Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Matematica

PROTOCOLLI DI AUTENTICAZIONE DIMOSTRABILMENTE SICURI

Tesi di Laurea Magistrale in Crittografia

Relatore:
Chiar.mo Prof.
Davide Aliffi

Presentata da:
Matteo Candita

III Sessione
A.A. 2014/2015

Indice

Prefazione

Parlando di crittografia viene naturale pensare a dei messaggi in codice, trasformati in modo da risultare incomprensibili a chiunque non conosca il codice utilizzato. Il procedimento di cifratura e decifrazione dei messaggi garantisce la *riservatezza* dei messaggi stessi, ma questa non è l'unica proprietà assicurabile tramite strumenti crittografici. Altre proprietà desiderabili possono essere l'*integrità* del messaggio, ovvero la possibilità di verificare che questo non sia stato alterato, e la sua *disponibilità*, ovvero che i contenuti siano sempre accessibili a chiunque abbia diritto di farlo (e solo ad essi).
Oltre a queste vi è poi la possibilità di fornire procedure per l'autenticazione delle parti in comunicazione. Questo aspetto risulta di cruciale importanza in molte applicazioni, come ad esempio nell'e-banking o nell'e-commerce, dove è fondamentale accertarsi dell'identità delle parti.
Nella presente tesi verrà analizzata quest'ultima proprietà, verrà introdotto formalmente il problema dell'autenticazione, dandone una modellazione matematica, e una definizione di protocollo matematicamente sicuro sotto l'assunzione dell'esistenza di una classe di funzioni detta pseudorandom. Verranno poi indagate le funzioni pseudorandom, mostrando quanto questa assunzione teorica sia vicina alla realtà pratica.

Capitolo 1

Il modello di Bellare-Rogaway

1.1 Introduzione

Il contesto è quello classico della crittografia: Alice vuole comunicare con Bob su un canale insicuro in cui può esserci una terza parte (che chiameremo avversario o Eva) che può intercettare, leggere, modificare i messaggi a piacimento.

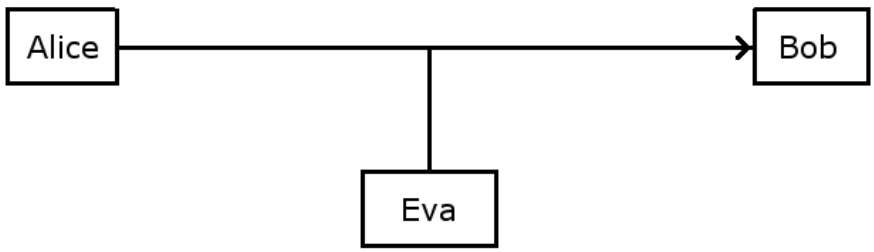


Figura 1.1: Schema di riferimento. Alice e Bob comunicano su un canale insicuro, Eva può intercettare

Come già detto, nel presente lavoro verrà analizzato esclusivamente il problema dell'autenticazione delle parti (*Entity Authentication*)[1] . Questa può essere:

[1]Da non confondersi con l'autenticazione dell'origine dei messaggi (*data origin authentication*). In quest'ultima si autenticano dei singoli messaggi, che possono essere anche datati. Nell'*Entity Authentication* l'autenticazione avviene in tempo reale e garantisce l'autenticazione soltanto della sessione in corso.

- unilaterale. Bob vuole essere sicuro che il suo interlocutore sia effettivamente Alice.
- mutua. Alice e Bob vogliono identificarsi vicendevolmente.

Concettualmente un utente per autenticarsi deve riuscire in qualche prova che possa essere svolta con successo da lui e da lui soltanto. La prova consiste usualmente nel dimostrare una conoscenza (es: password) o un possesso (es: smartcard) o una caratteristica biometrica (es: scan della retina, impronta digitale).
In un contesto di rete la metodologia più diffusa è senz'altro l'utilizzo di una password; questa può essere stata precondivisa fra i partecipanti (caso simmetrico) o essere appannaggio esclusivo della parte che si vuole identificare (caso asimmetrico, firma digitale). Noi ci interesseremo del caso simmetrico.

Per ottenere l'autenticazione su un canale insicuro occorre costruire dei protocolli, ovvero sequenze di messaggi tra A e B da effettuarsi in un ordine e in un formato specifico e stabilito a priori; questo deve essere strutturato in modo tale che, una volta terminato con successo, l'utente possa avere la certezza (a meno di una probabilità trascurabile) dell'identità dell'interlocutore.
Nel passato questa costruzione era spesso fatta per tentativi: veniva proposto uno schema e lo si provava a forzare. Questo modo di procedere non è ottimale per più motivi, il più importante dei quali è teorico: non aver trovato un modo per attaccare lo schema non esclude la possibilità che un tale modo venga trovato in futuro. Per avere questa certezza occorre cambiare approccio e cercare di formalizzare tutto al fine di poter provare matematicamente la sicurezza del protocollo, sotto certe ipotesi.
Partendo da queste osservazioni Mihir Bellare e Phillip Rogaway presentarono nel 1993 nell'articolo *Entity Authentication and Key Distribution* [BeRog] una modellazione matematica del problema piuttosto forte, in cui l'avversario ha il pieno controllo delle comunicazioni, e suggeriscono un protocollo che si dimostra essere sicuro sotto l'assunzione dell'esistenza di funzioni pseudorandom.
Ripercorriamo la loro costruzione del modello e la presentazione del protocollo, verificandone la dimostrazione di sicurezza. Presentiamo poi alcuni attacchi clas-

sici all'autenticazione (principalmente l'attacco del Replay e del Gran Maestro di Scacchi), dimostrando che non sono applicabili a questo protocollo. Investighiamo infine le basi crittografiche su cui il protocollo si poggia (in primis le funzioni pseudorandom) e come queste possano essere realizzate.

1.2 Premesse

Prima di presentare il modello matematico vero e proprio, sono necessarie delle nozioni preliminari e alcune notazioni.

1.2.1 Notazioni

Introduciamo alcune delle notazioni utilizzate nel seguito:

- Indicheremo con $\{0,1\}^*$ l'insieme delle stringhe binarie finite, con $\{0,1\}^\infty$ le stringhe infinite e con $\{0,1\}^{\leq L}$ quelle di lunghezza al più L
- λ rappresenta la stringa vuota.
- Se $a, b, c...$ sono stringhe $a.b.c.$ ... indica una codifica che permetta di risalire alle singole stringhe. Ad esempio si può utilizzare la concatenazione.

Diremo poi che una funzione è efficientemente computabile se esiste un algoritmo che la calcola in tempo polinomiale rispetto alla dimensione dell'input.
Diremo che una funzione $\epsilon(k)$ è *trascurabile* se $\forall c > 0 \quad \exists k_c > 0$ t.c. $\epsilon(k) < k^{-c} \ \forall k > k_c$

1.2.2 Protocolli

Nel nostro modello i protocolli prevedono due interlocutori e sono formalizzati attraverso una funzione efficientemente computabile Π che calcola il messaggio successivo nella sequenza. Nello specifico la funzione è $\Pi(1^k, i, j, a, \kappa, r) = (m, \delta, \alpha)$, più precisamente:
INPUT:

- 1^k è il parametro di sicurezza, $k \in N$
  La notazione 1^k indica una stringa costituita da k volte 1, in questo modo si può trovare il valore k esaminando la lunghezza della stringa in input.

- i è l'identità del mittente, $i \in I \subset \{0,1\}^k$.

- j è l'identità del destinatario del messaggio, $j \in I \subset \{0,1\}^k$.

- a è il segreto del mittente, $a \in \{0,1\}^*$
  Ci si riferisce ad a come LL key (*long lived key*) e nel caso simmetrico questa è la stessa per tutte le parti in gioco. Associato ad ogni protocollo vi è un generatore $\mathcal{G}$ di LL key, il cui funzionamento è specificato nel seguito, che determina chi e come ottiene la chiave a.

- κ è un registro che tiene traccia della conversazione fino all'istante attuale, $\kappa \in \{0,1\}^*$

- r è una sequenza random (potenzialmente infinita) generata dal mittente, $r \in \{0,1\}^\infty$

OUTPUT:

- m è il messaggio successivo nel protocollo, $m \in \{0,1\}^* \cup \{*\}$
  Il valore $*$ indica che l'utente non invia alcun messaggio.

