Daniele Venturi

Crittografia nel Paese delle Meraviglie

Springer

UNITEXT

Ad Anna e al suo bellissimo sorriso

Daniele Venturi

Crittografia nel Paese delle Meraviglie

Springer

Daniele Venturi
Dipartimento di Ingegneria, Elettronica e Telecomunicazioni (DIET),
Sapienza Università di Roma

ISBN 978-88-470-2480-9 ISBN 978-88-470-2481-6 (eBook)
DOI 10.1007/978-88-470-2481-6

Springer Milan Dordrecht Heidelberg London New York

Riprodotto da copia camera-ready fornita dall'autore
Copertina: Simona Colombo, Milano
Stampa: GECA Industrie Grafiche, Cesano Boscone (Mi)

Stampato in Italia

Springer-Verlag Italia S.r.l., Via Decembrio 28, I-20137 Milano
Springer fa parte di Springer Science + Business Media (www.springer.com)

Prefazione

La crittografia è un'arte antica che comprende un insieme di tecniche per rendere un dato sistema "sicuro". Un sistema (pensato per soddisfare determinati requisiti) è sicuro, se esso è in grado di continuare a mantenere la sua funzionalità anche in presenza di un fastidioso attaccante che tenta in tutti i modi di impedire che ciò avvenga.

Il progetto di uno schema crittografico sicuro è un compito molto delicato, in quanto richiederebbe di prevedere tutti i possibili abusi del sistema da parte dell'attaccante. In passato, questo problema era indirizzato per lo più affidandosi all'intuito ed all'esperienza, creando così un gioco del "gatto col topo" in cui nuovi schemi venivano ideati e nuovi attacchi in grado di violarne la sicurezza venivano (prima o poi) scoperti. L'approccio odierno, seguito dai crittografi moderni per superare questo stato di affari, si basa su metodologie rigorose, che hanno trasformato la crittografia da arte a vera e propria scienza. Tali metodologie costituiscono l'oggetto principale di questo testo.

Per fondare la crittografia su basi rigorose, è necessario astrarre le caratteristiche salienti di un sistema reale (e dei possibili attacchi ad esso) e dimostrarne la sicurezza in modo formale. Si finisce così per definire una specie di "realtà alternativa" in cui alcune caratteristiche del sistema reale di partenza sono inevitabilmente idealizzate. Tutti i risultati che si riescono a dimostrare rigorosamente in questo "reame della sicurezza dimostrabile" — che nel libro sarà identificato con il Paese delle Meraviglie di Carroll — sono validi solamente al suo interno; si capisce quindi che il punto fondamentale è far sì che il nostro "Paese delle Meraviglie" crittografico descriva la realtà nel modo più accurato possibile. Seppure, tipicamente, si riesce a rappresentare la realtà in modo sufficientemente accurato, a volte la pratica ci ha insegnato che è possibile violare un sistema ritenuto sicuro sfruttando caratteristiche che non abbiamo preso in considerazione nel nostro modello idealizzato della realtà. Pertanto, la metafora carrolliana ci aiuta a non dimenticare che i risultati che dimostreremo sono validi in un modello astratto della realtà e che lo scopo della ricerca in quest'area è, essenzialmente, quello di rendere questo mondo il più possibile vicino a rappresentare accuratamente la realtà stessa.

Presenteremo i requisiti di base di un generico sistema di comunicazione, facendo riferimento ad alcuni scenari astratti che vedranno coinvolti i personaggi del mondo di Carroll: Alice, il Bianconiglio, il Cappellaio Matto e così via. (Normalmente, simili scenari sono considerati nella letteratura, facendo riferimento ad Alice, Bob, Charlie e via dicendo.)

Uso del libro. Il libro è pensato per essere usato come base di un corso universitario introduttivo alla crittografia, tipicamente nella Facoltà di Scienze Matematiche, Fisiche e Naturali (per i corsi di laurea in informatica teorica e matematica applicata) e nella Facoltà di Ingegneria (per i corsi di laurea in Ingegneria Informatica e delle Telecomunicazioni). Una possibile organizzazione della didattica è suggerita di seguito:

Facoltà di Scienze (corso base). Introduzione: Il reame della sicurezza dimostrabile (Par. 1.1 e 1.3). Cifrari antichi (Par. 2.1) e teoria di Shannon (Par. 2.2), cenni sul risultato di Shannon (Par. 2.3). Teoria della pseudocasualità: impredicibilità ed indistinguibilità (Par. 3.1 e Par. 3.2). Funzioni hash: principi (Par. 4.1) e Merkle-Damgård (Par. 4.2). Richiami elementari di teoria dei numeri: algoritmo euclideo (App. B.1), congruenze (App. B.2). Problemi computazionali: primalità (App. C.1), fattorizzazione (App. C.2) e logaritmi discreti (App. C.3). Cifrari simmetrici: sicurezza CPA (Par. 5.1), il DES ed AES (Par. 5.3). Cifrari asimmetrici: sicurezza CPA (Par. 6.1), RSA (Par. 6.2) ed ElGamal (Par. 6.3). Codici autenticatori di messaggio: definizione (Par. 7.1), costruzioni teoriche (Par. 7.2). Firme digitali: definizione (Par. 8.1), DSS (Par. 8.4). Protocollo di Diffie-Hellmann (Par. 11.1).

Facoltà di Scienze (corso avanzato). Dimostrazione del teorema di Shannon (Par. 2.3). Bit e predicati estremi (Par. 3.3): applicazioni ai PRG (Par. 3.4) ed ai cifrari simmetrici (Par. 5.2). La trasformazione GGM ed analisi delle reti di Feistel (Par. 5.2). Residuosità quadratica (App. B.3) ed il cifrario di Goldwasser-Micali (Par. 6.4). Sicurezza CCA: il caso simmetrico (Par. 7.4), il caso asimmetrico (Par. 6.5). Curve ellittiche in crittografia (App. B.4) ed applicazioni ai cifrari ed alle firme digitali. Il modello dell'oracolo casuale (Par. 4.4): applicazioni alle firme digitali (Par. 8.2). Accoppiamenti bilineari e cifratura su base identità (Cap. 10). Reticoli geometrici ed il cifrario di Regev (Cap. 9). Autenticazione di persone: paradigmi generici (Par. 11.2 e Par. 11.3) e la famiglia di protocolli HB (Par. 11.4). Protocolli su base password (Par. 12.4 e Par. 12.2). Protocolli a conoscenza nulla (Cap. 13). Condivisione di segreti, schemi di impegno e trasferimento immemore (Cap. 14).

Facoltà di Ingegneria (corso base). Introduzione: Il reame della sicurezza dimostrabile (Par. 1.1 e 1.3). Cifrari antichi (Par. 2.1) e teoria di Shannon (Par. 2.2), cenni sul risultato di Shannon (Par. 2.3). Teoria della pseudocasualità: impredicibilità ed indistinguibilità (Par. 3.1 e Par. 3.2). Funzioni hash: principi (Par. 4.1) e costruzioni reali (Par. 4.3). Nozioni elementari di teoria dei numeri: cenni su algoritmo euclideo (App. B.1) e congruenze (App. B.2).

Confidenzialità: il DES ed AES (Par. 5.3), modi operativi (Par. 5.4), cifrari a flusso (Par. 5.5), RSA (Par. 6.2) ed Elgamal (Par. 6.3). Autenticazione di messaggio: principi (Par. 7.1) ed HMAC (Par. 7.3). Firme digitali: cenni su RSA (Par. 8.2) ed il DSS (Par. 8.4). Protocollo di Diffie-Hellmann (Par. 11.1). Infrastrutture a chiave pubblica (Par. 10.1).

Ritengo inoltre che il libro sia un ottimo punto di partenza per i dottorandi italiani che si avvicinano per la prima volta al mondo della ricerca in crittografia teorica.

Prerequisiti. Una comprensione del testo non richiede alcun prerequisito specifico, se non una certa "maturità matematica" ed una familiarità con il linguaggio tipico della teoria delle probabilità (distribuzioni, variabili aleatorie e via dicendo). Una panoramica della terminologia di base in quest'ambito si trova in Appendice A.

Termini inglesi. Siccome il libro è in lingua italiana, la scelta è stata quella di "tradurre" tutte le espressioni di uso comune in inglese. (Fanno eccezione gli acronimi, in quanto una loro traduzione avrebbe generato troppa confusione.) Seguire questa strada ha lo svantaggio che uno studente che studia i rudimenti della materia su questo testo, si potrebbe trovare spiazzato leggendo gli articoli specialistici, non avendo familiarità con la terminologia standard in crittografia. Per evitare che ciò accada, la prima volta che un termine italiano è usato al posto di uno inglese, la terminologia standard viene segnalata tra parentesi (in corsivo). Il glossarietto crittografico alla fine del libro tiene traccia di tutti i termini tradotti.

Approfondimenti e standard. Di seguito alcuni riferimenti per approfondimenti e curiosità.

Enti. Esiste un ente internazionale no-profit per lo sviluppo e la ricerca in crittografia: l'"International Association for Cryptographic Research" (IACR, `http://www.iacr.org/`).

Il "National Institute of Standards and Technology" (NIST) è un'agenzia del governo degli Stati Uniti d'America che si occupa della gestione delle tecnologie. Il NIST pubblica i "Federal Information Processing Standard" (FIPS), che specificano ad esempio il DES ed AES. Spesso gli standard FIPS sono varianti di quelli ANSI, IEEE, ISO etc.

Conferenze. I nuovi contributi di ricerca in crittografia vengono pubblicati su alcune conferenze specifiche: TCC (`http://www.wisdom.weizmann.ac.il/~tcc/`), Asiacrypt, Eurocrypt e Crypto (tutte sponsorizzate dallo IACR), PKC (`http://www.ipkc.org/`) ed SCN (`http://scn.dia.unisa.it/`) per nominarne alcune. Alcuni risultati vengono pubblicati anche su conferenze di carattere più generale: STOC (`http://www2.research.att.com/~dsj/stoc11/stoc11.html`), FOCS (`http://ieee-focs.org`) ed ICALP (`http://icalp11.inf.ethz.ch/`).

Errata. Nessun libro è privo di errori. Sarei felicissimo di ricevere commenti, positivi o negativi che siano, e correzioni:

`danone83@gmail.com`

Un'errata corrige sarà resa disponibile sulla mia pagina web personale.

Ringraziamenti. Il primo ringraziamento è per Andrea Baiocchi, per avermi chiesto di scrivere questo libro. So di non essere stato uno studente di dottorato "facile", ma Andrea è sempre stato disponibile per consigli e chiarimenti. La stesura del libro, che ha richiesto circa tre anni, mi ha accompagnato per tutto lo svolgimento del dottorato, rivelandosi tra l'altro un'ottima occasione per studiare lo stato dell'arte delle diverse branche della crittografia.

Sono in debito con Stefan Dziembowski e Krzysztof Pietrzak: sono stati i miei mentori (rispettivamente a Roma e ad Amsterdam) e tutto quello che so in ambito crittografico lo devo a loro. Grazie per essere stati sempre disponibili a discutere un problema e per aver fornito spiegazioni esaustive, anche alle domande semplici.

Sebbene il testo sia il primo in italiano sul tema della crittografia teorica (in senso strettamente informatico), non è sicuramente il primo nel panorama internazionale in cui si trovano diversi libri con un taglio simile. Due libri in particolare restano un punto di riferimento: il libro di Katz e Lindell "Introduction to Modern Cryptography" ed il libro di Goldreich "Foundations of Cryptography". Essi sono stati (inevitabilmente) una fonte di ispirazione continua ed insostituibile per alcune parti di questo testo e per molti degli esercizi proposti. Agli autori la mia gratitudine per aver fornito strumenti così chiari, illuminanti ed utili.

Un secondo spunto per la presentazione di alcuni argomenti, sono state le lezioni tenute da alcuni crittografi in diverse parti del mondo: Yevgeniy Dodis all'Università di New York, Jonathan Katz all'Università del Meryland,

Luca Trevisan all'Università di Berkeley (ora a Stanford), Ran Canetti all'Università di Tel Aviv, Susan Hohenberger all'Università Johns Hopkins, Stefan Dziembowski all'Università Sapienza di Roma, Dario Catalano all'Università di Catania.

Diverse persone mi sono state vicine in questi anni, ed hanno contribuito (a volte indirettamente) a questo libro: Sebastian Faust, Abishek Jain, Eike Kiltz, Serge Fehr, David Cash, Francesco Davì, Alp Bassa, Giannicola Scarpa, Özgür Dagdelen, Andreas Peter, Cristina Onete, Marc Fischlin, Ivan Damgård, Alessandra Scafuro, Ivan Visconti e tantissimi altri. Grazie a tutti voi per il supporto e per i bei momenti passati insieme!

La grafica del progetto è stata curata da Andrea Chronopoulos, che ha realizzato i personaggi (ispirati a quelli della novella di Carroll) e la copertina del libro con fantasia ed esuberanza. La maggior parte delle citazioni all'inizio di ciascun capitolo è tratta dall'edizione italiana Einaudi dell'opera di Carroll (con traduzione a cura di Alessandro Ceni).

La stesura del testo è stata realizzata con LaTeX usando la classe *Memoir* di Peter Wilson. È doveroso ringraziare Enrico Gregorio, Lorenzo Pantieri, Lapo Mori e tutti gli altri membri del forum del GUIT con cui ho discusso negli anni passati.

Grazie a Giorgia Azzurra Marson, per aver letto una prima bozza del testo e per aver fornito commenti esaustivi e segnalato imprecisioni.

Grazie ad Alessandro, per avermi fatto capire che la citazione non è solo un abbellimento e per le infinite discussioni filosofiche sul ruolo della crittografia e della ricerca.

In ultimo, desidero ringraziare Anna, per il suo supporto incondizionato, per la sua presenza costante e per il suo sorriso.

Roma, gennaio 2011 *Daniele Venturi*

Indice

Appendici Matematica

Capitolo 1

Introduzione

Il Bianconiglio inforcò gli occhiali. «Da dove debbo iniziare, piaccia a Vostra Maestà?». «Inizia dall'inizio» — disse il Re molto solennemente, — «e va' avanti finché non arrivi alla fine; poi fermati.»

Lewis Carroll, Attraverso lo Specchio [Car71]

Con il vocabolo *sicurezza* (dal latino "sine cura", ovvero "senza preoccupazione") si intende rappresentare il grado di protezione di un sistema contro pericoli, danneggiamenti e perdite. L'oggetto principale del nostro studio sarà la sicurezza nel contesto della comunicazione. L'arte della "scrittura nascosta" (ovvero la *crittografia*), in effetti, è nata proprio dall'esigenza di celare il contenuto informativo dei messaggi scambiati durante una comunicazione a distanza. Tale esigenza (molto sentita sopratutto in tempo di guerra) ha origini molto antiche: se ne trova traccia nella storia sin dai tempi dell'imperatore romano Giulio Cesare e degli antichi Greci.

Lo sviluppo dell'era digitale (segnata dalla comparsa dei primi calcolatori elettronici e, più avanti, dall'entrata in scena di Internet) ha portato con sé tante nuove applicazioni con requisiti di "sicurezza"[1] sempre più complessi. Questo cambio di paradigma ha fatto sì che la crittografia trascendesse il suo significato originale di "scrittura nascosta", trovando applicazioni in svariati contesti. In effetti, ad oggi, possiamo definire la crittografia come un insieme di tecniche che contribuiscono (in maniera preventiva) alla sicurezza globale di un dato sistema di comunicazione. Tuttavia, è bene sottolineare che la crittografia da sola non basta a rendere il sistema sicuro in ogni situazione. Volendo fare una metafora, la crittografia è come il lucchetto che protegge una porta: raramente i ladri cercheranno di entrare da una porta adeguatamente protetta, piuttosto romperanno le finestre o ruberanno le chiavi al proprietario!

[1] L'uso delle virgolette è dovuto, in quanto il termine nel suo significato generale è impreciso e, come vedremo, necessita di essere definito rigorosamente per ciascuna applicazione.

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_1,

Non dobbiamo mai dimenticare, dunque, che un un sistema *è sempre tanto sicuro quanto è sicuro il suo "anello più debole"*: non serve a molto blindare alla perfezione una porta se si lascia aperta una finestra!

Guida per il lettore. Nel Paragrafo 1.1, cercheremo di delineare le esigenze degli utenti appartenenti ad un generico contesto di comunicazione; come vedremo tali esigenze corrispondono a dei *requisiti di sicurezza* che i nostri schemi crittografici dovranno soddisfare. Introdurremo quindi il reame della sicurezza dimostrabile, ovvero il nostro Paese delle Meraviglie crittografico. Nel Paragrafo 1.2 tracceremo quindi una mappa del libro, spiegando il contenuto dei singoli capitoli; daremo anche qualche breve cenno ad altri scenari interessanti non trattati nel testo, indirizzando il lettore interessato ad approfondire alla letteratura. Siccome la maggior parte degli schemi crittografici che studieremo tenta di proteggere un sistema da attaccanti con risorse computazionali limitate, nel Paragrafo 1.3 introdurremo gli strumenti per rappresentare il concetto di efficienza computazionale, usando il linguaggio della *teoria della complessità*. La notazione usata nel testo è descritta, infine, nel Paragrafo 1.4.

1.1 Il reame della sicurezza dimostrabile

Immaginiamo il seguente scenario di comunicazione nel Paese delle Meraviglie: Alice ed il Bianconiglio comunicano attraverso un canale non-protetto[2] controllato dalla Regina Rossa (cf. Fig. 1.1). In un tale scenario possiamo considerare due requisiti fondamentali:

- *Confidenzialità.* Alice ed il Bianconiglio si preoccupano di *celare il contenuto* dei messaggi scambiati, in modo che, seppur la Regina Rossa li intercetti, non sia in grado di risalire al loro contenuto originale. D'altra parte Alice ed il Bianconiglio devono essere in grado di "interpretare" i messaggi ricevuti in modo opportuno, ricavando il loro contenuto originario. (Ritroviamo qui l'esigenza primaria della "comunicazione segreta".)

- *Integrità.* Alice ed il Bianconiglio vogliono essere sicuri che i messaggi *transitino immutati* sul canale, senza che la Regina Rossa possa modificarli a loro insaputa. (Ciò è di fondamentale importanza, ad esempio, nelle transazioni bancarie su Internet.)

[2] Si intende che chiunque è in grado di controllare il canale e leggere e/o modificare il contenuto dei messaggi inoltrati attraverso esso.

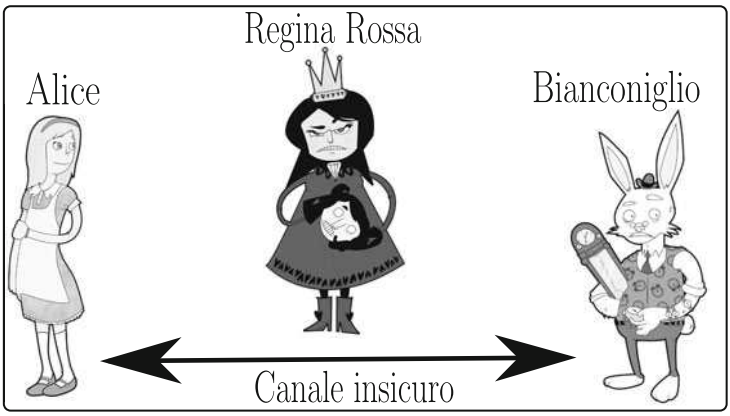


Fig. 1.1. Alice ed il Bianconiglio vogliono comunicare attraverso un canale insicuro controllato dalla Regina Rossa

Possiamo quindi spingerci oltre e pensare ad uno scenario più complesso, in cui Alice ed il Bianconiglio eseguono un *protocollo crittografico*[3] con scopi diversi:

- *Autenticazione.* Alice vuole essere "riconosciuta" dal Bianconiglio, dimostrando di essere *veramente* Alice. In questo contesto, la Regina Rossa non deve essere in grado di essere riconosciuta come Alice. (Questa esigenza è fondamentale in tutte le applicazioni in cui è necessario che le parti coinvolte si autentichino prima che una qualsiasi azione possa avere luogo.)

