto a quello che avrebbe potuto calcolare senza il messaggio cifrato.[2]

[2]Questa nozione si ricollega a quella di crittografia funzionale, vedi [*Functional Encyption: A New Vision for Public Key Cryptography*].

Diamo la definzione formale, che utilizza un algoritmo simulatore S che produce una distribuzione indistinguibile da quella dei testi cifrati ottenuti da un qualsiasi messaggio m.

Definizione 2.1. (Schema crittografico a Conoscenza Zero) Uno schema crittografico a chiave privata (Gen, Enc, Dec) è *a conoscenza zero* se esiste un algoritmo simulatore S tale che per ogni n.u. p.p.t. D, esiste una funzione trascurabile $\epsilon()$, tale che per ogni $m \in \{0,1\}^n$ si ha che D distingue le seguenti distribuzioni con probabilità al massimo $\epsilon(n)$:

- $\{k \leftarrow Gen(1^n) : Enc_k(m)\}$;
- $\{S(1^n)\}$.

Adesso dimostreremo, come già accennato, che uno schema crittografico a conoscenza zero è computazionalmente sicuro, e viceversa.

Teorema 2.1. *Sia (Gen, Enc, Dec) uno schema crittografico, con Gen, Enc p.p.t., sia M una p.t.m. (polynomial-time machine) tale che per ogni n, $M(n)$ ci restituisce messaggi in $\{0,1\}^n$. Allora (Gen, Enc, Dec) è sicuro se e solo se è a conoscenza zero.*

Dimostrazione.

Sicurezza implica conoscenza zero . Intuitivamente, se fosse possibile estrarre conoscenza da un messaggio cifrato, allora ci dovrebbe essere un modo per distinguere la cifratura di due diversi messaggi. Più formalmente, supponiamo che (Gen, Enc, Dec) sia sicuro. Consideriamo il seguente simulatore $S(1^n)$:

1. Sceglie, utilizzando M, un messaggio $m \in \{0,1\}^n$
2. Sceglie $k \leftarrow Gen(1^n), c \leftarrow Enc_k(m)$.
3. Restituisce c.

Dobbiamo dimostrare che l'output di S è indistingubile dalle cifrature di un qualsiasi messaggio. Supponiamo per assurdo che esiste un distinguisher D (n.u. p.p.t.) e un polinomio $p()$ tale che per infiniti n_i, esistono alcuni messaggi m_{n_i} tali che D distingue:

- $\{k \leftarrow Gen(1^n) : Enc_k(m_{n_i})\}$
- $\{S(1^n)\}$

con probabilità $1/p(n)$. Da $\{S(1^n)\} = \{k \leftarrow Gen(1^n); m_n \leftarrow M(1^n) : Enc_k(m_n)\}$, segue che esistono messaggi m_n e m_{n_i} tali che le loro cifrature si distinguono con probabilità $1/p(n)$. Questo contraddice l'ipotesi di sicurezza.

Conoscenza zero implica sicurezza . Supponiamo che (Gen, Enc, Dec) sia a conoscenza zero, ma esiste un distinguisher D (p.p.t. non uniforme) e un polinomio $p(n)$, tali che per infiniti n esistono messaggi m_n^1 e m_n^2 tali che D distingue:

- $H_n^1 = \{k \leftarrow Gen(1^n) : Enc_k(m_n^1)\}$
- $H_n^3 = \{k \leftarrow Gen(1^n) : Enc_k(m_n^2)\}$

con probabilità $1/p(n)$. Denotiamo con S il simulatore per lo schema (Gen, Enc, Dec) a conoscenza zero, e definiamo la distribuzione:

- $H_n^2 = \{S(1^n)\}$

Per ipotesi,e per il Lemma ibrido, D distingue H_n^1 e H_n^2 o H_n^2 e H_n^3, con probabilità $1/2p(n)$ per infiniti n. Questo contraddice l'ipotesi iniziale.

□

Capitolo 3

Zerocoin

In questo capitolo daremo un esempio di applicazione di un protocollo a conoscenza zero. Descriveremo un'estensione di Bitcoin, chiamata appunto *Zerocoin*, che rende anonime le transazioni in bitcoin, utilizzando un protocollo a conoscenza zero.