- δ è la decisione dell'utente, $\delta \in \{A, R, *\}$
  I valori indicano che l'utente ha Accettato, Rifiutato o non è ancora giunto a una conclusione. Osserviamo che generalmente la decisione non viene presa se non a protocollo ultimato.

- α è l'output privato, $\alpha \in \{0,1\}^* \cup \{*\}$
  Anche qui il valore $*$ indica che l'utente non ha generato un output privato. Questo in effetti non viene utilizzato ai fini dell'autenticazione, ma può essere necessario, ad esempio, per inviare una chiave di sessione.

LL Key Generator $\mathcal{G}$ Associato ad ogni protocollo c'è un generatore $\mathcal{G}(1^k, i, r_\mathcal{G})$ che, presi in input il parametro di sicurezza k, l'identità del richiedente i e un numero random $r_\mathcal{G}$ fornisce una chiave ad i. Nel nostro protocollo le chiavi sono simmetriche, ovvero $\mathcal{G}(1^k, i, r_\mathcal{G}) = \mathcal{G}(1^k, j, r_\mathcal{G}) \ \ \forall i, j \in I$, mentre $\mathcal{G}(1^k, E, r_\mathcal{G}) = \lambda$ (i.e. E non dispone della chiave).
Per i nostri scopi basta considerare un generatore banale che restituisce semplicemente i primi k valori di $r_\mathcal{G}$.
Il motivo per cui si introduce un generatore associato al protocollo invece che darne uno predefinito (come quello appena suggerito) permette di prescindere dal contesto e garantisce maggiore generalità.

1.2.3 Macchine probabilistiche e Oracoli

Introduciamo adesso dei concetti che saranno utili nella definizione del modello:
Una *macchina probabilistica* è una macchina (ad esempio una macchina di Turing, o un qualche altro modello astratto di calcolatore) il cui comportamento non è univocamente determinato, ma può invece compiere delle scelte casuali con una certa distribuzione di probabilità. È il contrario di una macchina deterministica, in cui ad ogni momento è sempre specificata ed unica l'operazione successiva da compiere.
Una macchina dotata di *oracolo* è un oggetto astratto utile nell'analisi di problemi decisionali e nello studio degli algoritmi. Un oracolo è da pensarsi come una scatola nera che, quando interpellato dalla macchina, riesce istantaneamente a fornire la soluzione a un dato problema. Osserviamo che non ci interessa *come* un oracolo trovi la soluzione, né se la soluzione sia effettivamente computabile: per ogni dato input l'oracolo *in un qualche modo* conosce la risposta e la restituisce immediatamente in output.

1.2.4 Random e Pseudorandom

Diamo ora una introduzione ai concetti di random e pseudorandom.

Stringhe pseudorandom (cfr. [KaLi], cap. 3.3) Cominciamo con l'osservare che concettualmente non ha senso definire random o pseudorandom una singola stringa, ha senso invece dire che la stringa proviene da una distribuzione random o da una distribuzione pseudorandom. Fatta questa osservazione, nel seguito parleremo di stringa random intendendo quanto appena detto.
Cosa si richiede a una stringa binaria perché sia considerata random? Idealmente ogni bit dovrebbe avere probabilità $\frac{1}{2}$ di essere 0 o 1, ovvero essere il risultato di un lancio di una moneta perfetta. Questo si può ottenere da funzioni che generano numeri casuali attingendo da *fonti di casualità* sia a livello hardware (ad esempio fenomeni fisici come il numero di decadimenti radioattivi in un'unità di tempo), che software (ad esempio il movimento del mouse, il numero di accessi all'hard disk, ecc.). Generatori di questo tipo esistono, ma il loro problema principale è il *tempo* impiegato per produrre dei dati. Quello che viene fatto usualmente è prendere una stringa random *corta* ed *estenderla* utilizzando un generatore pseudorandom. Un generatore è pseudorandom se, utilizzando un algoritmo D che richiede tempo polinomiale, una stringa da esso prodotta è indistinguibile da una stringa random. Formalmente:

Definizione 1 (**Generatore pseudorandom**)**.** *Sia n la lunghezza del seme casuale preso in input, sia $l(\cdot)$ un polinomio e G un algoritmo deterministico a tempo polinomiale che per ogni input $s \in \{0,1\}^n$ restituisce una stringa di lunghezza $l(n)$. Diciamo che G è un* generatore pseudorandom *se*

- *Espansione: Per ogni n vale che $l(n) > n$*
- *Pseudorandomness: Per ogni algoritmo probabilistico D con tempo polinomiale ed output in $\{0,1\}$, esiste una funzione trascurabile ϵ tale che*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| \leq \epsilon(n)$$

con r scelta in modo uniformemente casuale in $\{0,1\}^{l(n)}$ e il seme s scelto in modo uniformemente casuale in $\{0,1\}^n$ e G(s) è la stringa pseudorandom

prodotta da G a partire dal seme s.

La funzione $l(\cdot)$ è detta fattore di espansione di G

Il primo punto garantisce che la funzione espanda il seme random e il secondo garantisce che qualunque test probabilistico polinomiale D assegni il valore 1 con probabilità quasi uguale sia a un input random che a uno pseudorandom, ovvero **è trascurabile la probabilità che l'algoritmo D abbia successo nel distinguere i due casi**.

Funzioni pseudorandom (cfr [KaLi], cap 3.6.1) Come per le stringhe, osserviamo anche qui che è privo di senso dire che una singola funzione $f : \{0,1\}^n \to \{0,1\}^n$ sia pseudorandom; al contrario si può supporre che questa provenga da una distribuzione di funzioni pseudorandom.
Per fare ciò consideriamo una famiglia di funzioni $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, dove la prima variabile è un parametro che denoteremo con k e la seconda è detta input della funzione. Una volta scelto e fissato k si avrà la k-sima funzione F_k della famiglia F, definita come $F_k(x) = F(k,x)$. Su una famiglia di funzioni è naturalmente indotta una distribuzione data dalla distribuzione casuale uniforme sulle chiavi k.
Intuitivamente diremo che F è una famiglia di funzioni pseudorandom se è efficientemente calcolabile e F_k (ottenuta con una scelta uniformemente casuale di k) è indistinguibile da una funzione scelta in maniera uniformemente casuale tra tutte le funzioni con lo stesso dominio e codominio. Le funzioni da $\{0,1\}^n$ a $\{0,1\}^n$ (nel seguito $func_n$) si possono pensare come i modi di assegnare un valore (una stringa lunga n) ad ogni elemento del dominio (gli elementi sono 2^n); la cardinalità dello spazio delle funzioni è quindi $2^{n \cdot 2^n}$. Osserviamo invece che il numero di chiavi tra cui scegliere k è 2^n, numero di molto inferiore, e nonostante ciò in una funzione pseudorandom la scelta sui due spazi deve essere indistinguibile con risorse polinomiali.
Puntando ad una formalizzazione si potrebbe pensare di seguire lo stesso ragionamento utilizzato in def. 1: richiedere che ogni algoritmo D probabilistico che

riceva una descrizione della funzione F_k dia in output 1 con la stessa probabilità (a meno di una quantità trascurabile) che se ricevesse una descrizione di una funzione random f_n.
Ma una funzione random è costruita assegnando ad ogni elemento del dominio valori scelti in maniera uniformemente casuale e l'unico modo possibile di descrivere una tale funzione è fornire l'intero elenco dei valori. Questo è lungo $n \cdot 2^n$ e cresce quindi in maniera esponenziale con la dimensione dello spazio; d'altra parte il nostro algoritmo D dispone solo di risorse polinomiali e perciò non riuscirebbe neanche ad esaminare l'intero input.
Per la definizione formale bisogna quindi dotare D di un oracolo cui chiedere per ogni dato x l'output delle funzioni F_k e f_n che gli viene così fornito immediatamente. Indicheremo con $D^{F_k(\cdot)}$ e $D^{f_n(\cdot)}$ gli algoritmi D provvisti dei due oracoli.