- *Conoscenza nulla.* Alice vuole dimostrare al Bianconiglio la veridicità di una data affermazione, senza rivelare nient'altro oltre il fatto che tale affermazione è appunto vera.

- *Computazione sicura.* Alice ed il Bianconiglio vogliono calcolare una funzione di alcuni input assicurandosi che il risultato sia corretto e mantenendo la sicurezza degli input stessi. (Come vedremo questo scenario è estendibile al caso di più giocatori, in cui le parti coinvolte sono più di due.)

Sicurezza dimostrabile. Il progetto di un sistema sicuro è un compito arduo. In primo luogo, osserviamo che ciò richiede di definire in modo chiaro ed

[3]Per protocollo intendiamo qui uno scambio di messaggi con una "forma" prestabilita.

esplicito il significato della parola "sicuro". (Ovviamente tale significato sarà diverso a seconda del contesto in cui la primitiva crittografica viene utilizzata.) Ma, anche ammettendo di aver chiaro cosa significhi "sicurezza" per una data primitiva in un dato contesto, come assicurarsi che essa effettivamente non sia violabile (ovvero che essa sia in grado di resistere a tutti gli attacchi)? Il problema principale, infatti, è che il progetto di un sistema sicuro deve considerare la presenza di avversari che abuseranno del sistema per poterlo violare. In altri termini, diversamente da quanto avviene nella maggior parte delle discipline, ciò che deve preoccupare un crittografo non è l'uso normale del sistema, ma un suo eventuale abuso da parte di un attaccante, il che è ovviamente molto più difficile da prevedere. Sarebbe desiderabile, quindi, che la nostra primitiva sia resistente non solo agli attacchi noti al momento del progetto, ma anche ai possibili attacchi che possono essere pensati in futuro.

In passato (ed in parte ancora oggi) l'analisi di sicurezza dei sistemi avveniva in modo per lo più euristico: si tenta di spiegare (affidandosi all'intuizione ed all'esperienza) perché non è possibile violare la sicurezza in un attacco reale alla primitiva. Questo approccio si è rivelato più volte fallimentare (e sopratutto pericoloso!) portando al progetto di applicazioni apparentemente sicure che poi sono state violate negli anni a venire. (Un esempio concreto si può trovare nel sistema di cifratura usato nelle reti cellulari GSM [BBK03].)

Questo stato di affari ha reso necessario un ripensamento dell'approccio generale per il progetto di una primitiva sicura. La svolta si è avuta nei primi anni '80, quando Goldwasser e Micali [GM82] hanno gettato le fondamenta del *reame della sicurezza dimostrabile*, trasformando la crittografia da arte a vera e propria scienza.

L'idea è quella di fornire una "prova matematica" che un dato sistema è sicuro in presenza di una certa classe di attacchi (la più vasta possibile). In questo modo anche se in futuro verrà pensato un nuovo attacco, il nostro sistema sarà in grado di resistergli (purché tale attacco rientri nella classe di attacchi considerata). La "ricetta" per dimostrare che una data primitiva crittografica è sicura segue più o meno lo schema seguente:

1. Si astrae la realtà, definendo un modello che contenga tutte le caratteristiche salienti presenti nel mondo reale in cui la primitiva è utilizzata.

2. Si definisce cosa significa per la primitiva essere "sicura" in questo modello.

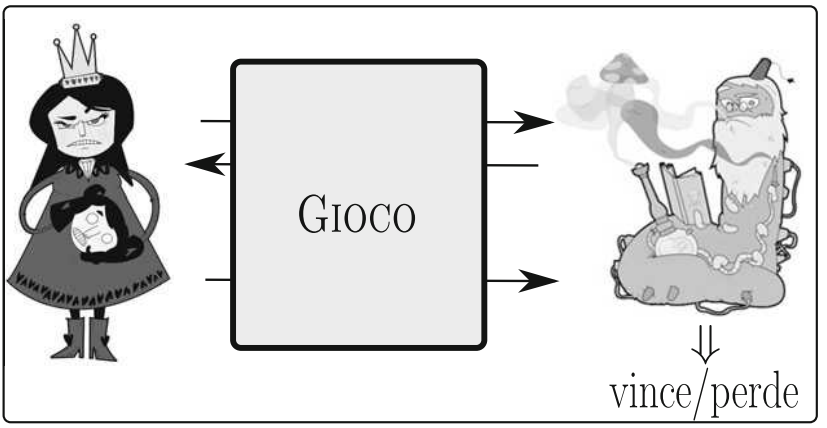


Fig. 1.2. Gioco per definire la sicurezza di Π

3. Si mostra che nessun attaccante di un certo tipo, che agisce nel modello definito al punto (1), è in grado di violare la definizione di sicurezza data al punto (2).

Per realizzare i punti (1) e (2), solitamente, si introduce un "gioco" tra un attaccante ed uno sfidante. Nel Paese delle Meraviglie tale gioco vede come protagonisti la Regina Rossa ed il Brucaliffo: il Brucaliffo è un "oracolo" che astrae le modalità con cui la Regina può attaccare il sistema nella realtà (cf. Fig. 1.2). Durante il gioco la Regina può interagire con il Brucaliffo secondo alcune regole prestabilite. La condizione di vittoria del gioco, determina il successo della Regina in un attacco ed un sistema è detto sicuro se vincere il gioco è impossibile (o comunque, in un certo senso, "difficile"). Osserviamo sin da subito che, siccome una dimostrazione di sicurezza sarà valida *solo* nel modello definito al punto (1) per la classe di attaccanti definita al punto (2), la scelta delle "regole del gioco" è cruciale.

Quanto alla natura dell'avversario, si è soliti distinguere due diversi scenari:

- *Attaccanti illimitati computazionalmente.* In questo caso si parla di *sicurezza incondizionata* (o anche *sicurezza perfetta*). Questa nozione nasce con il lavoro di Shannon [Sha49] e porta a progettare primitive con sicurezza dimostrabile contro avversari le cui risorse computazionali sono infinite. Intuitivamente l'avversario non è in grado di violare la sicurezza del sistema perché non ha mai abbastanza "informazione" per riuscire in un attacco.

- *Attaccanti limitati computazionalmente.* In questo caso si parla di *sicurezza computazionale*. L'idea è quella di porre un limite al potere computazionale dell'attaccante, costruendo primitive che potrebbero essere violate (in linea teorica) da un avversario con abbastanza tempo a disposizione (talmente tanto da rendere improbabile tale evento).

Riduzioni crittografiche. È naturale chiedersi perché si debbano costruire sistemi computazionalmente sicuri, quando è possibile costruire sistemi con sicurezza incondizionata. La risposta a questa domanda risiede nell'*efficienza*. Come avremo modo di vedere nel Capitolo 2, infatti, gli schemi con sicurezza incondizionata sono purtroppo *inerentemente inefficienti* (e a volte addirittura impossibili da ottenere). Il passaggio al concetto di sicurezza computazionale ci consentirà di progettare sistemi crittografici efficienti (e quindi utilizzabili in pratica), mantenendo un livello accettabile di sicurezza.

Per quantificare il potere computazionale dell'attaccante, useremo alcune nozioni di teoria della complessità computazionale. Restringere la classe di attaccanti a quelli con risorse computazionali limitate, ci consentirà di "ridurre" la sicurezza del sistema alla difficoltà nel risolvere alcuni problemi computazionali ritenuti difficili — almeno con le risorse di cui dispone l'attaccante.[4] Si parla in effetti di *approccio riduzionista*; tale approccio costituisce una delle tecniche principali per realizzare il punto (3) della nostra ricetta per dimostrare che una data primitiva è sicura (vedi sopra).

Consideriamo una primitiva crittografica generica Π la cui sicurezza sia definita attraverso un gioco tra la Regina Rossa ed il Brucaliffo e fissiamo le risorse computazionali a disposizione della Regina. Supponiamo d'altra parte che la Regina Bianca debba risolvere un'istanza di un problema P ritenuto difficile: dato un input per il problema P non è possibile calcolare la soluzione in modo "efficiente" con le risorse computazionali disponibili. Una riduzione da Π a P significa che se la Regina Rossa è in grado di vincere il gioco che definisce la sicurezza della primitiva Π, la sua strategia può essere usata dalla Regina Bianca per risolvere un'istanza del problema P (cf. Fig. 1.3). In questo senso, violare la sicurezza di Π è equivalente a risolvere P: se riteniamo che ciò sia difficile, possiamo considerare la nostra primitiva sicura.[5]

[4] Tali problemi sono ritenuti "difficili" in quanto l'uomo tenta di risolverli in modo "soddisfacente" da anni, senza successo. Il significato dei termini "difficile" e "soddisfacente" sarà chiarito in modo rigoroso nel Paragrafo 1.3.

[5] Notare che la terminologia "sicurezza dimostrabile" è in un certo senso fuorviante, in quanto in effetti non si dimostra che una primitiva è sicura in senso assoluto, ma piuttosto che un attacco alla primitiva in questione può essere usato per risolvere un problema computazionale ritenuto difficile.

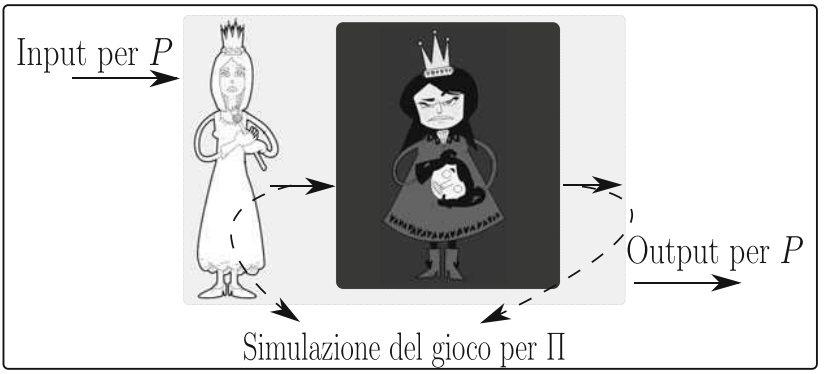


Fig. 1.3. Riduzioni crittografiche: la Regina Bianca deve simulare l'"ambiente" per la Regina Rossa abbastanza bene da far sì che quest'ultima sia convinta di attaccare Π

La controversia di Koblitz e Menezes. In alcuni articoli [KM07] Koblitz e Menezes hanno criticato alcuni aspetti del reame della sicurezza dimostrabile. L'opinione generale, tuttavia, è che l'approccio della sicurezza dimostrabile sia l'unico compatibile con la scienza [Gol06; Dam07], in quanto, citando lo stesso Goldreich: "è possibile costruire una cabina senza fondamenta, ma non una solida abitazione".

1.2 Mappa del libro

I requisiti di confidenzialità ed integrità di messaggio saranno trattati nella prima parte del libro (Capitoli 2 - 10). I requisiti di autenticazione, conoscenza nulla e computazione sicura, invece, sono oggetto della seconda parte (Capitoli 11 — 14). Le appendici forniscono le basi matematiche necessarie a comprendere alcune parti del testo, con particolare riferimento alla teoria delle probabilità, alla teoria dei numeri ed agli algoritmi migliori conosciuti per risolvere alcuni problemi computazionali ritenuti "difficili".

Più nello specifico:

- Il Capitolo 2 discute con maggiore dettaglio la nozione di sicurezza incondizionata nel contesto della confidenzialità, analizzando il lavoro di Shannon [Sha49] sui cifrari a segretezza perfetta.

- I Capitoli 3 e 4 introducono (rispettivamente) i concetti base di indistinguibilità e pseudocasualità e le funzioni hash; come vedremo questi concetti giocheranno un ruolo essenziale nei capitoli successivi.
- I Capitoli 5 e 6 indirizzano il problema della confidenzialità nel contesto delle tecniche simmetriche (ovvero quando Alice ed il Bianconiglio dispongano di una chiave segreta condivisa) e di quelle asimmetriche (ovvero quando Alice ed il Bianconiglio non condividono a priori nessun segreto).
- I Capitoli 7 e 8 indirizzano, rispettivamente, il problema dell'integrità di messaggio nel contesto delle tecniche simmetriche ed asimmetriche.
- Il Capitolo 9 è dedicato alla crittografia basata sui cosiddetti reticoli geometrici; le primitive costruite in questo contesto hanno l'attrattiva di basarsi sulla difficoltà di alcuni problemi computazionali la cui complessità è ben compresa dalla comunità matematica (e per cui, in alcuni casi, non esistono attacchi efficienti neanche disponendo di un calcolatore quantistico).
- Il Capitolo 10 è dedicato ancora al requisito di confidenzialità, assumendo però che Alice possa usare direttamente la sua "identità" per comunicare in segreto con il Bianconiglio.
- Il Capitolo 11 si occupa di diverse forme (via via più evolute) di autenticazione sicura.
- Il Capitolo 12 è dedicato all'uso delle password in crittografia. Una password è una parola appartenente ad un certo dizionario, che può essere memorizzata da un umano per scopi crittografici.
- Il Capitolo 13 descrive i protocolli a conoscenza nulla: è possibile dimostrare un'affermazione, senza che dalla relativa dimostrazione trapeli null'altro se non la veridicità dell' affermazione stessa?
- Il Capitolo 14 si occupa del problema della computazione sicura, nel caso di due o più giocatori.

Altri possibili scenari. Abbiamo dato un assaggio di quelli che sono i requisiti fondamentali di sicurezza in contesto crittografico. Oltre quelli citati, esistono tanti altri scenari interessanti (e di rilevanza pratica) la cui trattazione esula dagli scopi del testo. Alcuni di essi sono suggeriti di seguito, indirizzando il lettore interessato direttamente alla letteratura.

- *Nozioni di anonimato.* È possibile soddisfare i requisiti di confidenzialità ed integrità, in modo che non si possa celare *chi* sta inviando o ricevendo un determinato messaggio? Si veda [Cha81; Bel+01; BMW03; BKM09] per un'introduzione.

- *Crittografia quantistica.* La realizzazione dei calcolatori quantistici apre la strada a nuovi attacchi e nuove costruzioni crittografiche. Si rimanda a [Feh10] per una buona introduzione.

- *Crittografia resistente alle perdite.* L'approccio riduzionista introdotto in questo capito, assume intrinsecamente che l'avversario non abbia alcuna informazione a priori sui segreti memorizzati da Alice ed il Bianconiglio. Esistono attacchi per cui quest'ipotesi non è verificata, in quanto consento di ottenere informazione parziale sui segreti memorizzati. (Tali attacchi, tipicamente, sfruttano il modo in cui una data primitiva è implementata.) Questa informazione parziale, detta "perdita" (*leakage* in inglese), è spesso sufficiente a violare la sicurezza di tanti schemi che altrimenti avrebbero una dimostrazione di sicurezza. È possibile includere questi attacchi nel reame della sicurezza dimostrabile? Si veda ad esempio [MR04].

- *Steganografia.* È possibile comunicare, nascondendo il fatto che la comunicazione [Cac98] (o più in generale un protocollo [AHL05]) stia in effetti avendo luogo?

- *Teoria dei giochi e teoria dei codici.* Il rapporto tra teoria dei giochi/teoria dei codici e crittografia ha generato nuove prospettive in entrambe le discipline [Kat08; TW05].

- *Metodi formali.* I metodi formali comprendono alcune tecniche per lo sviluppo e la verifica dell'hardware e del software. Esse hanno diverse applicazioni nell'analisi della sicurezza di alcuni protocolli crittografici [DY83].

- *Sistemi biometrici.* Esistono sistemi crittografici che si basano sull'utilizzo di alcuni tratti biometrici (ad esempio l'iride dell'occhio oppure le impronte digitali) come chiavi crittografiche. Si veda ad esempio [DRS04; Cra+08].

- *Denaro digitale.* È possibile digitalizzare completamente un sistema monetario? Un sistema di denaro digitale (*e-cash*) prevede che il denaro sia gestito in maniera completamente elettronica. Questo scenario introduce una serie di questioni delicate non presenti in un sistema monetario vero e proprio, in quanto ad esempio bisogna preoccuparsi che una "moneta elettronica" non venga spesa ripetutamente. Si vedano [Cha82; Bra93] per un introduzione.

- *Scenari ad-hoc.* È interessante studiare in modo rigoroso alcuni protocolli di uso comune ad esempio su Internet. Esempi possono trovarsi in IPSec [CK02; Kra03], SSH [BKN04] e Kerberos [BK07].

- *Sicurezza del software.* Ci sono attacchi informatici legati intrinsecamente a come il software è strutturato [CW07].

1.3 Complessità computazionale

In questo paragrafo richiameremo alcuni concetti di base in teoria della complessità computazionale, che si riveleranno un utile strumento per formalizzare le principali nozioni di sicurezza di un sistema crittografico. I lettori interessati ad un approfondimento sono invitati a consultare testi specialistici, ad esempio quello di Arora e Barak [AB08].

Una volta noto che un dato problema è risolvibile, potremmo pensare che non ha importanza se il tempo necessario a risolverlo è 10 o 100 secondi. Tuttavia, questa conclusione non sarebbe così ovvia se la differenza fosse 10 secondi oppure $10^{10^{10}}$ secondi!

La teoria della complessità studia come le risorse necessarie a risolvere un dato problema, scalano con la "dimensione" n del problema stesso: in modo ragionevole (come n oppure n^2), oppure in modo non ragionevole (come 2^n)? Ad esempio, moltiplicare due numeri interi ad n cifre richiede n^2 passi se si utilizza l'algoritmo di moltiplicazione che tutti conosciamo dalle scuole elementari (cf. Esercizio B.1). D'altra parte il problema inverso, ovvero fattorizzare un dato intero ad n cifre, è un problema complesso che (al momento) richiede $\approx 2^{n^{1/3}}$ passi. In informatica teorica, un algoritmo è detto "efficiente" se è eseguibile in un tempo che è limitato superiormente da una funzione polinomiale in n, mentre si parla di algoritmi "inefficienti" quando il tempo di esecuzione è limitato inferiormente da una funzione esponenziale di n.

Lo strumento universale per modellare un algoritmo (implementato in un generico calcolatore) che tenta di risolvere un dato problema, è la cosiddetta *macchina di Turing*. Non daremo una definizione formale, ma cercheremo di coglierne l'essenza. Una macchina di Turing possiede un determinato numero di "nastri" in cui può leggere e/o scrivere simboli appartenenti ad un certo alfabeto: normalmente un nastro è utilizzato per gestire gli input, un nastro per contenere l'output e gli altri nastri sono detti "nastri di lavoro". Solitamente, una macchina di Turing ha associato uno stato che ne determina il comportamento istante per istante: ad ogni istante, lo stato attuale determina

la scrittura/lettura di simboli sui nastri e l'aggiornamento dello stato stesso.[6] Una macchina di Turing è detta *probabilistica* se usa un certo grado di randomicità come parte della sua logica, ovvero se è in grado di compiere scelte casuali. Quando ciò accade, l'output può essere visto come una variabile aleatoria (cf. Appendice A.1). Diremo che una macchina di Turing $\mathscr{M}$ calcola una funzione f in tempo $t(\cdot)$ se, per ogni input x, essa si arresta con il valore $f(x)$ scritto sul nastro di output in al più $t(|x|)$ passi (avendo indicato con $|x|$ la lunghezza dell'input x, cf. Paragrafo 1.4).