Prima descriveremo brevemente come funziona il sistema Bitcoin, poi vedremo dove Zerocoin utilizza il protocollo a conoscenza zero.

3.1 Bitcoin

Bitcoin è la prima moneta elettronica che ha visto un'adozione diffusa. Il suo funzionamento si basa su una rete di nodi peer-to-peer [1] che distribuisce e memorizza le transazioni, e i client usati per interagire con la rete. Il cuore del sistema Bitcoin è la *blockchain* (catena di blocchi), che funge da bacheca virtuale ed è gestita in modo distribuito dai nodi. La blockchain consiste in una serie di blocchi collegati in una hash-chain, ossia mediante funzioni unidirezionali. Ogni blocco memorizza un insieme di transazioni che vengono raccolte dalla rete Bitcoin. Ogni utente che partecipa alla rete Bitcoin possiede un numero arbitrario di coppie di chiavi crittografiche. Le chiavi pubbliche, o "indirizzi bitcoin", fungono da punti d'invio o ricezione per tutti i pagamenti[2]. Il possesso

[1] Peer-to-peer o rete paritaria, in informatica, è un'espressione che indica un modello di architettura logica di rete informatica in cui i nodi non sono organizzati in modo gerarchico sotto forma di client o server, ma sotto forma di nodi equivalenti o paritari (in inglese peer) che possono cioè fungere sia da client che da server verso gli altri nodi della rete.

[2] Chiaramente Bitcoin utilizza la crittografia a chiave pubblica

di bitcoin implica che un utente può spendere solo i bitcoin associati con uno specifico indirizzo. La corrispondente chiave privata serve ad apporre una firma digitale ad ogni transazione facendo così in modo che sia autorizzato al pagamento solo l'utente proprietario della moneta spesa nella transazione. La rete verifica la firma utilizzando la chiave pubblica.

3.1.1 Transazioni

I bitcoin contengono la chiave pubblica del loro proprietario (cioè l'indirizzo). Quando un utente A trasferisce della moneta all'utente B, rinuncia alla sua proprietà aggiungendo la chiave pubblica di B (il suo indirizzo) sulle monete in oggetto e firmandole con la propria chiave privata. Trasmette poi queste monete in un messaggio, la "transazione", attraverso la rete peer-to-peer. Il resto dei nodi valida le firme crittografiche e l'ammontare delle cifre coinvolte prima di accettarla.

3.1.2 Anonimato

L'anonimato non era negli obiettivi progettuali di Bitcoin. Infatti, il sistema Bitcoin ha importanti limitazioni in materia di privacy. In particolare, poiché il registro delle transazioni è completamente pubblico, la privacy degli utenti è protetta solo attraverso l'uso di pseudonimi (le chiavi pubbliche, o indirizzi). In un esempio, dei ricercatori sono stati in grado di tracciare la spesa di 25.000 bitcoins che erano stati rubati nel 2011. Sebbene il monitoraggio di monete rubate possa sembrare inoffensivo, notiamo che tecniche simili potrebbero essere applicate anche per tracciare transazioni sensibili, violando così la privacy degli utenti[3].

A prescindere dagli obiettivi progettuali di Bitcoin, la maggior parte degli utenti sembra essere disposta a compiere grandi sforzi per mantenere l'anonimato, inclusi rischiare i propri soldi e pagare le spese di transazione. Sul mercato ci sono molti operatori che offrono questo genere di servizi. Tuttavia, questi servizi, hanno una grave limitazione: possono rubare i fondi degli utenti. Proprio per questo motivo, gli utenti richiedono, di solito, l'aiuto di questi operatori solo per brevi periodi, quindi per piccoli volumi di transazioni.

[3]Si pensi, ad esempio, all'uso che potrebbe farne un regime autoritario.