Definizione 2 (Famiglia di funzioni pseudorandom)**.** *Sia* $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ *una famiglia di funzioni efficientemente calcolabili. Diremo che* F *è una* famiglia di funzioni pseudorandom *se per ogni algoritmo probabilistico* D *a tempo polinomiale, esiste una funzione trascurabile* ϵ *tale che:*

$$|Pr[D^{F_k(\cdot)}(1^n) = 1] - Pr[D^{f_n(\cdot)}(1^n) = 1]| \leq \epsilon(n)$$

Dove k *è scelto in maniera uniformemente casuale sullo spazio delle chiavi e* f_n *in maniera uniformemente casuale sullo spazio delle funzioni.*

Osservazione. È importante osservare che *non è stato dimostrato* che né generatori né funzioni pseudorandom esistano effettivamente. D'altra parte esistono delle funzioni che si ritengono essere pseudorandom ed esiste una costruzione teorica a partire da funzioni unidirezionali (i.e. funzioni difficili da invertire). Si rimanda la discussione al capitolo 2.1

1.3 Il modello di Bellare-Rogaway

Possiamo ora finalmente esporre il modello di Bellare-Rogaway. Nello schema le parti in gioco possono essere due o più (ma a priori potrebbe anche essere una sola). Le identificheremo con i, j (o anche A,B per Alice e Bob) e indicheremo con I l'insieme delle identità. Le parti in gioco condividono un segreto a, noto esclusivamente a loro. Si assume poi la presenza di un avversario E e che tutte le comunicazioni tra le parti passino per E. Ciò implica che, nel nostro modello, E ha il potere di leggere, modificare, ritardare e reinviare messaggi, ma anche crearne di nuovi e iniziare a suo piacimento delle sessioni del protocollo. L'unica cosa che E non può fare è leggere la chiave. Sottolineiamo che le parti in gioco comunicano *esclusivamente* con E, non potendo perciò comunicare tra loro. È evidente dalla formulazione che un protocollo che garantisca la sicurezza in un contesto in cui l'avversario ha pieno controllo dia automaticamente sicurezza anche in un contesto reale.
Formalmente, l'avversario sarà una macchina probabilistica $E(1^k, a_E, r_E)$ dotata di infiniti oracoli $\Pi^s_{i,j}$, con $i, j \in I$ e $s \in N$. Gli oracoli $\Pi^s_{i,j}$ modellano l'utente i che cerca di identificare j nella sessione s.
E comunica con gli oracoli tramite richieste del tipo (i, j, s, x) (E manda un messaggio x a i nella sessione s, sostenendo che provenga da j). Alle richieste di questo tipo l'oracolo risponde effettuando le seguenti operazioni:

1. Si sceglie una stringa random $r_{\mathcal{G}} \in \{0,1\}^\infty$ e si stabiliscono i segreti $a_i = \mathcal{G}(1^k, i, r_{\mathcal{G}})$, con $i \in I$, e $a_E = \mathcal{G}(1^k, E, r_{\mathcal{G}})$

2. Si sceglie una stringa random $r_E \in \{0,1\}^\infty$ e per ogni i, j, s una stringa random $r^s_{i,j} \in \{0,1\}^\infty$

3. Si inizializzano le $\kappa^s_{i,j} = \lambda$. Queste terranno traccia delle conversazioni di E con $\Pi^s_{i,j}$.

4. Quando E fa una richiesta del tipo (i, j, s, x), l'oracolo $\Pi^s_{i,j}$ calcola $(m, \delta, \alpha) =$

$\Pi(1^k, i, j, a_i, \kappa^s_{i,j}.x, r^s_{i,j})$ e risponde con (m, δ)[2]. A quel punto viene aggiornato $\kappa^s_{i,j} = \kappa^s_{i,j}.x$

Finora non abbiamo ancora specificato nulla riguardo identità e autenticazione; questo appena descritto è un modello generale che può essere preso come riferimento per molte analisi su problemi diversi come lo scambio di chiavi.

Passiamo ora a definire cosa vuol dire autenticare e a fornire un protocollo di autenticazione *sicuro*.

1.3.1 Conversazioni corrispondenti

Nel nostro ambiente riveste una particolare importanza il concetto di *conversazioni corrispondenti* (*Matching Conversations*): in sostanza due conversazioni corrispondono (sono *matching*) se sono le due parti di uno stesso scambio di messaggi.
Prima di dare una formalizzazione, osserviamo che in uno scambio di messaggi ha importanza tener conto del

Tempo Diciamo che l'i-sima richiesta dell'avversario avviene al tempo $\tau = \tau_i \in \mathbb{R}$, con l'unica restrizione che $\tau_i < \tau_j$ se $i < j$ (i.e. messaggi successivi sono stati prodotti in tempi successivi[3]). La nozione di tempo può essere specificata in vario modo: come tempo astratto $\tau_i = i$, oppure se le parti sono macchine di Turing si può definire τ_i =i-simo passo nella computazione di E, oppure come tempo *reale*, dove τ_i è l'ora in cui è stato mandato l'i-simo messaggio.

Conversazione Fissata un'esecuzione $\mathcal{E}$, possiamo registrare la conversazione di ogni oracolo $\Pi^s_{i,j}$, come

$$K = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), ..., (\tau_m, \alpha_m, \beta_m).$$

[2]Notare che nella risposta non compare α, la chiave di sessione segreta generata da $\Pi^s_{i,j}$. Questa *non* viene infatti mai comunicata ad E.

[3]Osserviamo che non si fa riferimento ai tempi in cui i messaggi arrivano, ma a quelli in cui vengono prodotti; in condizioni reali spesso l'ordine in cui i messaggi arrivano non è garantito.

Dove la sequenza indica che al tempo τ_i l'oracolo ha ricevuto il messaggio α_i, rispondendo con β_i. Se il primo messaggio ricevuto è una stringa vuota ($\alpha_1 = \lambda$) diremo che l'oracolo $\Pi^s_{i,j}$ ha dato il via alla comunicazione, altrimenti diremo che è un oracolo di risposta.

Conversazioni corrispondenti Fissiamo un numero dispari di mosse $R = 2\rho - 1$ (nel caso pari si definisce analogamente) e consideriamo un protocollo Π ad R mosse eseguito in presenza di un avversario E tra gli oracoli $\Pi^s_{A,B}$ e $\Pi^s_{B,A}$, che registrano rispettivamente le conversazioni K e K'.

- Diremo che K **corrisponde** (è *matching*) a K' se esistono $\tau_0 < \tau_1 < ... < \tau_R$ e $\alpha_1, \beta_1, ..., \alpha_\rho, \beta_\rho$ tali che K sia della forma:

  $$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), ..., (\tau_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1}), (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

  e K' della forma:

  $$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), ..., (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1})$$

- Diremo che K' **corrisponde** a K se esistono $\tau_0 < \tau_1 < ... < \tau_R$ e $\alpha_1, \beta_1, ..., \alpha_\rho, \beta_\rho$ tali che K' sia della forma:

  $$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), ..., (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}), (\tau_{2\rho-1}, \alpha_\rho, *)$$

  e K della forma:

  $$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), ..., (\tau_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1}), (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

Nella definizione $\Pi^s_{A,B}$ è l'oracolo che ha iniziato la comunicazione e $\Pi^s_{B,A}$ è un oracolo di risposta. In sostanza una conversazione K corrisponde a una K' se ogni messaggio ricevuto in K è stato generato in un tempo precedente in K'; viceversa perché K' corrisponda a K ogni messaggio ricevuto in K' deve essere

stato generato in un tempo precedente in K.

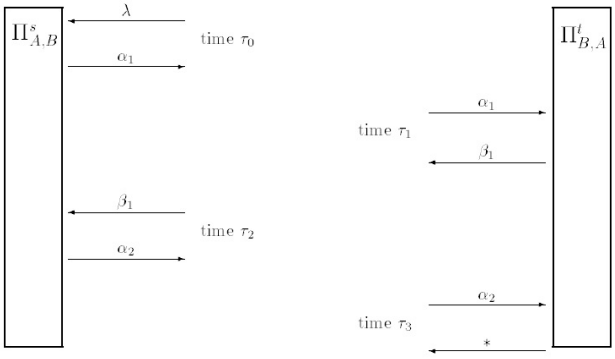


Figura 1.2: Esempio di due conversazioni corrispondenti con $R = 3$ mosse

Possiamo adesso dare la definizione di mutua autenticazione.