Sicurezza computazionale. La nozione di macchina di Turing è il punto di partenza per il concetto di sicurezza computazionale. Ci occorre un po' di notazione:

Definizione 1.1 (Andamenti asintotici). Siano $f, g : \mathbb{N} \to \mathbb{N}$ due funzioni arbitrarie. Diremo che:

(i) $f = O(g)$ se esiste una costante $c > 0$ ed una costante n_0 tali che per ogni $n > n_0$ si abbia $f(n) \leq c \cdot g(n)$. Intuitivamente quindi f è limitata superiormente da g (a meno di un fattore costante);

(ii) $f = \Omega(g)$ se $g = O(f)$, ovvero f è limitata inferiormente da g (a meno di un fattore costante);

(iii) $f = \Theta(g)$ se esistono $c_1, c_2 > 0$ ed una costante n_0, tali che per ogni $n > n_0$ si abbia $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. In altri termini si ha che $f = O(g)$ e $g = O(f)$, ovvero f è limitata sia superiormente che inferiormente da g;

(iv) $f = o(g)$ se *per ogni* $c > 0$ esiste una costante n_0 per cui si abbia $|f(n)| \leq c \cdot |g(n)|$. In altri termini f è dominata asintoticamente da g;

(v) $f = \omega(g)$ se $g = o(f)$, ovvero g è dominata asintoticamente da f. ■

Come abbiamo anticipato nel Paragrafo 1.1, la nozione di sicurezza computazionale rilassa quella di sicurezza incondizionata in due direzioni: (i) si considerano solo avversari limitati computazionalmente e (ii) il sistema può

[6] L'espressività delle macchine di Turing è così potente che esiste una famosa congettura — dovuta allo stesso Turing e a Church — secondo cui se un problema è risolvibile, allora esisterà una macchina di Turing in grado di risolverlo. Non sappiamo se ciò vero oppure no: ad esempio i computer quantistici non possono essere rappresentati dalle macchine di Turing, ma non è chiaro al momento se sia effettivamente possibile costruire un calcolatore quantistico.

essere violato con probabilità molto piccola (tipicamente così piccola che può essere trascurata in pratica). Per quantificare il limite computazionale di un avversario, faremo riferimento alla classe delle macchine di Turing probabilistiche a tempo polinomiale (*Probabilistic Polynomial Time*, PPT).

Definizione 1.2 (Avversari PPT). Diremo che un avversario $\mathscr{A}$ è un avversario PPT, se usa un certo grado di randomicità come parte della sua logica (ovvero $\mathscr{A}$ è probabilistico) e se, per ogni input $x \in \{0,1\}^*$ (l'insieme di stringhe a lunghezza arbitraria), l'esecuzione di $\mathscr{A}(x)$ termina in un numero di passi t polinomiale nella lunghezza della stringa x. (Si intende qui che il numero di passi è $O(|x|^c)$, per una qualche costante c, e si scrive $t = \text{poly}(|x|)$.) ■

Per rendere preciso il fatto che "il sistema può essere violato solamente con probabilità piccola", faremo riferimento al concetto di funzione trascurabile (*negligible* in inglese). Brevemente, una funzione trascurabile tende a zero più velocemente dell'inverso di ogni polinomio:

Definizione 1.3 (Funzione trascurabile). Una funzione $\epsilon : \mathbb{N} \to \mathbb{R}$ è trascurabile, indicato con $\epsilon(n) = \text{negl}(n)$, se per ogni $c > 0$ esiste una costante n_0 tale che per ogni $n > n_0$ si abbia $\epsilon(n) < n^{-c}$. ■

Le funzioni trascurabili e le funzioni polinomiali soddisfano alcune proprietà interessanti (la cui verifica è lasciata come esercizio, cf. Esercizio 1.1):

$$\begin{aligned} \text{poly}(n) + \text{poly}(n) &= \text{poly}(n) & \qquad \text{poly}(n) \cdot \text{poly}(n) &= \text{poly}(n) \\ \text{negl}(n) + \text{negl}(n) &= \text{negl}(n) & \qquad \text{negl}(n) \cdot \text{poly}(n) &= \text{negl}(n). \end{aligned}$$

Un crittosistema computazionalmente sicuro dipende da un parametro statistico di sicurezza n (noto a tutte le parti, attaccanti compresi). La probabilità di successo di un attacco al sistema ed il tempo necessario ad eseguire un attacco sono funzioni di tale parametro. In altri termini abbiamo la seguente:

Definizione 1.4 (Sicurezza computazionale). Siano $t(n), \epsilon(n)$ costanti positive, con $\epsilon < 1$, dipendenti da un parametro di sicurezza $n \in \mathbb{N}$. Un sistema crittografico è (t, ϵ)-sicuro se ogni avversario PPT eseguibile in tempo $t(n)$ che tenta di violare il sistema, ha successo con probabilità al più $\epsilon(n)$ (trascurabile in n). ■

Ovviamente il crittosistema è sicuro solo per valori di n abbastanza grandi. Supponiamo ad esempio che un dato crittosistema sia computazionalmente sicuro, dove nello specifico un attaccante eseguibile in n^4 minuti è in grado

di violare la sicurezza con probabilità $2^{20} \cdot 2^{-n}$. Quando $n \leq 20$, abbiamo che un avversario eseguibile in tempo 20^4 (circa 15 settimane) riesce a violare la sicurezza con probabilità 1 il che è ovviamente inaccettabile. Tuttavia, un valore $n = 300$ assicura che un attaccante che provi ad attaccare il sistema per più di 2000 anni abbia successo solamente con probabilità (circa) 2^{-300}. (Si consideri che un evento che ha probabilità 2^{-60} accade circa una volta ogni cento bilioni di anni...)

Classi di complessità. Più in generale, in teoria della complessità si studiano alcune relazioni tra diverse *classi di complessità*. Una classe di complessità racchiude problemi computazionali con alcune caratteristiche in comune.

Sia $\mathcal{L} \subseteq \{0,1\}^*$ un *linguaggio*, ovvero un insieme arbitrario di stringhe binarie. Diremo che un macchina di Turing $\mathscr{M}$ *decide* un linguaggio $\mathcal{L}$ se è in grado di calcolare una funzione $\varrho_{\mathcal{L}} : \{0,1\}^* \to \{0,1\}$, tale che $\varrho_{\mathcal{L}}(x) = 1$ se e solo se $x \in \mathcal{L}$. In altri termini, una macchina di Turing $\mathscr{M}$ decide $\mathcal{L}$ se e solo se $\mathscr{M}(x) = \varrho_{\mathcal{L}}(x)$ per ogni $x \in \{0,1\}^*$. La classe **P** racchiude tutti i linguaggi *decidibili in tempo polinomiale*, ovvero $\mathcal{L} \in \mathbf{P}$ se esiste una macchina di Turing che decide $\mathcal{L}$ in tempo polinomiale nella lunghezza del suo input. Ad esempio i linguaggi

$$\mathcal{L} = \{(a,b,c) : \ \gcd(a,b) = c\} \qquad \mathcal{L} = \{p : \ p \text{ è un primo}\},$$

sono entrambi in **P** (cf. rispettivamente Appendice B.1 ed Appendice C.1).

L'equivalente probabilistico della classe **P** è la classe **BPP**, ovvero l'insieme di linguaggi decidibili in tempo polinomiale da una macchina di Turing *probabilistica*. La relazione tra **P** e **BPP** non è nota e costituisce un grande problema aperto in informatica teorica. Se **P** fosse uguale a **BPP**, ogni linguaggio decidibile attraverso un algoritmo randomizzato, ammetterebbe anche una soluzione deterministica.

Riassumendo, la classe **P** identifica i problemi che possono essere *risolti* in modo efficiente. La classe **NP** riguarda, invece, i problemi la cui soluzione può essere *verificata* in modo efficiente. Più specificatamente, diremo che un linguaggio $\mathcal{L}$ è in **NP** se esiste una macchina di Turing $\mathscr{M}$ con tempo d'esecuzione polinomiale, tale che per ogni $x \in \{0,1\}^*$:

$$x \in \mathcal{L} \Leftrightarrow \exists\ w \in \{0,1\}^{\mathrm{poly}(|x|)} \text{ tale che } \mathscr{M}(x,w) = 1.$$

L'elemento w è detto *indizio* per x. Il più importante problema aperto in informatica teorica riguarda la relazione tra le classi **P** ed **NP**.[7] Ovviamente $\mathbf{P} \subseteq \mathbf{NP}$, in quanto **P** è il caso estremo in cui w è la stringa vuota. Se **P** fosse uguale ad **NP**, allora la capacità di *verificare* la soluzione di un puzzle in modo efficiente, implicherebbe la capacità di *trovare* una soluzione in modo efficiente. (Per fare un'analogia, è come dire che la capacità di apprezzare un sinfonia implichi la capacità di comporla!)

Siccome un tale scenario non è molto plausibile, diversi scienziati assumo che $\mathbf{P} \neq \mathbf{NP}$, tuttavia una dimostrazione di questo fatto è ben lontana, almeno con gli strumenti a disposizione oggi. Come vedremo più avanti nel testo, la congettura $\mathbf{P} \neq \mathbf{NP}$ è di fondamentale importanza in crittografia.

NP-completezza. La teoria dell'**NP**-completezza studia i linguaggi **NP**-completi. Un linguaggio $\mathcal{L} \subseteq \{0,1\}^*$ è *riducibile* in tempo polinomiale al linguaggio $\mathcal{L}' \subseteq \{0,1\}^*$ se esiste una funzione $f : \{0,1\}^* \to \{0,1\}^*$ calcolabile in modo efficiente tale che per ogni $x \in \{0,1\}^*$, si ha $x \in \mathcal{L}$ se e solo se $f(x) \in \mathcal{L}'$. Il linguaggio $\mathcal{L}'$ è detto **NP**-difficile se ogni altro linguaggio $\mathcal{L}$ in **NP** è riducibile in tempo polinomiale ad $\mathcal{L}'$. Quando $\mathcal{L}'$ è esso stesso in **NP** allora si dice che $\mathcal{L}'$ è **NP**-completo. Intuitivamente i linguaggi **NP**-completi sono i linguaggi *più difficili* in **NP**, in quanto ogni altro linguaggio in **NP** è riducibile ad essi. Esistono più di mille linguaggi **NP**-completi.[8]

1.4 Notazione

Gli insiemi $\mathbb{N}$, $\mathbb{Z}$ ed $\mathbb{R}$ rappresentano (rispettivamente) i numeri naturali, interi e reali. Tutti i numeri rappresentati dalle lettere i, j, k, l, ℓ, m, n sono interi, se non diversamente specificato. Se $n \geq 1$, allora $[n]$ rappresenta l'insieme $\{1, \ldots, n\}$. Dato un numero reale x, scriveremo $\lfloor x \rfloor$ (risp. $\lceil x \rceil$) per il più piccolo intero $n \in \mathbb{Z}$ tale che $n \leq x$ (risp. $n \geq x$). Il valore assoluto di $x \in \mathbb{R}$ — scritto $|x|$ — è x stesso se x è non negativo, e $-x$ se x è negativo. Scriveremo log per il logaritmo in base due ed ln per il logaritmo in base e_{nep} (costante di Neplero, $e_{\text{nep}} \approx 2{,}7$).

Matrici e vettori sono indicati in grassetto. Dato un vettore $\mathbf{r}$, scriveremo $\mathbf{r}[i]$ per l'elemento di $\mathbf{r}$ in posizione i-sima. (Analogamente, se $\mathbf{R}$ è una matrice,

[7] Infatti si tratta di uno dei famosi sette *problemi del millennio* per cui il Clay Mathematical Institut ha messo in palio 1 000 000 $. Per ulteriori informazioni si veda `http://www.claymath.org/millennium/`.

[8] Si veda `http://en.wikipedia.org/wiki/List_of_NP-complete_problems` per qualche esempio.

$\mathbf{R}[i]$ rappresenta l'i-sima colonna di $\mathbf{R}$.) I vettori sono intesi come vettori colonna. L'operatore di trasposizione è indicato con $^{\mathsf{T}}$. Dati due vettori $\mathbf{x}, \mathbf{y}$ il loro prodotto scalare è $\mathbf{x}^{\mathsf{T}} \cdot \mathbf{y}$.

Se $\mathcal{S}$ è un insieme finito, allora una stringa sull'alfabeto $\mathcal{S}$ è un insieme ordinato di elementi di $\mathcal{S}$. Considereremo principalmente stringhe sull'alfabeto binario $\mathcal{S} = \{0,1\} = \mathbb{Z}_2$. Per ogni intero $n \geq 0$, l'insieme $\mathcal{S}^n$ rappresenta l'insieme di stringhe lunghe n sull'alfabeto $\mathcal{S}$. (Per convenzione $\mathcal{S}^0$ è vuoto.) Scriveremo $\mathcal{S}^*$ per l'insieme di tutte le stringhe, ovvero $\mathcal{S}^* = \cup_{n \geq 0} \mathcal{S}^n$. Se x ed y sono stringhe, la loro concatenazione è indicata con $x||y$. Se x è una stringa e $n \geq 1$ un numero naturale, scriveremo x^n per la concatenazione di n copie di x. La lunghezza di x è indicata con $|x|$. Quando $\mathcal{S}$ è l'alfabeto binario, $x \oplus y$ indica lo xor (ovvero la somma binaria) delle stringhe x, y in $\{0,1\}$.

Il carattere corsivo è riservato agli insiemi ed agli eventi. Se $\mathcal{E}$ è un evento, scriveremo $\mathbb{P}[\mathcal{E}]$ per la probabilità associata all'evento $\mathcal{E}$; la negazione di $\mathcal{E}$ è indicata con $\overline{\mathcal{E}}$. Dati due eventi $\mathcal{E}_1$ ed $\mathcal{E}_2$, indicheremo con $\mathcal{E}_1 \wedge \mathcal{E}_2$ la loro unione. Se $\mathcal{X}$ è un insieme, scriveremo $x \in \mathcal{X}$ per un elemento dell'insieme $\mathcal{X}$. La cardinalità di $\mathcal{X}$ è indicata con $\#\mathcal{X}$. Se X è una distribuzione su un insieme $\mathcal{X}$, allora $x \leftarrow \mathsf{X}$ significa che x è scelto casualmente in $\mathcal{X}$ in accordo alla distribuzione X. (Se non diversamente specificato, gli insiemi $\mathcal{X}$ sono associati con la distribuzione uniforme; in questo caso scriveremo esplicitamente $\xleftarrow{\$}$.) Se A è un algoritmo (eventualmente randomizzato), allora $y \leftarrow \mathsf{A}(x;\omega)$ indica un'esecuzione di A con input x, randomicità ω ed output y; in particolare quando A è probabilistico, y è una variabile aleatoria.

Esercizi

Esercizio 1.1. Sia n un parametro statistico di sicurezza. Giustificare formalmente le seguenti espressioni:

$$\begin{aligned} \mathrm{poly}(n) + \mathrm{poly}(n) &= \mathrm{poly}(n) & \mathrm{poly}(n) \cdot \mathrm{poly}(n) &= \mathrm{poly}(n) \\ \mathrm{negl}(n) + \mathrm{negl}(n) &= \mathrm{negl}(n) & \mathrm{negl}(n) \cdot \mathrm{poly}(n) &= \mathrm{negl}(n). \end{aligned}$$

Esercizio 1.2. Spiegare cosa significa che una funzione è $O(1)$.

Esercizio 1.3. Dare una stima delle seguenti funzioni usando la notazione O:

1. $(n^2 + 3n + 8)(n + 1)$,
2. $(3 \log n + 5n^2)(n^3 + 3n + 2)$.

Esercizio 1.4. Siano f, g funzioni arbitrarie. Dimostrare che $f(n) + g(n) = O(\max(f(n), g(n)))$.

Esercizio 1.5. Per ognuna delle seguenti coppie di funzioni $f(n)$ e $g(n)$, stabilire se $f(n) = \Omega(g(n))$ e se $f(n) = \Theta(g(n))$:

1. $f(n) = 6n^2, g(n) = n^2 \log n$,
2. $f(n) = 3/2n^2 + 7n - 4, g(n) = 8n^2$,
3. $f(n) = n^4, g(n) = n^3 \log n$.

Esercizio 1.6. Siano f, g funzioni arbitrarie. Dimostrare che se $f(n) = \Omega(g(n))$, allora $g(n) = O(f(n))$.

Esercizio 1.7. Dimostrare che $2^n = n^{\omega(1)}$.

Esercizio 1.8. Dire se la seguente funzione è trascurabile in n:

$$f(n) = \cos(n) \cdot 2^{-n/4}.$$

Esercizio 1.9. Sia $\epsilon : \mathbb{N} \to \mathbb{R}^+$ una funzione trascurabile. Mostrare che $\epsilon'(n) = \epsilon(n) \cdot \mathrm{poly}(n)$ ed $\epsilon''(n) = \epsilon(\Omega(n^c))$ (per ogni $c > 0$) sono entrambe trascurabili.

Esercizio 1.10. Un algoritmo con complessità $O(f(n))$ e tempo di processamento $t(n) = c \cdot f(n)$, per una certa funzione $f(n)$, impiega 10 secondi a processare 1000 elementi. Quanto tempo impiega a processare 100 000 elementi nei casi $f(n) = n$ ed $f(n) = n^4$?

Esercizio 1.11. Due pacchetti software, A e B, sono scelti per processare basi di dati contenenti ognuno fino a 109 elementi. Il tempo medio di processamento del pacchetto A è $t_A(n) = 0.01 \cdot n$ millisecondi, mentre il tempo di processamento del pacchetto B è $t_B(n) = 500\sqrt{n}$ millisecondi. Quale pacchetto ha prestazioni migliori? Trovare le condizioni esatte in cui un pacchetto è più efficiente dell'altro.

Letture consigliate

[AB08] Sanjeev Arora e Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2008.

[AHL05] Luis von Ahn, Nicholas J. Hopper e John Langford. "Covert two-party computation". In: *STOC*. 2005, pp. 513–522.

[BBK03] Elad Barkan, Eli Biham e Nathan Keller. "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication". In: *CRYPTO*. 2003, pp. 600–616.

[Bel+01] Mihir Bellare, Alexandra Boldyreva, Anand Desai e David Pointcheval. "Key-Privacy in Public-Key Encryption". In: *ASIACRYPT*. 2001, pp. 566–582.

[BK07] Alexandra Boldyreva e Virendra Kumar. "Extended Abstract: Provable-Security Analysis of Authenticated Encryption in Kerberos". In: *IEEE Symposium on Security and Privacy*. 2007, pp. 92–100.

[BKM09] Adam Bender, Jonathan Katz e Ruggero Morselli. "Ring Signatures: Stronger Definitions, and Constructions without Random Oracles". In: *J. Cryptology* 22.1 (2009), pp. 114–138.

[BKN04] Mihir Bellare, Tadayoshi Kohno e Chanathip Namprempre. "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm". In: *ACM Trans. Inf. Syst. Secur.* 7.2 (2004), pp. 206–241.

[BMW03] Mihir Bellare, Daniele Micciancio e Bogdan Warinschi. "Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions". In: *EUROCRYPT*. 2003, pp. 614–629.

[Bra93] Stefan Brands. "Untraceable Off-line Cash in Wallets with Observers (Extended Abstract)". In: *CRYPTO*. 1993, pp. 302–318.

[Cac98] Christian Cachin. "An Information-Theoretic Model for Steganography". In: *Information Hiding*. 1998, pp. 306–318.