3.2 Zerocoin

Zerocoin non è la prima soluzione proposta per risolvere i problemi di privacy di Bitcoin. Tuttavia, un problema comune a molti protocolli di e-cash è che si basano fondamentalmente su una emittente di fiducia o "banca", che crea "monete" elettroniche. Una soluzione (provata senza successo con Bitcoin) è semplicemente quella di nominare una tale parte. In alternativa, si può distribuire la responsabilità tra un gruppo limitato di nodi. Purtroppo, entrambe queste soluzioni sembrano essere incompatibili con il modello di rete Bitcoin, che è formato da molti nodi non affidabili che entrano ed escono continuamente dalla rete. Inoltre, il problema della scelta dei nodi di fiducia a lungo termine, in particolare per gli apetti legali di Bitcoin, sembra un ostacolo difficile da superare. Zerocoin elimina la necessità di tali emittenti, permettendo ai singoli clienti di generare le proprie monete, a condizione che dispongano di sufficienti bitcoin classici.

3.2.1 Intuizione alla base di Zerocoin

Per comprendere l'intuizione alla base di Zerocoin, consideriamo il seguente esempio. Immaginiamo che tutti gli utenti abbiano accesso ad una bacheca fisica. Un utente Alice, per coniare uno zerocoin di denominazione fissa 1$, prima genera un numero casuale di monete S, quindi mette al sicuro S utilizzando un *commitment scheme* [4]. Ne risulta una moneta, denotata con C, che rivela il valore di S solamente se "aperta" con un numero casuale r. Alice carica C nella bacheca pubblica, insieme ad 1$. Tutti gli utenti accetteranno C a condizione che sia strutturata correttamente e riporti la somma corretta della valuta.

Alice, per riscattare C, controlla la bacheca e ottiene l'insieme delle monete $(C_1, ..., C_N)$ che sono state inviate finora da tutti gli utenti del sistema. Successivamente produce una prova a conoscenza zero π per le due seguenti affermazioni:

1. conosce un $C \in (C_1, ..., C_N)$
2. conosce il numero casuale r.

Allora, Alice, usando un travestimento per nascondere la sua identità, inserisce nella bacheca una transazione contenente (S, π). Gli altri utenti verificano la prova π e control-

[4] Un commitment scheme è un sistema crittografico che consente all'utente di impegnarsi per un certo valore mantenendolo nascosto agli altri, con la possibilità di rivelare il valore impegnato più tardi.

lano se S non è stato utilizzato in nessun'altra transazione. Se soddisfa queste condizioni, gli altri utenti permettono ad Alice di prendere 1$ da ogni C_i presente nella bacheca; altrimenti rifiutano la sua transazione e le impediscono di raccogliere la valuta.

Questo semplice protocollo raggiunge alcuni importanti obiettivi. In primo luogo, la moneta di Alice non può essere collegata ai suoi fondi recuperati: per collegare C al numero di serie S, si deve sapere di quale moneta Alice ha dimostrato la conoscenza, e questo non viene rivelato dalla prova. Così, anche se la moneta in passato è stata usata per transazioni illecite, Alice può utilizzarla. Allo stesso tempo, Alice non può duplicare nessuna moneta senza riutilizzare il numero di serie S e quindi rivelarsi agli altri utenti della rete.

Ovviamente, il protocollo sopra non è praticabile: le bacheche fisiche sono un luogo scadente per memorizzare denaro e informazioni critiche. La valuta potrebbe essere rubata o i numeri di serie rimossi per consentire la doppia spesa. Per condurre questo protocollo su una rete, Alice utilizza una valuta di distribuzione digitale.

Bibliografia

[1] Rafael Pass, Abhi Shelat, *A Course in Cryptography*, disponibile al sito www.cs.cornell.edu/courses/cs4830/2008fa/lecnotes.pdf.

[2] Jonathan Katz, Yehuda Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2007.

[3] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin, *Zerocoin: Anonymous Distributed E-Cash from Bitcoin*, disponibile al sito http://zerocoin.org/media/pdf/ZerocoinOakland.pdf.

[4] Dan Boneh, Amit Sahai, Brent Waters, *Functional Encryption: A New Vision for Pubblic Key Cryptography*, disponibile al sito https://pdfs.semanticscholar.org/e0f3/26795c492246f6c37ab1e8ee22f2482b827a.pdf.

[5] Wikipedia, *Dimostrazione a conoscenza zero*, disponibile al sito https://it.wikipedia.org/wiki/Dimostrazione_a_conoscenza_zero.