1.3.2 Mutua autenticazione

Dato un protocollo diremo che questo fornisce mutua autenticazione se alla fine del protocollo entrambe le parti possono essere sicure dell'identità dell'interlocutore. Un protocollo di questo tipo deve avere $R \geq 3$, almeno 3 messaggi.
Alla fine del protocollo una parte che ha avuto una conversazione K accetta se ritiene che ci sia un'altra parte con una conversazione corrispondente K' (questo in assenza di avversari deve avvenire automaticamente).
Definiamo $\mathbf{No-Matching}^E(k)$ l'evento in cui un oracolo $\Pi^s_{i,j}$ accetta senza che vi sia un oracolo $\Pi^s_{j,i}$ con una conversazione corrispondente. Definiamo:

Definizione 3 (**Mutua autenticazione sicura**)**.** *Diremo che* Π *è un protocollo di mutua autenticazione sicura se per ogni avversario* E *dotato di risorse polinomiali:*

1. *Se gli oracoli* $\Pi^s_{i,j}$ *e* $\Pi^s_{j,i}$ *hanno conversazioni corrispondenti accettano entrambi.*

2. *La probabilità di* $\mathbf{No-Matching}^E(k)$ *è trascurabile.*

La prima condizione specifica che se c'è stato uno scambio di messaggi *corretto* entrambi accettano l'autenticazione del partner; la seconda specifica che la probabilità di accettare erroneamente uno scambio di messaggi come buono è trascurabile per tempi polinomiali.

1.3.3 Il protocollo MAP1

Daremo ora la descrizione di un protocollo che garantisce la mutua autenticazione sicura nel senso appena descritto sotto l'ipotesi dell'esistenza di una famiglia di funzioni pseudorandom.
Data f una famiglia di funzioni pseudorandom, consideriamo $f_a : \{0,1\}^{\leq L(k)} \rightarrow \{0,1\}^k$ la funzione specificata dalla scelta di una chiave a. $L(k), l(k)$ e la lunghezza di a sono funzione del parametro di sicurezza k, nel protocollo MAP1 possiamo scegliere una chiave lunga k, $l(k) = k$ e $L(k) = 4k$.
Indicheremo con $[x]_a = (x, f_a(x))$ con $f_a(x)$ che funzionerà da codice di autenticazione del messaggio x.
$[i.x]_a$ con $i \in I$ sarà l'autenticazione da parte di i del messaggio x

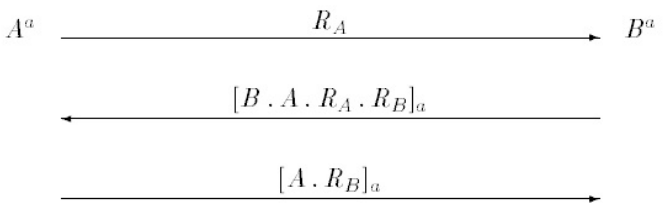


Figura 1.3: Protocollo MAP1 fra A e B con segreto condiviso a

Nel protocollo MAP1 (Mutual Authentication Protocol 1), rappresentato in Figura 1.3, Alice (A) manda a Bob (B) una sfida random R_A di lunghezza k. Bob risponde generando una sfida R_B e mandando $[B.A.R_A.R_B]_a$; Alice riceve il messaggio e controlla che sia effettivamente una risposta alla sua sfida (ovvero che contenga il suo R_A e che sia autentico) e in tal caso accetta e risponde mandando $[A.R_B]_a$. Bob riceve questo messaggio e controlla che ci sia il suo R_B e in tal caso accetta.

Osserviamo che essendo a segreto e condiviso solo tra A e B, E non può direttamente generare alcuno di quei messaggi autenticati.
Proviamo quindi il seguente:

Teorema 1. *Data f famiglia di funzioni pseudorandom, il protocollo MAP1 basato su f fornisce mutua autenticazione sicura (cfr. def 3).*

Dimostrazione. Ricordiamo la definizione. Π è un protocollo di mutua autenticazione sicura se:

1. Se gli oracoli $\Pi^s_{i,j}$ e $\Pi^s_{j,i}$ hanno conversazioni corrispondenti accettano entrambi.

2. La probabilità di $\mathbf{No-Matching}^E(k)$ è trascurabile.

È facile dimostrare la prima condizione della definizione: se un avversario E si limita a inoltrare i messaggi tra A e B, agendo semplicemente da tramite tra i due (*Avversario benigno*), le due parti accettano.
Vogliamo ora dimostrare che inoltrare messaggi legittimi è per E l'unico modo perché il protocollo abbia successo, ovvero che la probabilità che A accetti erroneamente un messaggio *'falso'* come proveniente da B (o viceversa) è trascurabile. Per farlo realizziamo degli esperimenti vicini all'esperimento reale in cui E utilizza MAP1.

Esperimento MAP1 con oracolo g Denotiamo con MAP1^g l'esperimento in cui, invece che un segreto a e una funzione pseudorandom f_a, le parti condividono un oracolo g, non accessibile da E, e quando debbano autenticare un messaggio utilizzino $[x]_g$ in vece di $[x]_a$. Osserviamo che se $g = f_a$ l'esperimento MAP1^g coincide con MAP1.
Interessante ai nostri fini è il caso in cui g sia una funzione random.

Esperimento random MAP1 nel random MAP1 g è una funzione random da $\{0,1\}^{\leq L(k)}$ a $\{0,1\}^k$. Consideriamo $T_E(k)$ il numero (polinomiale) di chiamate agli oracoli $\Pi^s_{A,B}$ realizzabili da E.

Diremo che l'avversario E *ha successo* se convince una parte ad accettare senza che ci sia una conversazione corrispondente, ovvero l'evento $\mathbf{No-Matching}^E(k)$.

Lemma 1. *Dimostriamo che la probabilità che l'avversario E abbia successo in un random MAP1 è al più $T_E(k)^2 \cdot 2^{-k}$*

Dimostrazione. Dividiamo la dimostrazione in due parti: che ad accettare sia l'oracolo che inizia il protocollo o che sia l'oracolo che risponde.

1. Fissati A, B, s, la probabilità che $\Pi^s_{A,B}$, oracolo che inizia il protocollo, accetti senza una conversazione corrispondente è al più $T_E(k) \cdot 2^{-k}$
   Infatti: A ha iniziato il protocollo, quindi a un certo momento τ_0 ha inviato R_A. Se A accetta vuol dire che a un certo momento τ_2 ha ricevuto $[B.A.R_A.R_B]_g$ con un qualche R_B. Le possibilità qui sono due:

   - E ha calcolato questo messaggio con successo, la probabilità che riesca a farlo è di 2^{-k}.
   - E aveva ricevuto questo messaggio in precedenza da un oracolo. Per come è formulato, questo deve venire da un oracolo $\Pi^t_{B,A}$ che abbia ricevuto R_A come primo input in un tempo $\tau < \tau_0$, perché altrimenti si avrebbe una conversazione corrispondente.
     Se E aveva interrogato $T_E(k)-1$ volte l'oracolo con input random $R'_A \in \{0,1\}^k$, la possibilità che uno di questi fosse proprio R_A è $[T_E(k)-1]2^{-k}$

   Ne concludiamo che la probabilità che A accetti erroneamente dopo aver iniziato il protocollo è $T_E(k) \cdot 2^{-k}$

2. Fissati A, B, s, la probabilità che $\Pi^s_{B,A}$, oracolo che risponde, accetti senza una conversazione corrispondente è al più $T_E(k) \cdot 2^{-k}$.
   Supponiamo $\Pi^s_{B,A}$ abbia ricevuto a un tempo τ_0 un R_A, abbia calcolato e rimandato $[B.A.R_A.R_B]_g$ in un tempo $\tau_1 > \tau_0$. Se $\Pi^s_{B,A}$ accetta vuol dire che ha ricevuto in un tempo $\tau_3 > \tau_1$ un messaggio della forma $[A.R_B]_g$. Anche qui i casi son due

- E ha calcolato direttamente il messaggio con probabilità 2^{-k}.
- E ha ricevuto il messaggio in precedenza da un oracolo $\Pi^u_{A,C}$. Una conversazione di $\Pi^s_{A,C}$ ha la forma

$$(\tau_0, \lambda, R'_A), (\tau_2, [C.A.R'_A.R'_B]_g, [A.R'_B]_g)$$

Per ognuna di queste interazioni c'è, a meno di una probabilità 2^{-k}, un oracolo $\Pi^u_{C,A}$ che in qualche istante ha computato quel $[C.A.R'_A.R'_B]_g$. Ora, se $(u, C) \neq (s, B)$ allora la probabilità che $R'_B = R_B$ è al più $[T_E(k) - 2] \cdot 2^{-k}$. D'altra parte, se $(u, C) = (t, B)$ allora $\tau_0 < \tau_1 < \tau_2 < \tau_3$, $R'_A = R_A$ e $R'_B = R_B$ e quindi si avrebbe una conversazione corrispondente.