[Car71] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1871.

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–88.

[Cha82] David Chaum. "Blind Signatures for Untraceable Payments". In: *CRYPTO*. 1982, pp. 199–203.

[CK02] Ran Canetti e Hugo Krawczyk. "Security Analysis of IKE's Signature-Based Key-Exchange Protocol". In: *CRYPTO*. 2002, pp. 143–161.

[Cra+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró e Daniel Wichs. "Detection of Algebraic Manipulation with Applications to Robust Secret Sharing and Fuzzy Extractors". In: *EUROCRYPT*. 2008, pp. 471–488.

[CW07] Brian Chess e Jacob West. *Secure Programming with Statistic Analysis*. Addison-Wesley, 2007.

[Dam07] Ivan Damgård. "A proof-reading of Some Issues in Cryptography". In: *ICALP*. 2007, pp. 2–11.

[DRS04] Yevgeniy Dodis, Leonid Reyzin e Adam Smith. "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data". In: *EUROCRYPT*. 2004, pp. 523–540.

[DY83] Danny Dolev e Andrew Chi-Chih Yao. "On the security of public key protocols". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–207.

[Feh10] Serge Fehr. "Quantum Cryptography". In: *Foundations of Physics* 40 (2010), pp. 494–531.

[GM82] Shafi Goldwasser e Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *STOC*. 1982, pp. 365–377.

[Gol06] Oded Goldreich. *On Post-Modern Cryptography*. Rapp. tecn. Si veda `http://www.wisdom.weizmann.ac.il/~oded/on-pmc.html`. The Weizmann Institute of Science, 2006.

[Kat08] Jonathan Katz. "Bridging Game Theory and Cryptography: Recent Results and Future Directions". In: *TCC*. 2008, pp. 251–272.

[KM07] Neal Koblitz e Alfred Menezes. "Another Look at "Provable Security"". In: *J. Cryptology* 20.1 (2007), pp. 3–37.

[Kra03] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols". In: *CRYPTO*. 2003, pp. 400–425.

[MR04] Silvio Micali e Leonid Reyzin. "Physically Observable Cryptography (Extended Abstract)". In: *TCC*. 2004, pp. 278–296.

[Sha49] Claude Elwood Shannon. "Communication Theory of Secrecy Systems". In: *Bell Sys. Tech. J.* 28 (1949), pp. 657–715.

[TW05] Wade Trappe e Lawrence C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, 2005.

Parte I

Crittografia di Base

Capitolo 2

Sicurezza incondizionata

Come detto nel Paragrafo 1.1, esistono due accezioni di sicurezza per una primitiva crittografica: sicurezza *incondizionata* e sicurezza *computazionale*. Sebbene, in pratica, il secondo approccio conduca a soluzioni più efficienti, il primo approccio ha il vantaggio di non dover assumere nessuna ipotesi non dimostrata in teoria della complessità (ad esempio il fatto che $\mathbf{P} \neq \mathbf{NP}$). In effetti, come sarà più chiaro in seguito, due ipotesi implicite nell'analisi di ogni schema crittografico sono: (i) che esistano *sorgenti di randomicità* atte a generare segreti casuali utilizzabili da Alice ed il Bianconiglio e (ii) che i segreti generali siano completamente oscuri alla Regina Rossa. Quando si parla di sicurezza computazionale si considerano altre due ipotesi: (iii) che le risorse computazionali a disposizione della Regina siano limitate (nel senso definito nel Paragrafo 1.3) e (iv) che un certo problema computazionale sia difficile, ovvero che esso richieda un tempo enorme per essere risolto dato il limite computazionale di cui al punto (iii). Nel contesto della sicurezza incondizionata non si considerano i punti (iii) e (iv), ovvero si suppone che l'attaccante abbia *risorse computazionali infinite*.

In questo capitolo ci occuperemo quindi di sicurezza incondizionata nel contesto della *confidenzialità* (cf. Paragrafo 1.1). Immaginiamo lo scenario in Fig. 1.1: Alice ed il Bianconiglio vogliono comunicare (scambiando messaggi) su un canale insicuro controllato dalla Regina Rossa, in modo che essa non possa accedere al contenuto dei messaggi inviati sul canale. L'idea chiave è quella di introdurre una trasformazione (reversibile) del messaggio da trasmettere. Alice si occuperà di *cifrare* i messaggi in modo che, qualora la Regina intercetti il testo cifrato (detto anche *crittotesto* o *crittogramma*), non possa ottenere alcuna informazione sul messaggio originale. D'altra parte la trasformazione introdotta deve essere reversibile, nel senso che il Bianconiglio dovrà possedere una qualche informazione aggiuntiva, da usare congiuntamente al crittotesto nel processo cosiddetto di *decifratura*, per invertire la trasformazione iniziale e recuperare il messaggio originale.

Questo problema è stato studiato per la prima volta da Shannon [Sha49]

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_2, © Springer-Verlag Italia 2012

nel contesto delle *tecniche simmetriche*,[9] ovvero assumendo che Alice ed il Bianconiglio abbiano condiviso in precedenza una *chiave segreta*: sarà questa "l'informazione aggiuntiva" che consentirà loro di comunicare mantenendo la confidenzialità.[10] In altri termini ci chiediamo:

> *Supponiamo che Alice ed il Bianconiglio condividano un segreto. Possono comunicare attraverso un canale insicuro, celando il contenuto dei messaggi alla Regina che controlla il canale? Come definire la sicurezza incondizionata in quest'ambito?*

Guida per il lettore. La struttura del capitolo è quindi la seguente. Nel Paragrafo 2.1 daremo la definizione formale di cifrario simmetrico e quindi discuteremo alcune costruzione, per lo più insicure, usate nell'antichità. Nel Paragrafo 2.2 vedremo come definire formalmente la sicurezza incondizionata. Nel Paragrafo 2.3 descriveremo quindi il risultato di Shannon, analizzandone le implicazioni pratiche. Infine, nel Paragrafo 2.4, daremo una panoramica degli sviluppi successivi al lavoro di Shannon, dando alcuni cenni su risultati recenti nel contesto della sicurezza incondizionata.

2.1 Cifrari simmetrici nell'antichità

Come anticipato ne capitolo precedente, l'esigenza della comunicazione segreta era sentita fin dall'antichità, sopratutto in contesto bellico. In questo paragrafo descriveremo alcuni schemi usati nei tempi antichi. Sebbene la maggior parte di essi non sia affatto sicura, il loro studio è importante perché introduce principi fondamentali in crittografia (che torneranno utili anche in seguito). Per una trattazione più dettagliata dei cifrari antichi si veda ad esempio [Sti95, Capitolo 1].

Nel contesto delle tecniche simmetriche, per soddisfare il requisito di confidenzialità si ricorre ad un *cifrario simmetrico*.

Definizione 2.1 (Cifrario simmetrico). Indichiamo con $\mathcal{M}$ lo spazio dei possibili testi in chiaro, con $\mathcal{C}$ lo spazio dei testi cifrati e con $\mathcal{K}$ lo spazio delle chiavi. Un cifrario simmetrico è una tripletta di algoritmi $\Pi_{\mathsf{SKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ definita come segue:

[9] Come vedremo nel Capitolo 6 esistono anche *tecniche asimmetriche*. Purtroppo per esse il concetto di sicurezza incondizionata è intrinsecamente impossibile da enunciare.

[10] Il principio secondo cui un sistema debba rimanere sicuro anche quando il nemico conosce tutto su di esso, a parte la chiave, è stato formulato per la prima volta da Auguste Kerckhoffs nel 1883. Tale principio si può riassumere con il motto "il nemico conosce il sistema".

- `Generazione delle chiavi.` L'algoritmo Gen è l'algoritmo di generazione della chiave segreta. Dato un parametro statistico di sicurezza $n \in \mathbb{N}$, nella forma[11] 1^n (dove n quantificherà la sicurezza del cifrario) restituisce una chiave $k \leftarrow \mathsf{Gen}(1^n)$ in $\mathcal{K}$ che è condivisa tra Alice ed il Bianconiglio.

- `Cifratura.` L'algoritmo $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ è l'algoritmo di cifratura. Dato un messaggio da cifrare $m \in \mathcal{M}$ e la chiave condivisa k calcola il crittotesto $c = \mathsf{Enc}_k(m)$.

- `Decifratura.` L'algoritmo $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ è l'algoritmo di decifratura. Dato un crittotesto c e la chiave condivisa k restituisce il messaggio $m \in \mathcal{M}$ tale che $m = \mathsf{Dec}_k(c)$. ■

Richiederemo che lo schema sia *completo*, nel senso che per ogni n, per ogni chiave $k \leftarrow \mathsf{Gen}(1^n)$ e per ogni messaggio $m \in \mathcal{M}$ si abbia

$$m = \mathsf{Dec}_k(\mathsf{Enc}_k(m)).$$

Siccome di base un cifrario simmetrico cifra stringhe binarie di una data lunghezza, si parla anche di *cifrari a blocco*. Spesso avremo a che fare con algoritmi di cifratura *randomizzati* (ovvero algoritmi che usano una sorgente di randomicità come parte della loro logica). Scriveremo in questo caso $c \leftarrow \mathsf{Enc}_k(m)$.

Cifrari monoalfabetici. Una delle idee più antiche è sicuramente quella di rappresentare le lettere dell'alfabeto usando l'aritmetica modulare (cf. Appendice B.2); in particolare con riferimento alla lingua inglese, i numeri dall'1 al 26 sono sufficienti per rappresentare tutte le lettere dell'alfabeto. Sia allora $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$; per ogni $k, c, m \in \mathbb{Z}_{26}$ definiamo

$$\begin{aligned} c = \mathsf{Enc}_k(m) &= m + k \pmod{26} \\ m = \mathsf{Dec}_k(c) &= c - k \pmod{26}. \end{aligned}$$

In pratica la chiave k trasla tutte le lettere dell'alfabeto in avanti di un certo numero di posizioni.

[11] L'algoritmo riceve come input la stringa 1^n in modo che esso sia eseguibile in tempo polinomiale in n. Per assicurare ciò dobbiamo passare n come input. Invece di passare il valore vero e proprio di n (i.e. $\log n$ bit) si usa esprimere n come una stringa fatta solo di 1, i.e. 1^n. In questo modo tutti gli algoritmi, che sono eseguibili in tempo polinomiale *rispetto alla lunghezza del loro input*, possono essere eseguiti in tempo polinomiale in n.

Un esempio di tale approccio è il *cifrario di Cesare*, in quanto si pensa fosse usato dallo stesso Giulio Cesare nell'antica Roma. Tipicamente Cesare usava uno spostamento di tre posizioni, i.e. $k = 3$. In questo modo le lettere dell'alfabeto sono mappate come segue: $a \to D$, $b \to E$, $c \to F, \ldots, z \to C$. (Indichiamo con le lettere maiuscole le lettere del crittotesto e con le lettere minuscole quelle del messaggio originale.) Supponiamo ad esempio che Cesare voglia incontrare Marco Antonio sulle sponde del fiume Tevere (messaggio: "fiume") o nell'arena (messaggio: "arena"); usando il cifrario di Cesare si ottengono (rispettivamente) i crittotesti "INAPH" e "DRHQD".

Abbiamo già enunciato il *principio di Kerckhoffs*: uno schema non deve basare la sua sicurezza sulla segretezza delle modalità in cui opera, ma solamente sulla segretezza della chiave. Osserviamo che questo apre sempre la strada al cosiddetto *attacco a forza bruta* che consiste nel tentare di "forzare" la sicurezza del sistema provando tutte le chiavi possibili. Segue che, per poter sperare che un cifrario sia sicuro, lo spazio delle chiavi $\mathcal{K}$ non deve essere troppo piccolo. Sulla base di questa osservazione è immediato vedere che il cifrario sopra definito non è affatto sicuro, in quanto $\#\mathcal{K} = 26$ e quindi l'attacco a forza bruta è facilmente eseguibile. (Basta provare tutte le 26 chiavi possibili!) Un altro problema, è che il cifrario è *monoalfabetico*: lettere uguali nel testo in chiaro corrispondono a lettere uguali nel testo cifrato. Ad esempio, se nell'esempio precedente Bruto intercettasse il crittotesto "DRHQD" saprebbe immediatamente che si dovrebbe recare all'arena, senza neanche tentare l'attacco a forza bruta (perché la parola "arena" ha una lettera ripetuta e questa proprietà è invariante nel crittotesto).

Un'estensione immediata, è quella di considerare una qualsiasi permutazione $\pi(\cdot)$ sull'insieme $\mathbb{Z}_{26}$, vale a dire

$$\mathcal{K} = \{\text{permutazioni } \pi(\cdot) \text{ su } \mathbb{Z}_{26}\}\,.$$

Ciò rende l'attacco a forza bruta più oneroso, in quanto ora $\#\mathcal{K} = 26!$. Purtroppo ciò non è sufficiente ad evitare il problema intrinseco di tutti i cifrari monoalfabetici, ovvero l'invarianza delle lettere ripetute. È sufficiente basarsi sulle proprietà statistiche della lingua in cui il messaggio è composto per decifrare senza difficoltà un crittogramma. Ad esempio, nella lingua italiana, le lettere più frequenti sono le vocali "a", "e", "i", "o" ed "u" (con lievi differenze), seguite dalle consonanti "l", "n", "r", "s" e "t", mentre sono rare le lettere "b", "f", "q" e "z" e praticamente assenti le lettere straniere "j", "k", "y", "x" e "w" (cf. Fig. 2.1).

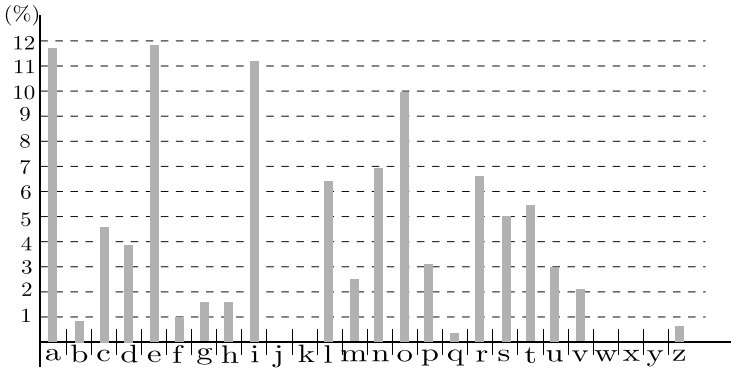


Fig. 2.1. Frequenza lettere lingua italiana

Cifrari polialfabetici. Il modo più naturale per ovviare al difetto principale dei cifrari monoalfabetici è quello di definire un cifrario tale che uno stesso carattere del testo in chiaro possa corrispondere a caratteri diversi nel testo cifrato. Un cifrario con questa proprietà è detto *polialfabetico*. Un primo esempio è dato dal *cifrario vettore-affine*. Sia $\mathcal{M} = \mathcal{C} = \mathbb{Z}_{26}^n$, l'insieme dei vettori lunghi n a valori in $\mathbb{Z}_{26}$. L'insieme delle chiavi è definito come

$$\mathcal{K} = \left\{(\mathbf{A}, \mathbf{b}) : \ \mathbf{A} \in \mathbb{Z}_{26}^{n \times n}, \mathbf{b} \in \mathbb{Z}_{26}^n, \gcd(\det(\mathbf{A}), 26) = 1\right\}.$$

La condizione sul determinante della matrice $\mathbf{A}$ è necessaria e sufficiente affinché la matrice sia invertibile in $\mathbb{Z}_{26}$ (cf. Appendice B.2, Lemma B.9). Data una chiave $\mathbf{A}, \mathbf{b} \in \mathcal{K}$ ed il messaggio $\mathbf{m} \in \mathbb{Z}_{26}^n$ definiamo

$$\begin{aligned} \mathbf{c} = \mathsf{Enc}_k(\mathbf{m}) &= \mathbf{A}^{\mathsf{T}} \cdot \mathbf{m} + \mathbf{b} \pmod{26} \\ \mathbf{m} = \mathsf{Dec}_k(\mathbf{c}) &= (\mathbf{A}^{-1})^{\mathsf{T}} \cdot (\mathbf{c} - \mathbf{b}) \pmod{26}. \end{aligned}$$

Casi particolari dello schema descritto sono il *cifrario di Vigenère* ($\mathbf{A} = \mathbf{I}$, è la matrice identità in $\mathbb{Z}_{26}^{n \times n}$) ed il *cifrario di Hill* ($\mathbf{b} = 0^n$, è il vettore nullo lungo n bit). Qualora la lunghezza del testo in chiaro fosse superiore ad n, è sufficiente dividere il messaggio in blocchi lunghi n elementi. A titolo d'esempio, mostriamo la cifratura del messaggio "arrivano!" con il cifrario di Vigenère. Sia $n = 5$, data la chiave "verme", corrispondente al vettore $\mathbf{b} = (21, 4, 17, 12, 4) \in \mathbb{Z}_{26}^5$ si divide il messaggio in blocchi di n lettere e si applica il cifrario ottenendo la stringa "VVIUZVRF".

Pubblicato nel 1586, il cifrario di Blaise de Vigenère fu ritenuto per secoli inattaccabile, godendo di una fama in buona parte immeritata essendo molto più debole di altri cifrari polialfabetici precedenti. La sua fama è durata per molti anni anche dopo la scoperta del primo metodo di crittanalisi da parte di Charles Babbage, e la successiva formalizzazione da parte del maggiore Friedrich Kasiski: il Metodo Kasiski del 1863.

Il maggiore Kasiski notò che spesso in un crittogramma di Vigénère si possono notare sequenze di caratteri identiche, poste ad una certa distanza fra di loro; questa distanza può, con una certa probabilità, corrispondere alla lunghezza della chiave, oppure ad un suo multiplo. Infatti, essendo il cifrario polialfabetico, la stessa lettera viene cifrata in modo diverso nelle sue varie occorrenze; l'osservazione è che però, se due lettere del testo in chiaro sono poste ad una distanza pari alla lunghezza della chiave (oppure un suo multiplo), questo fa sì che le due lettere vengano cifrate nello stesso modo. Individuando tutte le sequenze ripetute, si può dedurre quasi certamente che la lunghezza della chiave è il massimo comun divisore tra le distanze tra sequenze ripetute, o al più un suo multiplo. Siccome il cifrario di Vigenère non è altro che un insieme di cifrari di Cesare intercalati a distanza fissa, conoscere la lunghezza n della chiave permette di ricondurre il messaggio cifrato ad n messaggi intercalati cifrati con un cifrario di Cesare facilmente decifrabile.

2.2 Cifrari perfetti

Vogliamo ora definire la sicurezza incondizionata per un cifrario simmetrico. Supponiamo che la Regina Rossa intercetti il crittotesto c inviato da Alice. Quale requisito deve soddisfare la trasformazione attuata dal cifrario affinché esso possa dirsi (incondizionatamente) sicuro? Una prima idea è quella di dire che la Regina non deve essere in grado di ricavare la chiave condivisa k (se ciò fosse possibile infatti essa potrebbe immediatamente recuperare m). Purtroppo non è difficile convincersi che tale definizione non è affatto soddisfacente, in quanto esistono funzioni che la soddisfano pur non essendo affatto sicure! Supponiamo ad esempio che $\mathsf{Enc}_k(m) = m$ sia la funzione identità: ovviamente la Regina non può recuperare la chiave, ma ciò non è necessario in quanto il testo cifrato è esattamente il testo in chiaro. Seppur questo esempio sia artificiale, ci fa capire che dobbiamo cercare una definizione diversa.