Si trova quindi che la probabilità che $\Pi^s_{B,A}$, oracolo che risponde, accetti senza una conversazione corrispondente è al più $T_E(k) \cdot 2^{-k}$.

In definitiva, poiché l'avversario può provare al più $T_E(k)$ volte, la probabilità che un oracolo accetti erroneamente è al più $T_E(k)$ volte il valore $T_E(k) \cdot 2^{-k}$ e quindi $T_E(k)^2 \cdot 2^{-k}$. □

Adesso procediamo a dimostrare il Teorema utilizzando un ragionamento per assurdo. Supponiamo quindi che la probabilità di $\mathbf{No-Matching}^E(k)$ nell'esperimento reale MAP1 non sia trascurabile e cioè che

$$\exists c > 0 \text{ t.c. } \forall k_C \ \ \exists k > k_C \ \ P(\mathbf{No-Matching}^E(k)) > k^{-c}$$

Con queste premesse mostriamo che è possibile costruire un test T che in tempo polinomiale distingua la funzione pseudorandom da una funzione puramente random, andando contro l'ipotesi di funzione pseudorandom.
T dispone di un oracolo g che in base al lancio di una moneta, a lui ignoto, può essere una funzione random o una f_a pseudorandom. Scopo di T è riuscire a determinare con probabilità maggiore di $\frac{1}{2}$ se g è random o pseudorandom. Per farlo

T realizza un esperimento MAP1^g con E, rispondendo al posto degli oracoli $\Pi^s_{i,j}$ utilizzando g. Ora, dal lemma sappiamo che E con funzione random ha probabilità di successo minore di $T_E(k)^2 \cdot 2^{-k}$, mentre nel caso reale ha, per ipotesi, probabilità maggiore di k^{-c}. D'altra parte T sa se E ha avuto successo o meno nell'esperimento. T può quindi ipotizzare che g sia pseudorandom in caso di successo di E e che sia random altrimenti, avendo ottenuto un vantaggio di k^{-d} per qualche d contro l'ipotesi di funzione pseudorandom (cfr. definizione 2). □

1.4 Resistenza ad attacchi classici

Il protocollo MAP1 proposto rientra in una più grande famiglia costituita dagli schemi di sfida-risposta (*challenge-response*), che è tra i metodi d'autenticazione più utilizzati. In uno schema di questo tipo quando l'utente A vuole autenticarsi presso B riceve da quest'ultimo una sfida a cui deve saper rispondere. Un'implementazione banale potrebbe essere la richiesta della password: B richiede la password, A deve inviarla. Ovviamente questo metodo soffre la mancanza di variabilità: se un avversario E intercetta la password una volta può facilmente riutilizzarla in seguito (attacco di *replay*). Per ovviare a ciò, quello che viene fatto usualmente è generare una sfida random R_B inviandola all'interlocutore A che deve saperla trasformare in un modo precedentemente concordato. Nel nostro caso la trasformazione considerata consiste in una funzione pseudorandom selezionata da una famiglia di funzioni tramite la chiave segreta condivisa a.
In questa sezione valutiamo la resistenza del protocollo MAP1 ad alcuni attacchi classici.

Indeboliamo il protocollo per mostrare come, modificando alcuni messaggi, esso diventi vulnerabile.

1.4.1 Reflection

Osserviamo cosa accadrebbe se all'interno del protocollo non venisse specificato alcun identificativo (vedi figura 1.4).

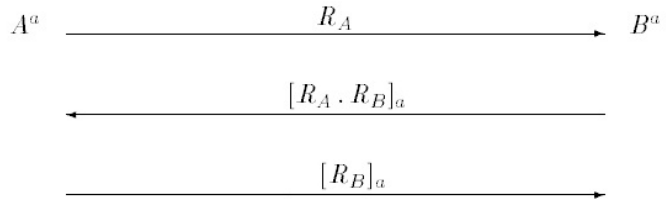


Figura 1.4: Protocollo MAP1 senza identificativi

Si osserva immediatamente che il protocollo diventa simmetrico tra le due parti ed è quindi suscettibile ad un attacco di tipo *reflection* (vedi figura 1.5):
E vuole identificarsi come B presso A; nel primo messaggio riceve R_A da A e invece di inoltrarlo a B, E *riflette* il protocollo e inizia una seconda sessione del protocollo con A, inviandogli lo stesso R_A; A risponderà con un $[R_A.R_{A^2}]_a$ che E restituirà ad A come secondo messaggio nella prima sessione. A controlla il formato del messaggio e trovandolo consistente con il protocollo accetta l'autenticazione.

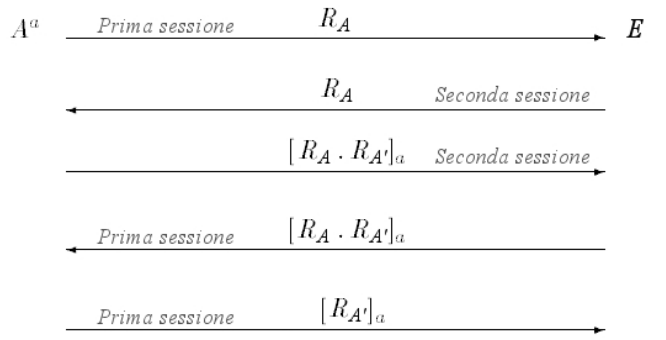


Figura 1.5: Attacco di tipo *Reflection*

Al fine di impedire questo tipo di attacco bisogna eliminare la simmetria negli scambi, introducendo l'utilizzo di identificatori (nell'ordine corretto).[4]

[4]Un'altra soluzione potrebbe essere richiedere che A tenga traccia dei numeri random inviati in

1.4.2 Interleaving

Consideriamo ora il seguente schema (vedi figura 1.6)

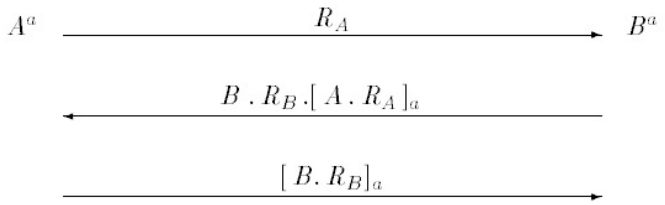


Figura 1.6: Protocollo MAP1 senza concatenazione dei messaggi

In questo protocollo, a differenza del MAP1, i messaggi non sono strettamente concatenati (la risposta di B è solo parzialmente autenticata) e questo crea una vulnerabilità ad attacchi del tipo *interleaving*. In un attacco di tipo *interleaving* l'avversario si pone nel mezzo della comunicazione, chiedendo informazioni ad una delle parti per usarla con l'altra.

In figura 1.7 vediamo come è possibile per E farsi identificare da B come A:

- E manda a B un R_E random.
- B risponde con un $B.R_B.[A.R_E]_a$
- E ha bisogno di $[B.R_B]_a$ e può ottenerlo facilmente iniziando una sessione con A spacciandosi per B e inviando come primo messaggio R_B.
  A risponderà a questo messaggio con un $A.R_A.[B.R_B]_a$, dando ad E l'informazione necessaria per concludere il protocollo.

Al fine di impedire questo tipo di attacco bisogna concatenare i messaggi. Il problema di questo protocollo è infatti che i messaggi non sono concatenati a prova di integrità.

sessioni passate in modo da accorgersi di attacchi di questo tipo. Per vari ragioni pratiche, come la richiesta di memoria potenzialmente infinita, questa soluzione non viene attuata, rendendo ogni sessione del protocollo indipendente (stateless)

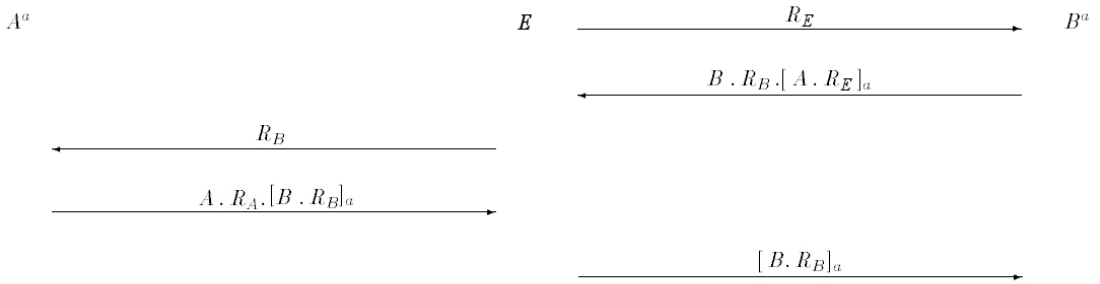


Figura 1.7: Attacco di *interleaving*

Osservazione. L'inserimento dell'identificativo del destinatario all'interno del messaggio rende impossibile l'attacco di *reflection*

Osservazione. É facile vedere come non sia possibile muovere questo tipo di attacchi al protocollo MAP1.

Capitolo 2

Le funzioni pseudorandom

Il protocollo MAP1 risulta sicuro matematicamente e resistente ai principali attacchi, sotto l'assunzione che esistano delle funzioni pseudorandom, ma come detto, non è dimostrato che tali funzioni effettivamente esistano.
Dal punto di vista teorico, delle funzioni pseudorandom si possono ricavare a partire da una classe di funzioni dette *unidirezionali*, ovvero difficili da invertire.
All'atto pratico esistono invece delle funzioni che si ritiene siano pseudorandom, le più utilizzate delle quali rientrano nello schema dei cifrari a blocchi.

Vediamo quindi nel dettaglio entrambi questi approcci

2.1 Costruzione teorica

A partire da una classe di funzioni unidirezionali si definiscono i predicati hard-core e, sfruttando questi, si costruiscono dapprima dei generatori di numeri pseudocasuali e infine delle funzioni pseudocasuali. (cfr [KaLi],cap.6)

2.1.1 Funzioni unidirezionali e predicati hard-core

Cominciamo innanzitutto col definire formalmente una funzione unidirezionale.

Definizione 4 (**Funzione unidirezionale**). *Una funzione* $f : \{0,1\}^* \to \{0,1\}^*$ *è detta* unidirezionale *se:*

- Facilità di calcolo. *Esiste un algoritmo M_f che per ogni input $x \in \{0,1\}^*$ calcoli $f(x)$ in tempo polinomiale*
- Difficoltà di calcolo dell'inversa. *Per ogni algoritmo probabilistico $\mathcal{A}$ che inverta f esiste una funzione trascurabile ϵ tale che*

$$Pr[\mathcal{A}(f(x)) \in f^{-1}(f(x))] \leq \epsilon(n)$$

con x scelto in maniera uniformemente casuale in $\{0,1\}^n$

Osserviamo come la probabilità nel punto 2 è assunta su input uniformemente casuali, non è perciò detto che la funzione sia difficile da invertire per *ogni* input; sottolineiamo inoltre che la definizione non implica che, dato un $y = f(x)$, la ricerca della x sia *impossibile*, anzi, essendo gli spazi finiti, la ricerca esaustiva porta sicuramente ad un risultato. Il metodo forza bruta appena descritto richiede però tempo esponenziale in riferimento alla lunghezza dell'input. Si tratta quindi di una *impossibilità computazionale*, ovvero di una probabilità trascurabile di trovare l'inversa avendo a disposizione solamente risorse polinomiali.

Tra le funzioni unidirezionali identifichiamo quelle con la proprietà di essere biezioni e le chiamiamo *permutazioni unidirezionali.* Essendo il dominio delle funzioni infinito, definiamo una biezione chiedendo che le restrizioni a input lunghi n siano biezioni per ogni n. Formalmente:

Definizione 5 (**Permutazioni unidirezionali**)**.** *Sia f una funzione con dominio $\{0,1\}^*$ e definiamo la funzione f_n come la restrizione di f al dominio $\{0,1\}^n$. Allora una funzione unidirezionale f è detta* permutazione unidirezionale *se per ogni n la funzione f_n è iniettiva e suriettiva su $\{0,1\}^n$.*

Una proprietà interessante delle permutazioni è che, essendo biezioni, ogni $y = f(x)$ determina *univocamente* una preimmagine x. D'altra parte, essendo unidirezionali, è computazionalmente impossibile risalire a x. In questo senso la funzione f nasconde delle informazioni sul suo input x che sembra rimanere del tutto sconosciuto.

Questo non è però sempre vero: una funzione f che fornisce informazioni parziali sul suo input potrebbe comunque essere difficile da invertire. Ad esempio possiamo costruirne una in questo modo: si consideri f unidirezionale, si definisce $g(x_1, x_2) = (f(x_1), x_2)$ con $|x_1| = |x_2|$. g è ancora unidirezionale, ma è evidente che fornisce informazioni sull'input. Abbiamo perciò bisogno di classificare con maggior precisione cosa viene effettivamente nascosto da una funzione unidirezionale e definiamo a questo scopo i *predicati hard-core*. Un predicato hard-core **hc** è una funzione con output un singolo bit che goda della seguente proprietà: se f è unidirezionale, dato l'input $f(x)$ non è possibile calcolare $\mathbf{hc}(x)$ con una probabilità non trascurabilmente maggiore di $\frac{1}{2}$. Ovvero, pur dato $f(x)$ non si ha una maggiore probabilità di indovinare $\mathbf{hc}(x)$ rispetto a provare casualmente. Formalmente:

Definizione 6 (**Predicati hard-core**)**.** *Una funzione a tempo polinomiale* $\boldsymbol{hc}$*:*$\{0,1\}^* \to \{0,1\}$ *è detta* predicato hard-core *di una funzione f se, per ogni algoritmo probabilistico $\mathcal{A}$ a tempo polinomiale, esiste una funzione trascurabile ϵ tale che*

$$Pr[\mathcal{A}(f(x)) = \mathbf{hc}(x)] \leq \frac{1}{2} + \epsilon(n)$$

con x scelto in maniera uniformemente casuale in $\{0,1\}^n$

Così definita, non è immediato capire se questi predicati esistano sempre o solo per determinate funzioni. Nel 1989 Goldreich e Levin (cfr. [GoLe]) dimostrarono che ogni f unidirezionale è modificabile in una funzione g, ancora unidirezionale, per cui esiste un predicato hard-core. Più precisamente:

Teorema 2 (Goldreich e Levin)**.** *Sia f una funzione unidirezionale. Consideriamo la funzione $g(x, r) = (f(x), r)$, con $|x| = |r|$. Allora la funzione* $\mathbf{gl}(x, r) = \bigoplus_{i=1}^{n} x_i \cdot r_i$*, dove $x = x_1, ..., x_n$ e $r = r_1, ..., r_n$, è un predicato hardcore di g* [1]*.*

Dimostrazione. Vedi [KaLi], cap. 6.3 □

Osserviamo che la funzione $\mathbf{gl}(x, r)$ (gl da Goldreich Levin) restituisce in output un bit che è la disgiunzione esclusiva (XOR, ovvero somma modulo 2) di un

[1] Somme e prodotti di bit chiaramente sono in $\mathbf{Z}_2$

sottoinsieme random di bit di x, quello selezionato dagli $r_i = 1$.
Per convenienza di notazione nel prosieguo, quando indicata una f unidirezionale, intenderemo la g modificata perché disponga di un predicato hard-core.

2.1.2 Generatori e funzioni pseudorandom

Descriviamo i passaggi che portano a definire dapprima dei generatori pseudorandom e, utilizzando questi, una famiglia di funzioni pseudorandom. Per le dimostrazioni fare riferimento a [KaLi].

Nella nostra costruzione adopereremo le permutazioni pseudorandom per le quali valgono due importanti risultati:

- se f è una permutazione pseudorandom e x è uniformemente distribuita, allora anche $f(x)$ è uniformemente distribuita.
- sia **hc** un predicato hardcore di f, allora il bit $\mathbf{hc}(x)$ appare pseudorandom, anche conoscendo $f(x)$. Questo poiché, per definizione di predicato hardcore, è impossibile indovinare $\mathbf{hc}(x)$ con probabilità significativamente maggiore di $\frac{1}{2}$ avendo a disposizione solo risorse polinomiali.