Un'idea decisamente migliore è quella di richiedere che la Regina non sia in grado di calcolare m a partire da c. Tuttavia, a pensarci bene, ciò non implica che non sia possibile recuperare qualche informazione parziale su m (ad esempio

il primo bit). Inoltre in questo modo non si tiene in considerazione il fatto che la Regina potrebbe avere una qualche informazione a priori sul messaggio m (ad esempio la lingua in cui esso è composto). Per questo motivo richiederemo che l'avversario non sia in grado di ottenere alcuna informazione *addizionale* su m. Formalmente possiamo esprimere questa richiesta attraverso la teoria delle probabilità.

Definizione 2.2 (Cifrario perfetto). Siano M, C, K le variabili aleatorie corrispondenti, rispettivamente, al testo in chiaro m, al crittotesto c ed alla chiave k. Un cifrario simmetrico è incondizionatamente sicuro se, per ogni $m \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$ tale che $\mathbb{P}[C = c] \geq 0$,[12] possiamo scrivere

$$\mathbb{P}[M = m] = \mathbb{P}[M = m \mid C = c].$$

■

In termini intuitivi abbiamo richiesto che l'informazione a priori sul messaggio m rimanga inalterata dopo aver visto il corrispondente crittotesto $c = \mathsf{Enc}_k(m)$. Il seguente lemma fornisce alcune formulazioni equivalenti della Definizione 2.2.

Lemma 2.1 (Nozioni equivalenti di sicurezza incondizionata). *Le seguenti definizioni di sicurezza incondizionata sono equivalenti.*

(i) Per ogni $m \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$, si ha

$$\mathbb{P}[C = c] = \mathbb{P}[C = c \mid M = m].$$

(ii) Per ogni $m \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$, si ha

$$\mathbb{P}[M = m] = \mathbb{P}[M = m \mid C = c].$$

(iii) Per ogni distribuzione su $\mathcal{M}$, per ogni $m_0, m_1 \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$, si ha

$$\mathbb{P}[C = c \mid M = m_0] = \mathbb{P}[C = c \mid M = m_1].$$

[12] Tale requisito è necessario solo per non condizionare rispetto ad un evento con probabilità nulla. Nel seguito supporremo sempre che $\mathbb{P}[M = m]$ e $\mathbb{P}[C = c]$ siano positive. Quanto mostrato può essere generalizzato al caso di distribuzioni arbitrarie su $\mathcal{M}$ e $\mathcal{C}$.

Dimostrazione. Basta mostrare che (i)⇒(ii)⇒(iii)⇒(i).

(i)⇒(ii) Supponiamo che data una distribuzione su $\mathcal{M}$, un messaggio arbitrario $m \in \mathcal{M}$ ed un crittotesto $c \in \mathcal{C}$, risulti

$$\mathbb{P}[C = c \mid M = m] = \mathbb{P}[C = c] .$$

Moltiplicando ambo i membri per $\mathbb{P}[M = m] / \mathbb{P}[C = c]$ otteniamo

$$\frac{\mathbb{P}[C = c \mid M = m] \cdot \mathbb{P}[M = m]}{\mathbb{P}[C = c]} = \mathbb{P}[M = m] .$$

Infine, applicando il teorema di Bayes (cf. Appendice A, Eq. (A.3)) il termine a sinistra dell'uguale coincide con $\mathbb{P}[M = m|C = c]$, da cui l'asserto.

(ii)⇒(iii) Supponiamo che risulti $\mathbb{P}[M = m|C = c] = \mathbb{P}[M = m]$. È immediato accorgersi che il teorema di Bayes implica $\mathbb{P}[C = c|M = m] = \mathbb{P}[C = c]$. Moltiplicando ambo i membri per $\mathbb{P}[C = c] / \mathbb{P}[M = m]$ otteniamo

$$\frac{\mathbb{P}[M = m \mid C = c] \cdot \mathbb{P}[C = c]}{\mathbb{P}[M = m]} = \mathbb{P}[C = c] .$$

Quindi, applicando nuovamente il teorema di Bayes, il termine a sinistra dell'uguale coincide con $\mathbb{P}[C = c|M = m]$ come desiderato. Una volta mostrato ciò, l'asserto segue dal punto precedente della dimostrazione e dal fatto che $m_0, m_1 \in \mathcal{M}$ sono arbitrari, ovvero

$$\mathbb{P}[C = c \mid M = m_0] = \mathbb{P}[C = c] = \mathbb{P}[C = c \mid M = m_1] .$$

(iii)⇒(i) Supponiamo che per ogni distribuzione su $\mathcal{M}$, ogni $m_0, m_1 \in \mathcal{M}$ ed ogni $c \in \mathcal{C}$ risulti $\mathbb{P}[C = c|M = m_0] = \mathbb{P}[C = c|M = m_1]$. Fissiamo una distribuzione su $\mathcal{M}$, un messaggio $m_0 \in \mathcal{M}$ ed un crittotesto $c \in \mathcal{C}$ e definiamo $p_0 = \mathbb{P}[C = c|M = m_0]$. Abbiamo

$$\begin{aligned}
\mathbb{P}[C = c] &= \sum_{m \in \mathcal{M}} \mathbb{P}[C = c \mid M = m] \cdot \mathbb{P}[M = m] \\
&= [m_0 \text{ è arbitrario}] \\
&= \sum_{m \in \mathcal{M}} p_0 \cdot \mathbb{P}[M = m] \\
&= p_0 \cdot \sum_{m \in \mathcal{M}} \mathbb{P}[M = m] \\
&= p_0 = \mathbb{P}[C = c \mid M = m_0] .
\end{aligned}$$

Siccome m_0 è arbitrario abbiamo mostrato $\mathbb{P}[C = c] = \mathbb{P}[C = c|M = m]$ per ogni $c \in \mathcal{C}$ e per ogni $m \in \mathcal{M}$. L'asserto segue quindi dal punto (i) della dimostrazione. □

2.3 Il risultato di Shannon

Claude Shannon è stato il primo a caratterizzare i cifrari incondizionatamente sicuri. L'osservazione di partenza riguarda la dimensione dello spazio delle chiavi:

Lemma 2.2 (Dimensione di $\mathcal{K}$). *Se un cifrario è a sicurezza incondizionata, allora $\#\mathcal{K} \geq \#\mathcal{M}$.*

Dimostrazione. Mostriamo che se $\#\mathcal{K} < \#\mathcal{M}$, il cifrario non può essere incondizionatamente sicuro. Sia $\#\mathcal{K} < \#\mathcal{M}$. Consideriamo la distribuzione uniforme su $\mathcal{M}$ e sia $c \in \mathcal{C}$ un possibile crittotesto (vale a dire $\mathbb{P}[C = c] > 0$). Definiamo l'insieme $\mathcal{M}(c)$ dei possibili messaggi che possono essere una decifratura di c:

$$\mathcal{M}(c) = \{m : \ m = \mathsf{Dec}_k(c) \text{ per qualche } k \in \mathcal{K}\}.$$

Ovviamente $\#\mathcal{M}(c) \leq \#\mathcal{K}$, in quanto per ogni messaggio $m \in \mathcal{M}(c)$ esiste *almeno* una chiave $k \in \mathcal{K}$ tale che $m = \mathsf{Dec}_k(c)$. Ma poiché $\#\mathcal{K} < \#\mathcal{M}$, ciò implica che esiste un messaggio m' tale che $m' \in \mathcal{M}$ ma $m' \notin \mathcal{M}(c)$. Quindi

$$\mathbb{P}[M = m' \mid C = c] = 0 \neq \mathbb{P}[M = m'].$$

ed il cifrario non è a segretezza perfetta. □

D'altra parte è immediato accorgersi che $\#\mathcal{C} \geq \#\mathcal{M}$, altrimenti per ogni chiave ci possono essere due testi in chiaro con cifratura identica (il che rende impossibile decifrare in modo non ambiguo). Ne segue che il caso $\#\mathcal{M} = \#\mathcal{K} = \#\mathcal{C}$ è il caso *ottimo*. Il teorema di Shannon individua le condizioni necessarie e sufficienti per avere sicurezza incondizionata in questo caso.

Teorema 2.3 (Teorema di Shannon). *Sia $\#\mathcal{M} = \#\mathcal{K} = \#\mathcal{C}$. Un cifrario è a segretezza perfetta se e solo se:*

(i) ciascuna chiave $k \in \mathcal{K}$ è scelta uniformemente con probabilità $1/(\#\mathcal{K})$;

(ii) per ogni messaggio $m \in \mathcal{M}$ e per ogni crittotesto $c \in \mathcal{C}$ esiste una ed una sola chiave k tale che $c = \mathsf{Enc}_k(m)$.

Dimostrazione. ($\Rightarrow$) Supponiamo che il cifrario sia perfetto. Non è difficile convincersi che per ogni $m \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$ esiste *almeno* una chiave $k \in \mathcal{K}$ tale che $\mathsf{Enc}_k(m) = c$.[13] Consideriamo l'insieme $\{\mathsf{Enc}_k(m)\}_{k\in\mathcal{K}}$: per quanto detto $\#\{\mathsf{Enc}_k(m)\}_{k\in\mathcal{K}} \geq \#\mathcal{C}$. Siccome in generale $\#\{\mathsf{Enc}_k(m)\}_{k\in\mathcal{K}} \leq \#\mathcal{C}$, possiamo concludere

$$\#\{\mathsf{Enc}_k(m)\}_{k\in\mathcal{K}} = \#\mathcal{C}.$$

Poiché $\#\mathcal{K} = \#\mathcal{C}$, ne segue $\#\{\mathsf{Enc}_k(m)\}_{k\in\mathcal{K}} = \#\mathcal{K}$. Ma allora non possono esistere due chiavi $k_1, k_2 \in \mathcal{K}$ *distinte* tali che $\mathsf{Enc}_{k_1}(m) = \mathsf{Enc}_{k_2}(m)$. Ciò implica che per ogni m e per ogni c c'è *al più* una chiave k tale che $\mathsf{Enc}_k(m) = c$. Poiché però abbiamo già detto che ne deve anche esistere *almeno* una, possiamo concludere che ne esiste *una ed una sola*, che è il punto (ii) dell'asserto.

Resta da mostrare che per ogni $k \in \mathcal{K}$ si ha $\mathbb{P}[K = k] = 1/(\#\mathcal{K})$. Sia $\#\mathcal{K} = \ell$, possiamo scrivere $\mathcal{M} = \{m_1, \ldots, m_\ell\}$. Siccome abbiamo appena mostrato che per ogni m_i e per ogni c esiste un'unica chiave tale che c è la cifratura di m_i, possiamo etichettare le chiavi in modo che $\mathsf{Enc}_{k_i}(m_i) = c$ per ogni $1 \leq i \leq \ell$. Usando la definizione di segretezza perfetta possiamo scrivere:

$$\begin{aligned}\mathbb{P}[M = m_i] = \mathbb{P}[M = m_i \mid C = c] &\overset{(*)}{=} \frac{\mathbb{P}[C = c \mid M = m_i]\,\mathbb{P}[M = m_i]}{\mathbb{P}[C = c]} \\ &\overset{(**)}{=} \frac{\mathbb{P}[K = k_i]\,\mathbb{P}[M = m_i]}{\mathbb{P}[C = c]},\end{aligned}$$

dove la $(*)$ deriva dal teorema di Bayes, mentre la $(**)$ deriva dal fatto che k_i è l'unica chiave che mappa m_i in c. Ma allora, per ogni $i = 1, \ldots, \ell$

$$\mathbb{P}[K = k_i] = \mathbb{P}[C = c],$$

e quindi tutte le chiavi sono scelte *con la stessa probabilità*. Ne segue $\mathbb{P}[K = k_i] = 1/(\#\mathcal{K})$, come richiesto dal punto (i) dell'asserto.

($\Leftarrow$) Supponiamo che ogni chiave sia scelta con uguale probabilità $1/(\#\mathcal{K})$ e che per ogni $m \in \mathcal{M}$ e $c \in \mathcal{C}$ esista un'unica chiave $k \in \mathcal{K}$ tale per cui $\mathsf{Enc}_k(m) = c$. Allora per ogni m e per ogni c

$$\mathbb{P}[C = c \mid M = m] = \frac{1}{\#\mathcal{K}},$$

[13] L'argomento è simile alla dimostrazione del Lemma 2.2. Se così non fosse

$$\mathbb{P}[M = m \mid C = c] = 0 \neq \mathbb{P}[M = m],$$

ed il cifrario non è perfetto.

Crittosistema 2.1. Il blocco monouso Π_{OTP}

Sia $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$.

- `Generazione delle chiavi.` Genera una chiave casuale $k \xleftarrow{\$} \{0,1\}^n$ condivisa tra Alice ed il Bianconiglio.
- `Cifratura.` Per cifrare il messaggio $m \in \{0,1\}^n$ si pone:

$$c = \mathsf{Enc}_k(m) = k \oplus m.$$

- `Decifratura.` Dato un crittotesto $c \in \{0,1\}^n$ e la chiave k, calcola:

$$m = \mathsf{Dec}_k(c) = c \oplus k.$$

indipendentemente dalla distribuzione di M su $\mathcal{M}$. Quindi per ogni distribuzione su $\mathcal{M}$, per ogni coppia di messaggi $m_0, m_1 \in \mathcal{M}$ e per ogni $c \in \mathcal{C}$ abbiamo

$$\mathsf{IP}\left[C = c \mid M = m_0\right] = \frac{1}{\#\mathcal{K}} = \mathsf{IP}\left[C = c \mid M = m_1\right],$$

e l'asserto segue dal Lemma 2.1. □

Blocco monouso. Nel 1917 Vernam ha brevettato un cifrario detto blocco monouso (*One-Time Pad*, OTP). Come vedremo, in luce del Teorema di Shannon, tale cifrario ha sicurezza incondizionata. Il cifrario è mostrato nel Crittosistema 2.1. È banale verificare che la decifratura inverte la cifratura. L'intuizione per cui Π_{OTP} è a sicurezza incondizionata è che per ogni possibile m esiste una sola chiave k tale che $c = \mathsf{Enc}_k(m)$: la chiave $k = m \oplus c$. Siccome ogni chiave è scelta uniformemente (e tenuta segreta), nessuna chiave è più probabile delle altre e quindi c non rivela alcuna informazione su m. Più formalmente:

Corollario 2.4 (Il blocco monouso è un cifrario perfetto). Π_{OTP} *è incondizionatamente sicuro.*

Dimostrazione. L'asserto segue direttamente dal teorema di Shannon. □

Commentiamo alcune limitazioni pratiche relative all'uso del blocco monouso (e di ogni cifrario a sicurezza perfetta caratterizzato dal teorema di Shannon).

Prima di tutto *è necessario che la chiave sia tanto lungo quanto il testo in chiaro.* Ciò non consente l'uso di Π_{OTP} quando si devono cifrare messaggi troppo lunghi (in quanto bisogna memorizzare e distribuire in modo sicuro una quantità non indifferente di dati) né quando non si conosce in anticipo la lunghezza dei messaggi da cifrare (in quanto non si sa quanto deve essere lunga la chiave da condividere). Inoltre lo schema è sicuro *solo se utilizzato una sola volta (con una data chiave).* Infatti se si usa la stessa chiave k per cifrare i messaggi $m, m' \in \mathcal{M}$ ottenendo $c = k \oplus m$ e $c' = k \oplus m'$, un attaccante può calcolare

$$c \oplus c' = (k \oplus m) \oplus (k \oplus m') = m \oplus m',$$

contro l'ipotesi che il cifrario abbia sicurezza perfetta.[14]

2.4 Sicurezza incondizionata?

Volendo riassumere il risultato di Shannon, esso ci dice che sebbene la nozione di sicurezza incondizionata sia molto più forte di quella di sicurezza computazionale, essa è meno pratica. Il fatto che Shannon abbia dimostrato che per ottenere un cifrario perfetto bisogna necessariamente utilizzare una chiave lunga quanto il testo in chiaro è spesso usato come evidenza che la sicurezza incondizionata non è utilizzabile in pratica.

Tuttavia questa conclusione è affrettata. Infatti mentre il Teorema 2.3 è valido nel modello della realtà definito da Shannon, la crittografia è usata nel mondo reale dove non sempre è possibile avere informazione completa sullo stato del sistema (ad esempio a causa del rumore o delle leggi d'indeterminismo tipiche della meccanica quantistica). La conoscenza imperfetta dello stato del sistema può essere sfruttata per ottenere sicurezza incondizionata con migliore efficienza. In effetti, sfruttando tale ipotesi, il lavoro di Shannon è stato esteso in diversi aspetti, sia per quanto riguarda la confidenzialità [GMS74; WC81; Sim92; Sti94; Mau96; Mau97; MW99] e l'integrità di messaggio, sia per quanto riguarda il problema dello scambio delle chiavi. Si vedano anche [Wol98; Mau99] per una panoramica.

Il modello dello spazio di memoria limitato. I due parametri fondamentali che specificano le risorse dell'avversario sono la sua capacità computazionale (come definito nel Paragrafo 1.3) e la quantità di dati che esso è

[14] Usando le proprietà statistiche della lingua in cui il testo è scritto (se i messaggi sono sufficientemente lunghi) ciò consente di recuperare (con elevata probabilità) il contenuto dei messaggi stessi.

in grado di memorizzare (ovvero lo *spazio di memoria* a sua disposizione). Mentre l'approccio della sicurezza computazionale limita il potere computazionale a disposizione dell'avversario, il modello dello spazio di memoria limitato (*Bounded-Storage Model*, BSM), introdotto da Maurer [Mau92] ed analizzato da Dziembowski e Maurer [DM02; DM04], ipotizza che lo spazio di memoria a disposizione dell'avversario sia limitato senza fare alcuna ipotesi sul suo potere computazionale.

Tipicamente una primitiva nel BSM utilizza una certa informazione ausiliaria ω detta il *randomizzatore*.[15] Indichiamo con s il limite di memoria dell'attaccante: se ω è lunga r bit, si richiede che $r > s$ affinché l'avversario non sia in grado di memorizzare ω. Alice ed il Bianconiglio usano quindi una chiave condivisa $k \in \{0,1\}^n$ ed il randomizzatore ω per derivare una nuova chiave k^* a lunghezza $n^* > n$ (da usare ad esempio in un blocco monouso). Il requisito è che la distribuzione di probabilità della variabile aleatoria K^* (corrispondente alla chiave k^*) sia "praticamente uniforme", nonostante l'attaccante conosca r bit del randomizzatore ω (o più in generale di una qualsiasi funzione applicata ad ω per ricavare r bit). Il rapporto $\nu = s/r$ è detto efficienza di randomicità dello schema. Ovviamente (per essere realistici) i valori s (e di conseguenza r) non devono essere troppo piccoli. La sfida principale in questo modello è quella di progettare uno schema sicuro con r non troppo più grande di s.

Il modello della capacità limitata. Un'idea per certi versi simile è quella del modello della capacità limitata (*Bounded-Retrieval Model*, BRM) introdotto da Dziembowski [Dzi06a; Dzi06b; Cas+07]. Lo scopo è quello di progettare protocolli che siano sicuri anche quando eseguiti su una macchina sotto il controllo parziale dell'avversario (ad esempio attraverso l'uso di un virus[16] o di un cavallo di troia). Si considera quindi uno scenario in cui la macchina su cui una data primitiva è eseguita può essere sotto il controllo dell'avversario. Ovviamente se la macchina è totalmente controllata dall'attaccante non c'è speranza di ottenere alcuna forma di sicurezza. Pertanto si suppone che esistano intervalli di tempo in cui la macchina è esente da virus.

Se i segreti memorizzati sulla macchina sono troppo corti, questi possono essere facilmente recuperati dall'attaccante quando la macchina è sotto il suo controllo, per essere utilizzati anche quando il controllo della macchina è perso.

[15] Tale informazione può essere ad esempio una sequenza casuale di bit trasmessa in radiodiffusione circolare (*broadcast* in inglese) da satellite.

[16] I virus ed i cavalli di troia informatici sono dei codici maligni (*malware*) installati su un sistema per recare danni e/o prelevare informazioni.

Per prevenire questo tipo di attacco si rende il recupero infattibile facendo crescere arbitrariamente la dimensione dei segreti memorizzati ed assumendo che la capacità di recupero di informazione da parte dell'attaccante sia limitata. Quest'ultima ipotesi è perfettamente ragionevole ad esempio nel contesto di Internet, in cui il limite è rappresentato dalla capacità massima del mezzo fisico che trasporta l'informazione. La sfida consiste nel costruire protocolli efficienti, nonostante la dimensione della chiave. Ciò è fatto tipicamente richiedendo che il sistema debba accedere solamente ad una piccola porzione del segreto (opportunamente selezionata) quando la primitiva è invocata.

Esercizi

Esercizio 2.1. Il seguente testo cifrato è stato ottenuto usando un cifrario monoalfabetico a sostituzione:

> LXVKJCCR BXUX UN PDNAAN LQN YDXR ERWLNAN... YANYJAJCR YNA UN PDNAAN LQN MNER LXVKJCCNAN — MNUUJ PDNAAJ, LJAU EXW LUJDBNFRCI

Determinare il corrispondente testo in chiaro.

Esercizio 2.2. Il seguente testo cifrato è stato ottenuto usando un cifrario monoalfabetico a permutazione:

> WSEKP HPWWS NUPEES IY YDGPNDS S IVDAYHSEP DVD GVWKSDKV DPWWS QEVJSJYWYK ILP YW DPTYIV DVD GY QEPGPDKY, TS GUWWS DVGKES QEPQSESXYVDP S EYIPOPEWV; DVD GVWKSDKV GUWWS QVGGYJYWYK ILP DVD SKKSIILY, TS QYUKKVGKV GUWWSOPEP EPGV WP DVGKEP QVGYXYVDY YTQEPDHYJYWY — WSEKP HPWWS NUPEES, GUD KXU

Determinare il corrispondente testo in chiaro. (Notare che la permutazione comprende anche le lettere non tipicamente italiane, ovvero "j", "k", "w", "x", "y".)

Esercizio 2.3. Il seguente testo cifrato è stato ottenuto usando il cifrario di Vigénere:

> FQB DASWTS HT PJKAN I PJQJZZE DG QNAE FWCUM E LCBSWMWPC K'CLLKAN I MWVHDZE VC DZZTW SIDTLW XSBKHAG — OMBIUQ DQWVWTPHW SUQNYMSW

Determinare la chiave usando il metodo Kasiski e quindi recuperare il testo in chiaro.

Esercizio 2.4. Il messaggio binario $\mathbf{m} = (100110110110)$ viene cifrato usando un cifrario di Hill su $\mathbb{Z}_4$. La chiave è costituita da una matrice $\mathbf{A}$ con dimensione 2×2.

1. Quali condizioni deve rispettare $\mathbf{A}$ in generale?
2. Supponendo $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 3 & 3 \end{pmatrix}$, calcolare il crittotesto $\mathbf{c}$ corrispondente ad $\mathbf{m}$ e decifrare il risultato.

3. Data la coppia $(\mathbf{m}, \mathbf{c})$, attaccare il cifrario e recuperare la chiave.

Esercizio 2.5. Supponiamo di definire un cifrario (simmetrico) $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ "sicuro" se nessun attaccante $\mathscr{A}$ è in grado di recuperare la chiave a partire dal crittotesto. In altri termini, per ogni $\mathscr{A}$, per ogni messaggio m

$$\mathbb{P}\left[\mathscr{A}(c) = k : \ k \xleftarrow{\$} \mathcal{K}, m \xleftarrow{\$} \mathcal{M}, c \leftarrow \mathsf{Enc}_k(m)\right] \leq \frac{1}{|\mathcal{K}|}.$$

Dimostrare che questa condizione non è né necessaria né sufficiente per ottenere sicurezza incondizionata.

Esercizio 2.6. Consideriamo la seguente nozione di sicurezza perfetta per la cifratura di due messaggi. Un cifrario simmetrico $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ ha sicurezza perfetta per la cifratura di due messaggi, se per ogni distribuzione sullo spazio dei messaggi $\mathcal{M}$, per ogni $m_1, m_2 \in \mathcal{M}$ e per ogni $c_1, c_2 \in \mathcal{C}$ tali che $\mathbb{P}[C_1 = c_1 \wedge C_2 = c_2] > 0$, si ha

$$\mathbb{P}[M_1 = m_1 \wedge M_2 = m_2 \mid C_1 = c_1 \wedge C_2 = c_2] = \mathbb{P}[M_1 = m_1 \wedge M_2 = m_2].$$

Dimostrare che nessun cifrario simmetrico può soddisfare questa nozione.

Esercizio 2.7. Consideriamo la seguente definizione di sicurezza perfetta. Un cifrario simmetrico ha sicurezza perfetta se per ogni $m_0, m_1 \in \mathcal{M}$ e per ogni avversario $\mathscr{A}$, si ha

$$\mathbb{P}\left[\mathscr{A}(c) = 1 : \ k \xleftarrow{\$} \mathcal{K}, c = \mathsf{Enc}_k(m_0)\right] = \mathbb{P}\left[\mathscr{A}(c) = 1 : \ k \xleftarrow{\$} \mathcal{K}, c = \mathsf{Enc}_k(m_1)\right].$$

Dimostrare che questa definizione è equivalente alla Definizione 2.2.

Esercizio 2.8. Supponiamo che un cifrario $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ soddisfi la seguente proprietà di "correttezza debole". Per ogni messaggio $m \in \mathcal{M}$, abbiamo

$$\mathbb{P}\left[\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m : \ k \xleftarrow{\$} \mathcal{K}\right] \geq 1 - 2^{-r},$$

essendo r una costante. Mostrare che un tale cifrario può essere incondizionatamente sicuro anche se $\#\mathcal{K} < \#\mathcal{M}$. Calcolare un limite inferiore per la dimensione di $\mathcal{K}$.

Esercizio 2.9. Dare una dimostrazione diretta del fatto che OTP ha segretezza perfetta.

Esercizio 2.10. Notare che quando il cifrario OTP viene utilizzato con chiave $k = \{0,1\}^n$, ogni messaggio $m \in \{0,1\}^n$ è cifrato in $c = \mathsf{Enc}_k(m) = 0^n \oplus m = m$. Dobbiamo concludere che OTP non deve mai essere utilizzato con chiave nulla? Spiegare perché questo fatto non contraddice la sicurezza incondizionata di OTP.

Letture consigliate

[Cas+07] David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard J. Lipton e Shabsi Walfish. "Intrusion-Resilient Key Exchange in the Bounded Retrieval Model". In: *TCC*. 2007, pp. 479–498.

[DM02] Stefan Dziembowski e Ueli M. Maurer. "Tight Security Proofs for the Bounded-Storage Model". In: *STOC*. 2002, pp. 341–350.

[DM04] Stefan Dziembowski e Ueli M. Maurer. "On Generating the Initial Key in the Bounded-Storage Model". In: *EUROCRYPT*. 2004, pp. 126–137.

[Dzi06a] Stefan Dziembowski. "Intrusion-Resilience Via the Bounded-Storage Model". In: *TCC*. 2006, pp. 207–224.

[Dzi06b] Stefan Dziembowski. "On Forward-Secure Storage". In: *CRYPTO*. 2006, pp. 251–270.

[GMS74] Edgar N. Gilbert, F. Jessie MacWilliams e Neil J. A. Sloane. "Codes which Detect Deception". In: *Bell System Tech. J.* 53 (1974), pp. 405–424.

[Mau92] Ueli M. Maurer. "Conditionally-Perfect Secrecy and a Provably-Secure Randomized Cipher". In: *J. Cryptology* 5.1 (1992), pp. 53–66.

[Mau96] Ueli M. Maurer. "A Unified and Generalized Treatment of Authentification Theory". In: *STACS*. 1996, pp. 387–398.

[Mau97] Ueli M. Maurer. "Information-Theoretically Secure Secret-Key Agreement by NOT Authenticated Public Discussion". In: *EUROCRYPT*. 1997, pp. 209–225.

[Mau99] Ueli M. Maurer. "Information-Theoretic Cryptography". In: *CRYPTO*. 1999, pp. 47–64.

[MW99] Ueli M. Maurer e Stefan Wolf. "Unconditionally Secure Key Agreement and the Intrinsic Conditional Information". In: *IEEE Transactions on Information Theory* 45.2 (1999), pp. 499–514.

[Sha49] Claude Elwood Shannon. "Communication Theory of Secrecy Systems". In: *Bell Sys. Tech. J.* 28 (1949), pp. 657–715.

[Sim92] Gustavus J. Simmons. "A Survey of Information Authentication". In: *Contemporary Cryptology, The Science of Information Integrity*. IEEE Press, 1992, pp. 379–419.

[Sti94] Douglas R. Stinson. "Universal Hashing and Authentication Codes". In: *Des. Codes Cryptography* 4.4 (1994), pp. 369–380.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

[WC81] Mark N. Wegman e J. Lawrence Carter. "New Hash Functions and Their Use in Authentication and Set Equality". In: *JCSS* 22 (1981), pp. 265–279.

[Wol98] Stefan Wolf. “Unconditional Security in Cryptography”. In: *Lectures on Data Security*. 1998, pp. 217–250.

Capitolo 3

Randomicità e pseudorandomicità

La generazione di numeri casuali è troppo importante per essere lasciata al caso.

Robert Coveyou, ORNL

Il termine *randomicità* assume significati disparati a seconda del contesto in cui è utilizzato. Letteralmente si individua con *casuale* un fenomeno che non ha uno scopo preciso, che accade in modo incontrollato ed impredicibile. Se in natura esistano o meno sorgenti di vera randomicità, è un problema ampiamente discusso da scienziati e filosofi. (Tuttavia non ci soffermeremo su questo aspetto.)

Dal punto di vista informatico, possiamo evidenziare il ruolo della randomicità attraverso il seguente esempio. Il Cappellaio Matto organizza una cena per festeggiare il "non-compleanno" di Alice. Siccome si dà il caso che il non-compleanno di Alice coincida con quello del Bianconiglio alla festa sono presenti gli amici di entrambi. Possiamo dividere gli invitati in due gruppi: persone che si conoscono e persone che non si conoscono. Il Cappellaio Matto decide di apparecchiare due tavoli e vorrebbe sistemare gli invitati in modo che, alla fine, il numero di coppie di persone che non si conoscono e siedono in due tavoli diversi sia massimo. Come fare?

Il problema è più complesso di quanto sembra in apparenza. Ad esempio la soluzione banale di provare tutte le possibili configurazioni per gli ospiti e scegliere la migliore, richiede troppo tempo per essere attuata. Ecco che la randomicità viene in aiuto del Cappellaio Matto: non è complesso dimostrare che se questi assegna i posti agli invitati *casualmente*, la configurazione finale è tale che il numero di coppie di persone che non si conoscono e siedono in due tavoli diversi è almeno la metà del numero massimo (cf. Esercizio 3.1).

La randomicità ha un ruolo apparentemente fondamentale in tantissimi altri contesti, ben oltre l'informatica teorica (ad esempio nella genetica o nelle leggi fisiche). Una delle domande fondamentali in teoria della complessità è proprio

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_3,

il rapporto tra (le classi di) problemi che possono essere risolti senza l'uso di alcuna randomicità (cosiddetti *deterministici*) e (le classi di) problemi che invece necessitano di randomicità per essere risolti. Ad esempio per il problema degli invitati descritto sopra, esiste un algoritmo deterministico che conduce alla stessa soluzione: basta mettere ogni nuovo ospite al tavolo opposto a quello dove sono presenti più persone che non si conoscono. In effetti uno dei più grandi problemi aperti in teoria della complessità è se *ogni* problema risolvibile in modo efficiente con uso di randomicità è risolvibile in modo efficiente anche senza (ovvero il rapporto tra le classi **P** e **BPP**, cf. Paragrafo 1.3).

Come avremo ampiamente modo di vedere, anche in crittografia la randomicità ha un ruolo fondamentale. In questo capitolo ci occupiamo del problema di base della *generazione di numeri casuali*. Esistono principalmente due modi per generare randomicità: sfruttando alcuni *fenomeni fisici*, oppure usando *metodi computazionali*. Nel secondo caso ci si accontenta di generare sequenze *pseudocasuali*: sebbene il risultato non sia veramente casuale, esso "approssima" abbastanza bene una sequenza puramente casuale in tutti i contesti pratici. Quest'approccio solitamente conduce a costruzioni molto efficienti.

Nel primo caso si cerca di utilizzare l'impredicibilità intrinseca di alcuni fenomeni fisici (ad esempio il rumore termico oppure le sequenze di lettura/scrittura nel disco rigido di un calcolatore). In realtà si può vedere che anche l'informazione generata utilizzando queste sorgenti non è perfettamente casuale, ma presenta un qualche grado di correlazione. Per ovviare a questo problema si è soliti definire una procedura in grado di "estrarre" tutta la randomicità vera contenuta nella sequenza prodotta (si parla di *estrattori di randomicità*). Purtroppo l'efficienza di queste costruzioni è molto più bassa se paragonata ai generatori pseudocasuali.

Guida per il lettore. Nel Paragrafo 3.1 introdurremo l'importante concetto di *indistinguibilità computazionale* e vedremo come esso può essere utilizzato per definire formalmente un generatore pseudocasuale. Quindi nel Paragrafo 3.2 studieremo il concetto di *impredicibilità*, collegandolo a quello di pseudocasualità. Nei Paragrafi 3.3 e 3.4, quindi, cominceremo uno studio sistematico delle tecniche per la realizzazione dei generatori pseudocasuali, a partire da costruzioni più teoriche, fino a considerare alcuni schemi utilizzati in pratica. Concluderemo con una breve panoramica sugli estrattori di randomicità (nel Paragrafo 3.5).

3.1 Indistinguibilità ed argomento ibrido

Un modo naturale per esprimere formalmente il concetto di pseudorandomicità è attraverso l'uso del concetto di *indistinguibilità*. Informalmente diremo che una sequenza di valori è pseudocasuale se è impossibile (o comunque difficile) distinguerla da una sequenza di valori veramente casuale. Come vedremo il paradigma dell'indistinguibilità è usato in svariati contesti in crittografia.

Per formalizzare quest'intuizione useremo *insiemi di distribuzioni*, ovvero sequenze infinite di distribuzioni di probabilità.[17] Più formalmente, un insieme di distribuzioni $X = \{X_n\}_{n\in\mathbb{N}}$ è una collezione di variabili aleatorie $X_1, X_2, \ldots$. In concreto, la variabile aleatoria X_n potrebbe corrispondere all'output di una primitiva crittografica quando il parametro di sicurezza è n. Diremo che l'insieme di distribuzioni X è *efficientemente campionabile* se esiste un algoritmo efficiente che dato n preleva X_n da X.

Definizione 3.1 (Indistinguibilità). Sia $n \in \mathbb{N}$ un parametro statistico di sicurezza ed indichiamo con $X = \{X_n\}_{n\in\mathbb{N}}$ ed $Y = \{Y_n\}_{n\in\mathbb{N}}$ due insiemi di distribuzioni di probabilità efficientemente campionabili. Diremo che:

(i) X ed Y sono *identicamente distribuiti*, indicato con $X \stackrel{d}{\equiv} Y$, se hanno esattamente la stessa distribuzione;

(ii) X ed Y sono *computazionalmente* (t, ϵ)-indistinguibili, indicato con $X \stackrel{c}{\approx} Y$, se per ogni attaccante PPT $\mathscr{D}$ eseguibile in tempo t e per ogni n sufficientemente grande, abbiamo che

$$|\mathbb{P}[\mathscr{D}(X_n) = 1] - \mathbb{P}[\mathscr{D}(Y_n) = 1]| \leq \epsilon(n),$$

dove la probabilità è presa sulla randomicità di $\mathscr{D}$ e di X_n ed Y_n;

(iii) X ed Y sono *statisticamente* ϵ-indistinguibili, indicato con $X \stackrel{s}{\approx} Y$, se per ogni n sufficientemente grande, abbiamo che

$$\Delta(X_n, Y_n) \leq \epsilon(n),$$

dove $\Delta(\cdot)$ è l'operatore di distanza statistica tra distribuzioni (cf. Definizione A.3). ■

[17] Il motivo per cui dobbiamo parlare di insiemi di distribuzioni e non singole distribuzioni è che distinguere due distribuzioni singole può essere facile usando la ricerca esaustiva.

Osserviamo che l'attaccante $\mathscr{D}$ al punto (ii) della Definizione 3.1 restituisce un valore in $\{0,1\}$ come tentativo di distinguere X_n da Y_n. Notare inoltre che l'indistinguibilità statistica non fa alcuna ipotesi sul potere computazionale dell'attaccante. (In effetti essa è usata nel contesto della sicurezza incondizionata.) Non è difficile mostrare (come l'intuizione suggerisce) che l'indistiguibilità statistica implica quella computazionale (cf. Esercizio 3.2); tuttavia la direzione opposta non è verificata (ma una dimostrazione di questo fatto è più complessa [Gol01, Proposizione 3.2.3]).

Prima di procedere oltre (e vedere come applicare il concetto d'indistinguibilità per generare sequenze binarie pseudocasuali) introduciamo una tecnica molto usata in crittografia detta *l'argomento ibrido*. Tale tecnica permette di dimostrare che due insiemi di distribuzioni sono indistinguibili definendo una serie di *distribuzioni ibride intermedie* e mostrando che gli ibridi consecutivi sono indistinguibili.