Il primo passo è costruire un generatore che espanda il seme iniziale di un bit.

Teorema 3. *Sia f funzione unidirezionale e sia **hc** un predicato hardcore di f. Allora $G(s) = (f(s), \mathbf{hc}(s))$ è un generatore pseudorandom con fattore di espansione $l(n) = n + 1$*

Osserviamo che non è affatto banale la possibilità di generare in maniera deterministica anche un unico bit random e quello che cerchiamo è un generatore con fattore di espansione molto più lungo. A partire dal teorema appena enunciato si può però dimostrare che è possibile costruire generatori pseudorandom che hanno fattore di espansione un qualunque polinomio.

Teorema 4. *Assumendo che esistano generatori pseudorandom con fattore di espansione $l(n) = n+1$, per ogni polinomio $p(\cdot)$ esiste un generatore pseudorandom con fattore di espansione $l(n) = p(n)$.*

Quello di cui necessitiamo per l'utilizzo del protocollo MAP1 è una famiglia di funzioni pseudorandom. L'ultimo passo è perciò costruire delle funzioni pseudorandom a partire dai generatori:

Teorema 5. *Assumendo l'esistenza di generatori pseudorandom con fattore di espansione $l(n) = 2n$ esistono funzioni pseudorandom e permutazioni pseudorandom.*

In definitiva, mettendo insieme i teoremi da 2 a 5 si ha:

Corollario 1. *Assumendo l'esistenza di permutazioni unidirezionali esistono generatori pseudorandom con fattore di espansione polinomiale, funzioni pseudorandom e permutazioni pseudorandom.*

2.1.3 Esistenza e candidati

Avendo a disposizione delle funzioni unidirezionali si possono costruire funzioni pseudorandom. Il problema dell'esistenza di funzioni pseudorandom si trasferisce quindi sull'esistenza di quelle unidirezionali.
Ma esistono funzioni unidirezionali? Allo stato attuale delle conoscenze non c'è una risposta. Ricordiamo che l'impossibilità di inversione di una funzione è, in effetti, una impossibilità computazionale. Provare questa impossibilità sarebbe equivalente a dimostrare che $P \neq NP$ [2]. Infatti, se si dimostrasse che esiste una funzione in P unidirezionale, si avrebbe che il problema di trovare un x tale che $f(x) = y$ con y assegnato sarebbe in NP, portando quindi a concludere che $P \neq NP$. Questo è uno dei problemi aperti (se non IL problema principe) in teoria della complessità. Se si trovasse che $P = NP$ si avrebbe che i problemi matematici su cui si basa la crittografia, alcuni elencati in seguito, sarebbero risolubili in tempi polinomiali, facendo cedere tutta la struttura dalle fondamenta. Non si disporrebbe, ad esempio, di funzioni unidirezionali e verrebbero meno i sistemi di crittografia a chiave

[2] In estrema sintesi P è la classe dei problemi per cui è noto un algoritmo che risolva il problema in tempo polinomiale; per un problema in NP può non esistere un algoritmo polinomiale che lo risolve, ma esiste un algoritmo polinomiale che, data una candidata soluzione, verifica se la soluzione è valida. Un esempio di problema in NP può essere la fattorizzazione di interi. È evidente che $P \subset NP$.

pubblica noti.

D'altra parte esistono all'atto pratico delle funzioni che sembrano essere valide. *Sembrano* perché, per quanto non sia dimostrato che siano effettivamente unidirezionali, in molti anni di crittanalisi non si è neanche dimostrato il contrario. Le principali candidate sono:

- *Fattorizzazione di numeri.* Scegliendo oculatamente p e q, numeri primi abbastanza grandi, si trova che, dato $n = pq$, è computazionalmente difficile risalire ai fattori che lo compongono. Questo problema è alla base del crittosistema più diffuso RSA.

- *Logaritmo discreto.* Sia p numero primo abbastanza grande e $a < p$ noto. Dato $g = a^n \bmod p$, è computazionalmente difficile risalire a n. Questo problema è alla base del crittosistema ElGamal o dell'algoritmo di Diffie-Hellman per lo scambio di una chiave segreta condivisa.

- *Reticoli geometrici.* Dati $v_1, ..., v_n \in \mathbb{R}^n$, $n > 0$, si dice *reticolo geometrico* ogni combinazione lineare di $v_1, ..., v_l$ a coefficienti interi. Dato un reticolo abbastanza grande, è computazionalmente difficile trovare un elemento di lunghezza minima appartenente al reticolo. Questo problema è alla base del crittosistema NTRU.

2.2 Implementazione: i cifrari a blocchi

Abbiamo finora analizzato come teoricamente si possano costruire delle funzioni pseudorandom; vediamo ora come queste vengano praticamente implementate. Rimarchiamo che per le permutazioni pseudorandom che presenteremo non si dispone di prove di sicurezza se non quella euristica data dalla resistenza dimostrata negli anni di utilizzo.

Analizzeremo la struttura dei cifrari a blocchi[3] e in particolare le *reti a sostituzione e permutazione*, introdotte da Shannon, e le *reti di Feistel*, introdotte da Feistel.

2.2.1 Rete a sostituzione e permutazione

Come detto, la proprietà principale di un cifrario a blocchi è quella di apparire come una permutazione random. Una permutazione random su $\{0,1\}^n$ richiederebbe per la sua descrizione $n \cdot 2^n$ valori; occore trovare un modo per costruire una funzione la cui rappresentazione sia concisa, ma il cui comportamento sia indistinguibile da una funzione random.

Paradigma di confusione e diffusione [4] Per raggiungere questo scopo Shannon suggerì l'idea che, invece che adoperare un'unica funzione, l'input fosse scomposto in piccole parti (*blocchi* di lunghezza fissa) e sottoposto più volte a funzioni di base semplici e perciò più facili da descrivere. Più precisamente questi passaggi sono formati da:

- Piccole funzioni random che modificano le porzioni di input loro fornite.
- Altre funzioni devono combinare i risultati delle funzioni random tra loro.

Ogni passaggio attraverso queste funzioni è detto *round*. Questo paradigma è noto come *paradigma di confusione e diffusione*

Rete a sostituzione e permutazione Questo schema è un'implementazione del paradigma di confusione e diffusione in cui le due funzioni base vengono realizzate rispettivamente attraverso *sostituzioni* e *permutazioni*. Nel dettaglio:
L'input viene diviso in blocchi di lunghezza fissa a cui si applicano vari *round* in cui:

[3] Nonostante il nome di cifrario, questi sono in realtà delle permutazioni pseudorandom, che vengono spesso utilizzate come blocco di base per sistemi di cifratura. Nel seguito useremo indifferentemente i termini *cifrario a blocchi* e *permutazioni pseudorandom*.

[4] Confusione e diffusione sono due proprietà necessarie in un qualunque procedimento di cifratura, perché questo sia imprevedibile. Confusione: la dipendenza del testo cifrato dalla chiave deve essere la più complicata possibile. Diffusione: la modifica di un singolo bit deve modificare l'intero output. Cfr [Shan]

- Le permutazioni random di ogni blocco sono realizzate attraverso *sostituzione* con una funzione specificata attraverso una tavola di valori, detta *S-box*, con il requisito che sia invertibile (cioè sia iniettiva e suriettiva).

- L'operazione di combinazione viene realizzata con una permutazione dei blocchi.

Diversi schemi utilizzano questo modello specificando diversamente le S-box, le funzioni di permutazioni, il numero di round o la lunghezza dei blocchi.
Un altro aspetto importante è costituito dalla modalità con cui si fornisce variabilità all'output: lo schema può essere inizializzato utilizzando una chiave a scelta. A seconda delle implementazioni dello schema può variare l'uso che viene fatto della chiave: questa può ad esempio specificare una determinata S-box oppure modificare il blocco con uno XOR.