Teorema 3.1 (Argomento ibrido). *Sia $\ell = \ell(n)$ un polinomio in n, ovvero $\ell = \mathrm{poly}(n)$. Se $X^{(0)} \stackrel{c}{\approx} X^{(1)}$, $X^{(1)} \stackrel{c}{\approx} X^{(2)}, \ldots, X^{(\ell-1)} \stackrel{c}{\approx} X^{(\ell)}$, allora $X^{(0)} \stackrel{c}{\approx} X^{(\ell)}$. Più precisamente, se per ogni indice $i = 0, \ldots, \ell - 1$ gli insiemi di distribuzioni $X^{(i)}$ ed $X^{(i+1)}$ sono $(t, \epsilon/\ell)$-indistinguibili, allora $X^{(0)}$ ed $X^{(\ell)}$ sono (t, ϵ)-indistinguibili.*

Dimostrazione. Osserviamo che se $\epsilon(n)$ non è trascurabile, anche $\epsilon(n)/\ell(n)$ non lo è, poiché $\ell(n) = \mathrm{poly}(n)$ (cf. Esercizio 1.1). Basta mostrare che, se è possibile distinguere $X^{(0)}$ da $X^{(\ell)}$ con probabilità $\geq \epsilon$, esiste almeno un indice $0 \leq i < \ell$ per cui è possibile distinguere $X^{(i)}$ da $X^{(i+1)}$ con probabilità $\geq \epsilon/\ell$ contro l'ipotesi che $X^{(i)} \stackrel{c}{\approx} X^{(i+1)}$ per ogni i. Sia $p_i = \mathbb{P}\left[\mathscr{D}(X^{(i)}) = 1\right]$. Supponiamo che esista un'attaccante PPT $\mathscr{D}$ in grado di distinguere $X^{(0)}$ da $X^{(\ell)}$ con vantaggio $\epsilon(n)$, ovvero $|p_0 - p_\ell| \geq \epsilon(n)$. Manipolando,

$$\begin{aligned}
\epsilon &\leq |p_0 - p_\ell| \\
&= |(p_\ell - p_{\ell-1}) + (p_{\ell-1} - p_{\ell-2}) + \ldots + (p_1 - p_0)| \\
&= [\text{Dalla disuguaglianza triangolare (cf. Definizione A.1)}] \\
&\leq |p_\ell - p_{\ell-1}| + |p_{\ell-1} - p_{\ell-2}| + \ldots + |p_1 - p_0| \\
&= \sum_{i=0}^{\ell-1} |p_{i+1} - p_i| \,,
\end{aligned}$$

da cui segue immediatamente che deve esistere almeno un indice i tale per cui

$$|p_{i+1} - p_i| \geq \epsilon/\ell,$$

cioè è possibile distinguere $X^{(i)}$ da $X^{(i+1)}$ con vantaggio almeno ϵ/ℓ. □

L'argomento ibrido è uno strumento molto potente. Supponiamo di voler dimostrare che gli insiemi di distribuzioni X ed Y sono indistinguibili. Ci basta definire (a nostro piacere!) un numero polinomiale di distribuzioni ibride $X^{(0)}, \ldots, X^{(\ell)}$ tali che: (i) $X^{(0)} = X$ ed $X^{(\ell)} = Y$ e (ii) $X^{(i)} \stackrel{c}{\approx} X^{(i+1)}$ per ogni $i = 1, \ldots, \ell - 1$. Ne segue $X \stackrel{c}{\approx} Y$. Come vedremo, la definizione degli ibridi è spesso "artificiale", ma ciò è del tutto irrilevante, in quanto $\mathscr{D}$ non è altro un algoritmo che dato un certo input decide per un bit, non importa come l'input sia stato generato!

Mostriamo ora un fatto molto utile, cioè che campioni multipli (indipendenti) estratti da distribuzioni computazionalmente indistinguibili, costituiscono una stringa computazionalmente indistinguibile da una stringa puramente casuale.

Teorema 3.2 (Indistinguibilità di campioni multipli). *Consideriamo due insiemi di distribuzioni $X = \{X_n\}_{n\in\mathbb{N}}$ ed $Y = \{Y_n\}_{n\in\mathbb{N}}$, efficientemente campionabili e $(O(t), \epsilon/\ell)$-indistinguibili. Allora, per ogni $\ell(n) = \mathrm{poly}(n)$, gli insiemi di distribuzioni $\widehat{X} = \{X_n^{(1)}, \ldots, X_n^{(\ell)}\}_{n\in\mathbb{N}}$ e $\widehat{Y} = \{Y_n^{(1)}, \ldots, Y_n^{(\ell)}\}_{n\in\mathbb{N}}$ sono (t, ϵ)-indistinguibili.*

Dimostrazione. Dato un attaccante PPT $\mathscr{D}$ in grado di distinguere $\widehat{X}$ da $\widehat{Y}$ con probabilità ϵ come nell'enunciato del teorema, mostriamo come costruire un attaccante PPT $\mathscr{D}'$ che distingue un campione di X da un campione di Y con probabilità non trascurabile, contro l'ipotesi che X ed Y siano indistinguibili. Per fare ciò useremo l'argomento ibrido. Sia $\mathscr{D}$ un attaccante PPT tale che

$$\epsilon(n) = \left| \mathbb{P}\left[\mathscr{D}(X_n^{(1)}, \ldots, X_n^{(\ell)}) = 1\right] - \mathbb{P}\left[\mathscr{D}(Y_n^{(1)}, \ldots, Y_n^{(\ell)}) = 1\right] \right|.$$

Per ogni indice $0 \leq i \leq \ell(n)$, definiamo la distribuzione ibrida $\mathcal{H}_n^i$ come la sequenza contenente i campioni prelevati da X_n seguiti da $\ell(n) - i$ campioni prelevati da Y_n, cioè

$$\mathcal{H}_n^i = \left(X_n^{(1)}, \ldots, X_n^{(i)}, Y_n^{(i+1)}, \ldots, Y_n^{(\ell)}\right).$$

Osserviamo che $\mathcal{H}_n^0 \stackrel{d}{\equiv} \widehat{Y}_n$ e che $\mathcal{H}_n^\ell \stackrel{d}{\equiv} \widehat{X}_n$. Inoltre, dal Teorema 3.1, sappiamo che se $\mathscr{D}$ può distinguere $\widehat{X}_n$ e $\widehat{Y}_n$ con probabilità ϵ, allora può anche distinguere due ibridi consecutivi con probabilità ϵ/ℓ.

Il punto chiave ora è che, siccome due ibridi $\mathcal{H}_n^i$ ed $\mathcal{H}_n^{i+1}$ differiscono solo per l'elemento nella posizione $(i+1)$-sima (perché in entrambe le distribuzioni

i primi i campioni provengono da X_n e gli ultimi $\ell(n)-i-1$ da Y_n), possiamo usare il fatto che $\mathscr{D}$ può distinguere due ibridi consecutivi con probabilità non trascurabile per costruire un avversario $\mathscr{D}'$ che distingue un campione di X da uno di Y, contro l'ipotesi che $X \overset{c}{\approx} Y$. L'algoritmo per $\mathscr{D}'$ è il seguente:

1. Dato un campione ζ, scegli $i \xleftarrow{\$} \{0,\ldots,\ell(n)-1\}$ e genera il vettore $\mathcal{H}_n^* = (X_n^{(1)},\ldots,X_n^{(i)},\zeta,Y_n^{(i+2)},\ldots,Y_n^{(\ell)})$ (qui usiamo il fatto che X ed Y sono efficientemente campionabili per costruire il vettore $\mathcal{H}_n^*$).

2. Invoca $\mathscr{D}$ con input $\mathcal{H}_n^*$ e decidi lo stesso bit che decide $\mathscr{D}$.

Osserviamo che se ζ è prelevato da X_n, allora $\mathcal{H}_n^*$ è distribuito esattamente come $\mathcal{H}_n^{i+1}$. D'altra parte se ζ è prelevato da Y_n, allora $\mathcal{H}_n^*$ è distribuito esattamente come $\mathcal{H}_n^i$. Ciò accade perché i campioni sono indipendenti e non conta l'ordine con cui sono generati. Siccome poi ogni i è scelto con probabilità $1/\ell(n)$ possiamo concludere

$$\mathbb{P}\left[\mathscr{D}'(X_n)=1\right] = \frac{1}{\ell(n)} \cdot \sum_{i=0}^{\ell(n)-1} \mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{i+1})=1\right]$$
$$\mathbb{P}\left[\mathscr{D}'(Y_n)=1\right] = \frac{1}{\ell(n)} \cdot \sum_{i=0}^{\ell(n)-1} \mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{i})=1\right].$$

Ne deriva

$$\begin{aligned}
&\left|\mathbb{P}\left[\mathscr{D}'(X_n)=1\right] - \mathbb{P}\left[\mathscr{D}'(Y_n)=1\right]\right| \\
&\qquad = \frac{1}{\ell(n)} \cdot \left| \sum_{i=0}^{\ell(n)-1} \mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{i+1})=1\right] - \sum_{i=0}^{\ell(n)-1} \mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{i})=1\right] \right| \\
&\qquad = \frac{1}{\ell(n)} \cdot \left|\mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{\ell})=1\right] - \mathbb{P}\left[\mathscr{D}(\mathcal{H}_n^{0})=1\right]\right| \\
&\qquad = \frac{1}{\ell(n)} \cdot \left|\mathbb{P}\left[\mathscr{D}(\widehat{X}_n)=1\right] - \mathbb{P}\left[\mathscr{D}(\widehat{Y}_n)=1\right]\right| \\
&\qquad \geq \frac{\epsilon(n)}{\ell(n)},
\end{aligned}$$

contro l'ipotesi che $X \overset{c}{\approx} Y$. □

3.2 Pseudorandomicità ed impredicibilità

Un generatore pseudocasuale (*Pseudorandom Generator*, PRG), è un algoritmo deterministico G che riceve come input un piccolo seme di *vera* randomicità (diciamo n bit) per produrre una lunga stringa (diciamo $\ell(n)$ bit) che sia "pseudocasuale". (In questo senso un PRG è un *amplificatore di randomicità*.) Per dare una definizione formale, useremo il concetto di indistiguibilità introdotto nel paragrafo precedente: G è pseudocasuale se la distribuzione dell'output che produce è *computazionalmente indistinguibile* dalla distribuzione uniforme $U_{\ell(n)}$ su $\{0,1\}^{\ell(n)}$. Formalmente abbiamo la seguente:

Definizione 3.2 (Generatore Pseudocasuale). Sia $n \in \mathbb{N}$ un parametro statistico di sicurezza ed $\ell : \mathbb{N} \to \mathbb{N}$ una funzione calcolabile in modo efficiente tale che $\ell(n) > n$ per ogni n. La funzione $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ è un *generatore pseudocasuale* (t,ϵ)-sicuro se per ogni avversario PPT $\mathscr{D}$ eseguibile in tempo t, risulta

$$\left|\mathbb{P}\left[\mathscr{D}(G(U_n)) = 1\right] - \mathbb{P}\left[\mathscr{D}(U_{\ell(n)}) = 1\right]\right| \leq \epsilon(n),$$

per ogni $n \in \mathbb{N}$. ■

Intuitivamente la Definizione 3.2 equivale al seguente esperimento mentale (cf. Fig. 3.1). L'attaccante ha accesso ad un oracolo che restituisce una stringa in $\{0,1\}^{\ell(n)}$: tale stringa può essere estratta da $G(U_n)$ (ovvero applicando il PRG ad un input casuale in $\{0,1\}^n$) oppure da $U_{\ell(n)}$ (ovvero dalla distribuzione uniforme su $\{0,1\}^{\ell(n)}$). L'attaccante $\mathscr{D}$ deve distinguere i due casi, quindi ritorna un bit per esprimere la sua ipotesi sulla natura dell'oracolo. Quando G

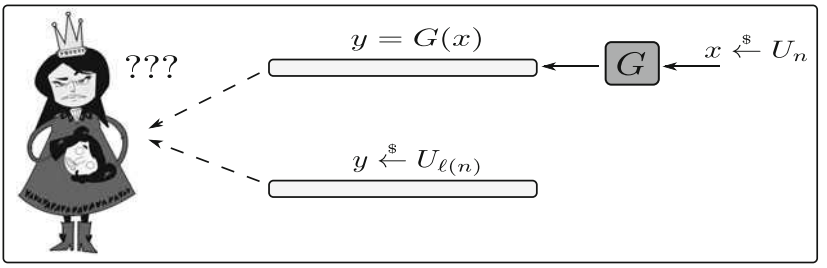


Fig. 3.1. Sicurezza di un PRG

è un generatore pseudocasuale, la probabilità che la decisione di $\mathscr{D}$ sia la stessa nei due casi è trascurabile (in n).

Per esemplificare la potenza della tecnica dell'argomento ibrido introdotta nel paragrafo precedente, mostriamo il seguente:

Lemma 3.3 (Composizione di PRG). *Se $G_1 : \{0,1\}^n \to \{0,1\}^{\ell_1(n)}$ è un PRG (t, ϵ_1)-sicuro e $G_2 : \{0,1\}^{\ell_1(n)} \to \{0,1\}^{\ell_2(n)}$ è un PRG (t, ϵ_2)-sicuro, allora $G : \{0,1\}^n \to \{0,1\}^{\ell_2(n)}$, definito come $G(x) = G_2(G_1(x))$, è un PRG $(t, \epsilon_1 + \epsilon_2)$-sicuro.*

Dimostrazione. Ci occorre il seguente fatto elementare: se X ed Y sono tali che $X \overset{c}{\approx} Y$ e se f è una funzione calcolabile in tempo polinomiale, allora $f(X) \overset{c}{\approx} f(Y)$ (cf. Esercizio 3.3). Infatti se esiste un attaccante PPT $\mathscr{D}$ che distingue $f(X)$ da $f(Y)$ è immediato costruire $\mathscr{D}'$ che distingue X da Y come segue: dato ζ (elemento di X oppure di Y), $\mathscr{D}'$ lancia $\mathscr{D}(f(\zeta))$ e ritorna lo stesso bit deciso da $\mathscr{D}$. (Notare che ciò è fattibile perché $f(\cdot)$ è calcolabile in tempo polinomiale.)

Passiamo alla dimostrazione del lemma. Come anticipato la prova usa l'argomento ibrido. Per ipotesi sappiamo che $G_1(U_n) \overset{c}{\approx} U_{\ell_1(n)}$ e $G_2(U_{\ell_1(n)}) \overset{c}{\approx} U_{\ell_2(n)}$. Dobbiamo mostrare che $G(U_n) = G_2(G_1(U_n)) \overset{c}{\approx} U_{\ell_2(n)}$. Definiamo la distribuzione ibrida intermedia $G_2(U_{\ell_1(n)})$. Siccome $G_2(\cdot)$ è calcolabile in tempo polinomiale e poiché $G_1(U_n)$ ed $U_{\ell_1(n)}$ sono (t, ϵ_1)-indistinguibili, usando l'osservazione fatta all'inizio della prova possiamo concludere che anche $G_2(G_1(U_n))$ e $G_2(U_{\ell_1(n)})$ sono (t, ϵ_1)-indistinguibili. D'altra parte $G_2(U_{\ell_1(n)})$ ed $U_{\ell_2(n)}$ sono (t, ϵ_2)-indistinguibili per ipotesi e quindi, usando il Teorema 3.1, abbiamo che $G_2(G_1(U_n))$ ed $U_{\ell_2(n)}$ sono $(t, \epsilon_1 + \epsilon_2)$-indistinguibili, ovvero G è un PRG. □

Osserviamo inoltre che, poiché campioni multipli di distribuzioni computazionalmente indistinguibili sono computazionalmente indistinguibili (cf. Teorema 3.2), otteniamo il seguente corollario:

Corollario 3.4. *Per ogni $\ell'(n) = \mathrm{poly}(n)$, abbiamo che $\ell'(n)$ campioni estratti da $G(\cdot)$ costituiscono una stringa pseudocasuale di lunghezza $\ell'(n) \cdot \ell(n)$. In altri termini $(G(x_1), \ldots, G(x_{\ell'}))$ è computazionalmente indistinguibile dalla distribuzione uniforme su $\{0,1\}^{\ell' \cdot \ell}$.*

Impredicibilità. Un altro modo per definire un generatore pseudocasuale è attraverso il concetto di *impredicibilità*. Tale nozione, a prima vista più debole

di quella di pseudorandomicità, è stata dimostrata essere (sorprendentemente) equivalente da Yao [Yao82]. Informalmente, una sequenza lunga n bit è impredicibile se, per ogni $1 \leq i \leq n$, nessun avversario PPT può prevedere il bit i-simo dati i precedenti $i-1$ bit con probabilità maggiore di $1/2$.

Definizione 3.3 (Impredicibilità). Sia $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ una funzione deterministica calcolabile in tempo polinomiale, ed indichiamo con $G(x) = g_1, \ldots, g_{\ell(n)}$ i bit prodotti da $G(\cdot)$. Diremo che $G(\cdot)$ è (t,ϵ)-impredicibile se per ogni indice $1 \leq i \leq \ell(n)$ ed ogni avversario PPT $\mathscr{A}$ eseguibile in tempo t risulta

$$\mathbb{P}\left[b = g_i : \; x \xleftarrow{\$} \{0,1\}^n, b \leftarrow \mathscr{A}(g_1, \ldots, g_{i-1})\right] < \frac{1}{2} + \epsilon(n).$$

In altri termini, nessun avversario $\mathscr{A}$ può predire $b = g_i$ dati $g_1, \ldots, g_{i-1}$ con probabilità migliore di $1/2$. ■

È facile vedere che se G è pseudocasuale, allora esso è anche impredicibile. Infatti se dato $G(x) = g_1, \ldots, g_\ell$ si può prevedere g_i a partire da $g_1, \ldots, g_{i-1}$, allora è immediato distinguere $G(U_n)$ da $U_{\ell(n)}$. Sorprendentemente anche l'altra direzione è verificata:

Teorema 3.5 (Impredicibilità ⇒ pseudocasualità). *Se $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ è $(O(t),\epsilon)$-impredicibile allora esso è anche un PRG $(t, \ell \cdot \epsilon)$-sicuro.*

Dimostrazione. Sia $\ell(n) = \text{poly}(n)$ la lunghezza dell'output di $G(\cdot)$. Definiamo $G(x) = g_1, \ldots, g_{\ell(n)}$ ed indichiamo con $G_i = g_1 \ldots, g_i$ i primi i bit di $G(x)$. Sia inoltre Ω_i una stringa casuale di lunghezza i. Dobbiamo mostrare che se $G(\cdot)$ è impredicibile, allora $G_\ell \overset{c}{\approx} \Omega_\ell$. Useremo un argomento ibrido con le seguenti distribuzioni:

$$\mathcal{H}_n^0 = \Omega_\ell, \mathcal{H}_n^1 = G_1\Omega_{\ell-1}, \ldots, \mathcal{H}_n^i = G_i\Omega_{\ell-i}, \ldots, \mathcal{H}_n^{\ell-1} = G_{\ell-1}\Omega_1, \mathcal{H}_n^\ell = G_\ell.$$

Siccome $\ell = \text{poly}(n)$ e poiché $\mathcal{H}_n^0 = \Omega_\ell$ ed $\mathcal{H}_n^\ell = G_\ell$, sappiamo dal Teorema 3.1 che è sufficiente mostrare $\mathcal{H}_n^{i-1} \overset{c}{\approx} \mathcal{H}_n^i$ per ogni $1 \leq i \leq \ell$. Osserviamo che due ibridi consecutivi sono molto simili, in particolare entrambi hanno la forma $G_{i-1}||b||\Omega_{\ell-i}$ dove b è il prossimo bit g_i del flusso prodotto da $G(\cdot)$ (nel caso di $\mathcal{H}_n^i$) oppure un bit casuale ω (nel caso di $\mathcal{H}_n^{i-1}$). Intuitivamente il fatto che i due ibridi sono indistinguibili (per ogni i) deriva proprio dal fatto che g_i è impredicibile dato G_{i-1}.

Per dimostrare che $G_{i-1}\Omega_{\ell-i+1} \overset{c}{\approx} G_i\Omega_{\ell-i}$ mostreremo che, se esiste un attaccante PPT $\mathscr{D}$ in grado di distinguere i due ibridi con probabilità $\epsilon(n)$, allora esiste un attaccante PPT $\mathscr{A}$ in grado di predire il valore di g_i dato G_{i-1} con probabilità $1/2+\epsilon(n)$. Poiché $G(\cdot)$ è impredicibile per ipotesi, ne segue che ϵ deve essere trascurabile in n e quindi anche $\ell \cdot \epsilon$ è trascurabile e $G(\cdot)$ è un PRG. Supponiamo allora che $\mathbb{P}[\mathscr{D}(G_i\Omega_{\ell-i}) = 0] = p^* + \epsilon(n)$ e $\mathbb{P}[G_{i-1}\Omega_{\ell-i+1} = 0] = p^*$, cioè $\mathscr{D}$ restituisce 0 più spesso quando il bit i-simo proviene da $G(\cdot)$ piuttosto che essere casuale. (Questo vale senza perdita di generalità, in quanto in caso contrario basta scambiare p^* con $1-p^*$ e 0 con 1 nell'output di $\mathscr{D}$.)