AES

Un cifrario che utilizza questo tipo di schema è AES, noto anche come Rijndael, algoritmo di cifratura a blocchi proposto da Joan Daeman e Vincent Rijmen e divenuto nel 2001 il nuovo standard, sostituendo il DES. Ne diamo una brevissima descrizione:
AES prevede blocchi da 128 bit, una chiave lunga tra 128,192 o 256 bit con cui si compiono rispettivamente 10, 12 o 14 round.
AES opera su matrici di 4x4 byte, in cui ad ogni round:

- (*SubBytes*) si compie una sostituzione dei bytes, seguendo una tabella.

- (*ShiftRows*) le righe della matrice vengono traslate.

- (*MixColumns*) le colonne vengono combinate con un'operazione lineare.

- (*AddRoundKey*) viene aggiunta una chiave di round, ricavata dalla chiave di partenza attraverso un algoritmo di generazione di chiavi.

Si riconosce la struttura a sostituzione e permutazione.

2.2.2 Rete di Feistel

Un altro modo di costruire un cifrario a blocchi è dato dalla rete di Feistel. Si utilizzano le stesse funzioni di base (S-box, permutazioni e generatori di chiavi) che vengono però adoperate in maniera diversa.
Come visto, in uno schema a sostituzione e permutazione è richiesto che le S-box siano invertibili; nella costruzione di uno schema che voglia apparire casuale la richiesta di invertibilità rappresenta un'introduzione di ordine in contrasto con il caos desiderato. In uno schema a rete di Feistel questa richiesta viene meno, ma nonostante l'utilizzo di componenti non invertibili, lo schema costruito è globalmente invertibile.
Una rete di Feistel utilizza in ogni round una *f-function* che può non essere invertibile e che al suo interno utilizza una chiave e delle componenti come S-box e permutazioni.
Vediamo adesso la rete di Feistel vera e propria, che è indipendente dalla formulazione della specifica *f-function*.

L'input x viene dapprima separato in due metà x_1 e x_2, ovvero per uno schema a blocchi di $2n$ bit, l'input viene diviso in due parti da n bit. Una delle parti viene data in input alla *f-function* e l'output così ottenuto viene utilizzato per modificare l'altra metà, dopodiché le due metà vengono scambiate. Più rigorosamente:

- Preso un input x, si considerano x_1 e x_2 le due metà di x.
- Sia $L_0 = x_1$ e $R_0 = x_2$
- Per $i = 1...d$ (con d il numero di round)
  - Si calcola $L_i = R_{i-1}$ e $R_i = L_{i-1} \oplus f_i(R_{i-1})$, con f_i *f-function* dell'i-simo round.
- L'output y è (L_d, R_d)

Lo schema in Figura 2.1 evidenzia visivamente la definizione appena data.[5]

[5]Nota storica: la struttura simmetrica con scambio delle due metà di un blocco è stata pensata

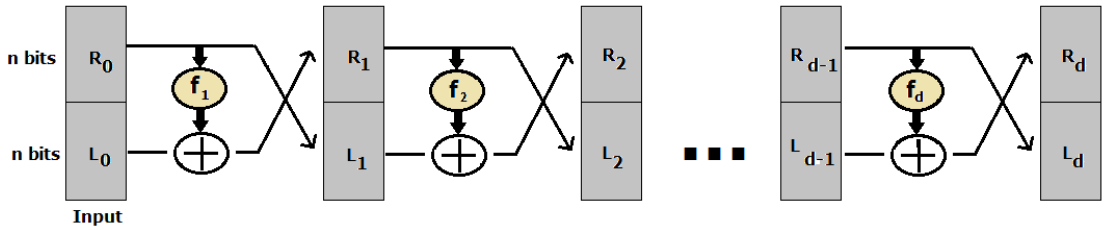


Figura 2.1: Schema di una rete di Feistel

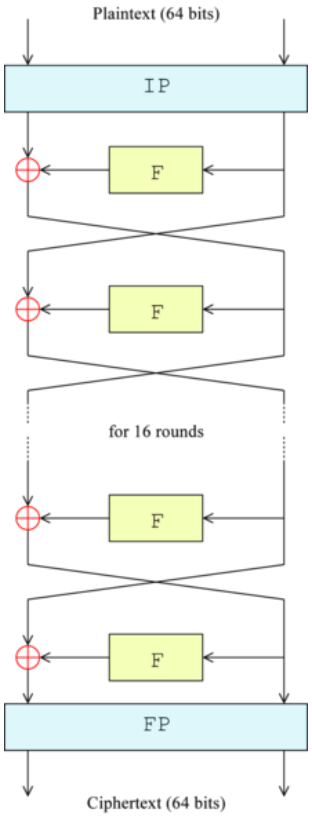
DES

Il DES, Data Encryption Standard, fu sviluppato negli anni '70 in IBM e adottato come standard a partire dal 1976. Pur non essendo più un algoritmo sicuro a causa della dimensione ridotta della chiave, motivo per cui è stato sostituito da AES, questo algoritmo è molto importante sia per motivi storici, sia perché una sua variante, il triple DES, è utilizzata ancor oggi.
Storicamente è stato il primo cifrario a blocchi ad essere pubblicato ed è quindi stato sottoposto a una notevole attenzione da parte di crittoanalisti di tutto il mondo. Tutti gli studi compiuti sul DES sono stati di fondamentale importanza anche nella progettazione di tutti gli algoritmi successivi.
Rimarchiamo come la debolezza di DES sia solo nella lunghezza della chiave troppo corta e nella vulnerabilità ad un attacco esaustivo sullo spazio delle chiavi.

Il DES è un cifrario a rete di Feistel costituito da 16 round con una dimensione del blocco di 64 bit e 56 bit di chiave. La *f-function* interna non è invertibile, prevede in input 32 bit ed è la stessa per ogni round. Quello che cambia round per round è la chiave utilizzata: viene presa una sottochiave di 48 bit dalla chiave generale secondo uno schema detto *key schedule*. Nel dettaglio:

per far sì che l'algoritmo di decifrazione sia identico (a parte l'ordine delle chiavi) a quello di cifratura, proprietà importante per un'implementazione hardware nel 1970

Descrizione



- Viene applicata una permutazione iniziale (IP) all'input
- Per ogni round viene applicato lo schema di Feistel, dove ogni *f-function* realizza le seguenti mosse:
  1. Espansione: il mezzo blocco di 32 bit viene portato a 48 bit
  2. Aggiunta della chiave: con un'operazione di XOR viene aggiunta la chiave di round
  3. Sostituzione: Viene realizzata una sostituzione tramite un S-box, riportando contemporaneamente l'output a 32 bit
  4. Permutazione: I 32 bit vengono riordinati secondo permutazioni fisse in delle P-box.
- Viene applicata una ultima permutazione (FP).

È evidente, anche graficamente nella figura, la struttura a schema di Feistel.

Capitolo 3

Conclusioni

Il problema dell'autenticazione tra le parti è uno degli aspetti cardine della crittografia moderna. Grazie alla formulazione rigorosa data da Bellare e Rogaway e alla loro dimostrazione di sicurezza è possibile assumere che esistano dei protocolli matematicamente sicuri che garantiscano l'autenticazione, sotto l'ipotesi che esistano delle funzioni pseudorandom.
Dimostrare l'esistenza (o meno) di queste porterebbe a un grande salto in avanti nello studio della crittografia in quanto, come visto, equivarrebbe a risolvere il problema $P = NP$.
Allo stato attuale si utilizzano spesso variazioni del protocollo che, invece di funzioni pseudorandom, adoperano cifratura simmetrica, firma digitale o funzioni hash che risultano essere *euristicamente sicure*.

Bibliografia

[BeRog] Mihir Bellare, Phillip Rogaway, *Entity Authentication and Key Distribution*,1993, disponibile presso `https://cseweb.ucsd.edu/~mihir/papers/eakd.pdf`

[KaLi] Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, Chapman and Hall/CRC , 2007

[GoLe] Oded Goldreich and Leonid A.Levin, *A Hard-Core Predicate for all One-Way Functions*, 1989, disponibile presso `http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.95.2079&rep=rep1&type=pdf`

[Shan] C.E.Shannon, *Communication Theory of Secrecy Systems*, 1949, disponibile presso `http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf`

[DoDwNa] D.Dolev, C.Dwork, M.Naor, *Non-malleable cryptography, Proceedings of the Twenty Third Annual Symposium on the Theory of Computing, ACM,1991.*