Costruiamo ora $\mathscr{A}$, che conosce G_{i-1} e deve predire g_i. Per far ciò $\mathscr{A}$ usa $\mathscr{D}$ come segue:

1. Estrai $\omega \xleftarrow{\$} \{0,1\}$ e lancia $\mathscr{D}(G_{i-1}||\omega||\Omega_{\ell-i})$.
2. Se $\mathscr{D}$ ritorna 0 restituisci ω, altrimenti ritorna $1-\omega$.

L'intuizione dietro il funzionamento di $\mathscr{A}$ è la seguente: siccome sappiamo che $\mathscr{D}$ ritorna zero con probabilità maggiore se il bit i-simo della stringa che riceve come input è g_i, quando la risposta è zero è più probabile che il bit da predire sia proprio ω. Se d'altra parte $\mathscr{D}$ ritorna 1, sembra più probabile che il bit da predire sia il complemento di ω, ovvero $1-\omega$.

Resta da analizzare la probabilità con cui $\mathscr{A}$ ha successo. Per fare ciò introduciamo le seguenti probabilità

$$\begin{aligned}
p_{00} &= \mathbb{P}[\mathscr{D}(G_{i-1}||0||\Omega_{\ell-i}) = 0 \mid g_i = 0] \\
p_{01} &= \mathbb{P}[\mathscr{D}(G_{i-1}||0||\Omega_{\ell-i}) = 0 \mid g_i = 1] \\
p_{10} &= \mathbb{P}[\mathscr{D}(G_{i-1}||1||\Omega_{\ell-i}) = 0 \mid g_i = 0] \\
p_{11} &= \mathbb{P}[\mathscr{D}(G_{i-1}||1||\Omega_{\ell-i}) = 0 \mid g_i = 1] .
\end{aligned}$$

Sia inoltre $\gamma = \mathbb{P}[g_i = 0]$. Intuitivamente, queste probabilità giocano un ruolo importante perché dovendo calcolare $\mathbb{P}[\mathscr{A}(G_{i-1}) = g_i]$, sarà conveniente condizionare sui casi $g_i = 0$ oppure $g_i = 1$. Inoltre poiché $\mathscr{A}$ usa $\mathscr{D}$, tale probabilità è esprimibile come funzione di γ e di p_{00}, p_{01}, p_{10} e p_{11}. Cominciamo a manipolare le espressioni relative alle quantità note:

$$\begin{aligned}
p^* &= \mathbb{P}[\mathscr{D}(G_{i-1}\omega\Omega_{\ell-i}) = 0] \\
&= \frac{1}{2}\left(\mathbb{P}[\mathscr{D}(G_{i-1}0\Omega_{\ell-i})] + \mathbb{P}[G_{i-1}1\Omega_{\ell-i}]\right) \\
&= \frac{1}{2}\left(\gamma p_{00} + (1-\gamma)p_{01} + \gamma p_{10} + (1-\gamma)p_{11}\right)
\end{aligned}$$

$$\begin{aligned} p^* + \epsilon(n) &= \mathbb{P}\left[\mathscr{D}(G_{i-1} g_i \Omega_{\ell-i}) = 0\right] \\ &= \gamma p_{00} + (1-\gamma) p_{11}. \end{aligned}$$

Risolvendo rispetto ad $\epsilon(n)$ troviamo quindi:

$$\epsilon(n) = \frac{\gamma(p_{00} - p_{10}) + (1-\gamma)(p_{11} - p_{01})}{2}.$$

Possiamo quindi calcolare la probabilità che $\mathscr{A}$ abbia successo per un valore *fissato* di ω come segue:

$$\begin{aligned} \mathbb{P}\left[\mathscr{A}(G_{i-1}) = g_i \mid \omega = 0\right] &= \gamma \cdot \mathbb{P}\left[\mathscr{D}(G_{i-1} 0 \Omega_{\ell-i}) = 0 \mid g_i = 0\right] + \\ &\qquad (1-\gamma) \cdot \mathbb{P}\left[\mathscr{D}(G_{i-1} 0 \Omega_{\ell-i}) = 1 \mid g_i = 1\right] \\ &= \gamma p_{00} + (1-\gamma)(1 - p_{01}) \\ &= (1-\gamma) + \gamma p_{00} - (1-\gamma) p_{01} \\ \mathbb{P}\left[\mathscr{A}(G_{i-1}) = g_i \mid \omega = 1\right] &= \gamma \cdot \mathbb{P}\left[\mathscr{D}(G_{i-1} 1 \Omega_{\ell-i}) = 1 \mid g_i = 0\right] + \\ &\qquad (1-\gamma) \cdot \mathbb{P}\left[\mathscr{D}(G_{i-1} 1 \Omega_{\ell-i}) = 0 \mid g_i = 1\right] \\ &= \gamma(1 - p_{10}) + (1-\gamma) p_{11} \\ &= \gamma - \gamma p_{10} + (1-\gamma) p_{11}. \end{aligned}$$

Infine, condizionando su ω, otteniamo:

$$\begin{aligned} \mathbb{P}\left[\mathscr{A}(G_{i-1}) = g_i\right] &= \frac{1}{2}\mathbb{P}\left[\mathscr{A}(G_{i-1}) = g_i \mid \omega = 0\right] + \frac{1}{2}\mathbb{P}\left[\mathscr{A}(G_{i-1}) = g_i \mid \omega = 1\right] \\ &= \frac{1}{2}\left((1-\gamma) + \gamma p_{00} - (1-\gamma) p_{01} + \gamma - \gamma p_{10} + (1-\gamma) p_{11}\right) \\ &= \frac{1}{2} + \frac{\gamma(p_{00} - p_{10}) + (1-\gamma)(p_{11} - p_{01})}{2} \\ &= \frac{1}{2} + \epsilon(n). \end{aligned}$$

□

3.3 Funzioni unidirezionali e predicati estremi

Introduciamo qui il concetto fondamentale di funzione unidirezionale (*One-Way Function*, OWF). Tale primitiva ha applicazioni svariate in crittografia e come vedremo è sufficiente per realizzare gran parte degli schemi crittografici

esistenti. In particolare una OWF è necessaria e sufficiente per costruire un PRG (come discuteremo in parte nel prossimo paragrafo).

Intuitivamente una funzione unidirezionale è una funzione *facile da calcolare*, ma (quasi sempre) *difficile da invertire*. Il primo requisito è semplice da formalizzare: richiederemo che la funzione sia calcolabile in tempo polinomiale. Quanto al secondo requisito, siccome siamo interessati alla sicurezza computazionale, richiederemo che nessun avversario PPT possa invertire la funzione con probabilità non trascurabile.

Definizione 3.4 (Funzione unidirezionale). Una funzione $f : \{0,1\}^* \to \{0,1\}^*$ è una funzione unidirezionale (t, ϵ)-sicura se rispetta le seguenti due condizioni:

(i) *Facile da calcolare.* Esiste un algoritmo polinomiale che riceve in ingresso un valore $x \in \{0,1\}^*$ e restituisce $f(x)$.

(ii) *Difficile da invertire.* Per ogni avversario PPT $\mathscr{A}$ eseguibile in tempo t e per ogni $n \in \mathbb{N}$ risulta

$$\mathbb{P}\left[\mathscr{A}\left(1^n, y\right) = x' : \; x \xleftarrow{\$} \{0,1\}^n, y = f(x), f(x') = y\right] \leq \epsilon(n),$$

dove la probabilità è calcolata sulla randomicità usata da $\mathscr{A}$. ■

In termini intuitivi la Definizione 3.4 è equivalente al seguente esperimento mentale: per una qualche funzione $f : \{0,1\}^* \to \{0,1\}^*$ l'attaccante $\mathscr{A}$ conosce $y = f(x)$ e deve calcolare un valore x' tale che $f(x') = f(x)$ (cioè deve trovare una *pre-immagine* di y). Osserviamo che dato $y = f(x)$ è sufficiente che $\mathscr{A}$ trovi *una qualsiasi* pre-immagine x' che soddisfi $f(x') = y$, non importa che $x' = x$.

Mondi di Impagliazzo. Esistono le funzioni unidirezionali nel nostro mondo? Non sappiamo dare una risposta a questa domanda. Nel 1995, Russel Impagliazzo [Imp95] ha ipotizzato 5 diverse possibilità per il mondo in cui viviamo, a seconda delle risposte (non note) ad alcune delle domande aperte più importanti in informatica teorica. Per esemplificare le conseguenze Impagliazzo si è riferito ad un famoso aneddoto relativo a Gauss.

Si tramanda che quando Gauss fosse studente alle scuole elementari, il suo Maestro, J.G. Büttner, assegnò come prova in classe ai suoi studenti di calcolare la somma dei primi 100 numeri naturali. A quanto pare il giovane Gauss

trovò la soluzione nel giro di pochi secondi [Hal70].[18] Impagliazzo immagina il proseguo della storia, supponendo che il Maestro fosse un esperto di teoria della complessità computazionale. Nelle settimane successive, il Maestro Büttner voleva avere la sua vendetta personale tentando di trovare un problema che Gauss non sarebbe stato in grado di risolvere. Il finale della storia è diverso a seconda della natura del mondo in cui viviamo.

1. Algoritmica. È il mondo in cui $\mathbf{P} = \mathbf{NP}$. In questo mondo il Maestro Büttner non ha alcuna speranza di ottenere la sua vendetta. Siccome il Maestro deve proporre un problema per il quale deve anche essere in grado di esibire una soluzione verificabile dagli studenti, egli dovrà scegliere un problema in $\mathbf{NP}$. Tuttavia, poiché $\mathbf{P} = \mathbf{NP}$, il metodo per verificare la soluzione può essere usato da Gauss per produrre la soluzione stessa.

2. Euristica. È il mondo in cui i problemi in $\mathbf{NP}$ non sono trattabili nel caso peggiore, ma sono risolvibili nel caso medio[19] per ogni distribuzione campionabile. Euristica è un mondo per certi versi paradossale: esistono istanze difficili di problemi in $\mathbf{NP}$, ma trovare tali istanze è anch'esso un problema intrattabile! Il Maestro Büttner potrebbe trovare un problema che Gauss non sia in grado di risolvere, ma potrebbero volerci settimane per trovare un problema che Gauss non possa risolvere in un giorno ed anni per trovare un problema che Gauss non possa risolvere in un mese.

3. Pessilandia. È il mondo in cui esistono problemi intrattabili nel caso medio, ma non esistono le funzioni unidirezionali. Questo significa che ogni processo $f(x)$ che sia facile da calcolare, è allo stesso tempo facile da invertire: dato $f(x)$ è possibile trovare un x' tale che $f(x) = f(x')$ circa nello stesso tempo necessario a calcolare $f(x)$. Sebbene in questo mondo esistano istanze di problemi difficili, non è possibile generare un'istanza *risolta* di un problema difficile. Consideriamo un processo che genera l'istanza di un problema e con esso la funzione che prende come input la randomicità usata per generarlo e restituisce il problema stesso. Se tale funzione fosse invertibile, allora, dato il problema come input, sarebbe possibile calcolare la randomicità usata per

[18] Probabilmente Gauss ha osservato che $100 + 1 = 101$, $99 + 2 = 101$, $98 + 3 = 101$ e così via, così che la somma dei primi 100 interi è semplicemente $50 \cdot 101 = 5050$.

[19] La *complessità nel caso peggiore* quantifica le risorse necessarie a risolvere un determinato problema nel caso peggiore. In pratica essa individua un limite superiore al tempo d'esecuzione di un algoritmo che risolve il problema *per ogni input*. La *complessità nel caso medio*, invece, studia la complessità di un problema per un'istanza casuale, ovvero quando l'input è estratto uniformemente a caso. Vedremo alcuni esempi concreti nel Capitolo 9.

produrre il problema stesso e quindi risolverlo. In Pessilandia il Maestro Büttner sarebbe in grado di trovare problemi che Gauss non è in grado di risolvere. Sfortunatamente neppure il Maestro sarebbe in grado di esibire una soluzione a tali problemi!

4. Minicrypt. È il mondo in cui le funzioni unidirezionali esistono, ma non esiste la "crittografia a chiave pubblica". Qui per crittografia a chiave pubblica si intende la possibilità di condividere un segreto con uno sconosciuto attraverso un canale pubblico non protetto (in realtà la crittografia a chiave pubblica, che studieremo nei Capitoli 6 e 8, è solo uno dei modi per ottenere tale scopo). In Minicrypt il Maestro Büttner può finalmente avere la meglio su Gauss: fissato un valore x e posto $y = f(x)$ può chiede a Gauss di calcolare x' tale che $f(x') = y$, conoscendo egli stesso una possibile soluzione.

5. Cryptomania. È il mondo in cui la crittografia a chiave pubblica è possibile, ovvero è possibile per due parti condividere un segreto usando solo canali pubblici non protetti, ed, inoltre, esistono le funzioni unidirezionali.[20] In Cryptomania Gauss è letteralmente umiliato. Non solo il Maestro Büttner può trovare un problema che Gauss non sia in grado di risolvere e di cui lui può esibire una soluzione. Egli può anche condividere la soluzione in segreto con tutto il resto della classe, facendo sì che Gauss sia l'unico a non poter risolvere il problema!

In quale mondo viviamo? La risposta a questa domanda non è nota, ma la maggior parte degli informatici teorici congettura che il nostro mondo sia Minicrypt o Cryptomania. In particolare si congettura l'esistenza delle funzioni unidirezionali. Alcuni candidati che incontreremo nel corso del testo sono:

- *Moltiplicazione di interi.* Fattorizzare interi della forma $N = pq$ con p e q primi di una certa dimensione (cf. Appendice C.2).

- *Funzione RSA.* Invertire l'esponenziale modulare $m^e \bmod N$ in $\mathbb{Z}_N^*$ dove $N = pq$ ed e è coprimo con $\varphi(N) = \#\mathbb{Z}_N^*$ (cf. Paragrafo 6.2).

- *Logaritmo discreto.* Il logaritmo in $\mathbb{Z}_p^*$ o un suo sottogruppo (cf. Appendice C.3).

- *Somme di sottoinsiemi.* Sia S un intero. Dato un insieme di oggetti con un certo peso, ad esempio in chilogrammi, determinare un sottoinsieme

[20] In realtà la prima ipotesi implica la seconda [IL89].

di essi tale da riempire un sacco contenente fino ad S chilogrammi (cf. Paragrafo 9.4).

Si può dimostrare che l'esistenza delle funzioni unidirezionali implica $\mathbf{P} \neq \mathbf{NP}$. D'altra parte non è chiaro se $\mathbf{P} \neq \mathbf{NP}$ implichi l'esistenza delle funzioni unidirezionali (questo è un altro grande problema aperto).

Predicati estremi. Sia $f(\cdot)$ una OWF, sappiamo che è difficile determinare x a partire da $f(x)$. Ma quanto è difficile calcolare un particolare bit di x? Potremmo essere portati a pensare che siccome $f(x)$ non è invertibile, non è possibile ricavare alcuna informazione su x in tempo polinomiale. Questo non è affatto vero. Esistono, infatti, esempi di funzioni che non sono invertibili in tempo polinomiale, eppure rivelano parecchia informazione su x. Ad esempio è facile vedere che la funzione $g(x_1, x_2) = (x_1, f(x_2))$ è una OWF, seppur evidentemente riveli metà del suo input! Si definisce allora il concetto di "predicato estremo" o anche "bit estremo" (*predicato hard-core* o anche *bit hard-core*) per una funzione unidirezionale, come segue:

Definizione 3.5 (Predicati estremi). Sia $f : \{0,1\}^* \to \{0,1\}^*$ una funzione unidirezionale (t, ϵ)-sicura. La mappa $\chi : \{0,1\}^* \to \{0,1\}$ è un predicato (t, ϵ)-estremo per $f(\cdot)$, se è calcolabile in tempo polinomiale, e se, per ogni avversario PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathbb{P}\left[\mathscr{A}(f(x)) = \chi(x)\right] \leq \frac{1}{2} + \epsilon(n),$$

dove la probabilità è presa sulla randomicità di $\mathscr{A}$ e $x \xleftarrow{\$} \{0,1\}^n$. ■

Una funzione non deve essere necessariamente unidirezionale per possedere un predicato estremo. D'altra parte in crittografia siamo interessati a costruire predicati estremi per funzioni unidirezionali. Il seguente importante teorema, dovuto a Goldreich e Levin [GL89], mostra che in effetti ciò è possibile *per ogni* funzione unidirezionale.

Teorema 3.6 (Predicati estremi di Goldreich e Levin). *Sia* $f : \{0,1\}^n \to \{0,1\}^n$ *una funzione unidirezionale* $(O(t), O(\epsilon^3/n^2))$*-sicura. Posto* $\mathbf{x} = (\mathbf{x}[1], \ldots, \mathbf{x}[n])$ *ed* $\mathbf{r} = (\mathbf{r}[1], \ldots, \mathbf{r}[n])$*, definiamo* $g(\mathbf{x}, \mathbf{r}) = (f(\mathbf{x}), \mathbf{r})$*, per* $|\mathbf{x}| = |\mathbf{r}| = n$*. Allora:*

(i) la funzione $g(\cdot,\cdot)$ è una OWF;

(ii) il bit

$$\chi(\mathbf{x},\mathbf{r}) = \mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 = \bigoplus_{i=1}^{n} \mathbf{x}[i] \cdot \mathbf{r}[i] = \bigoplus_{i:\mathbf{r}[i]=1} \mathbf{x}[i],$$

è un predicato (t,ϵ)-estremo per $g(\cdot,\cdot)$.

Dimostrazione. Il fatto che $g(\cdot,\cdot)$ sia una funzione unidirezionale è banale: siccome $g(\mathbf{x},\mathbf{r}) = (f(\mathbf{x}),\mathbf{r})$, invertire $g(\cdot,\cdot)$ è equivalente ad invertire $f(\cdot)$.

Quanto al secondo punto invece mostreremo come, dato un attaccante PPT $\mathscr{A}$ in grado di predire il valore del predicato estremo $\mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \pmod 2$ con probabilità sensibilmente migliore di $1/2$, sia possibile costruire un attaccante PPT $\mathscr{B}$ capace di calcolare $\mathbf{x}$ a partire da $\mathbf{y} = f(\mathbf{x})$, contro l'ipotesi che $f(\cdot)$ sia una funzione unidirezionale. Siccome la prova è complessa, nel seguito discuteremo prima due casi semplici per cercare di comprendere l'idea dietro la dimostrazione di Goldreich e Levin.

Alcuni casi semplici. Supponiamo che esista un attaccante PPT $\mathscr{A}$ in grado di predire il valore di $\mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \pmod 2$ dato $(f(\mathbf{x}),\mathbf{r})$ *sempre e comunque*. In questo caso, tutto diventa molto semplice: $\mathscr{B}$ può semplicemente invocare $\mathscr{A}(\mathbf{y},\mathbf{e}_i)$, dove $\mathbf{e}_i$ è l'i-esimo vettore della base canonica di $\mathbb{Z}_2^n$ (ovvero $\mathbf{e}_i$ è il vettore nullo con un solo 1 nella posizione i). Siccome $\mathbf{e}_i^{\mathsf{T}} \cdot \mathbf{x} = \mathbf{x}[i]$, in questo modo è possibile recuperare $\mathbf{x}$ bit a bit.

In realtà ovviamente $\mathscr{A}$ *non* è in grado di calcolare $\mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \pmod 2$ sempre e comunque, ma solo con una certa probabilità. Ad esempio è più ragionevole supporre che

$$\mathbb{P}\left[\mathscr{A}(f(\mathbf{x}),\mathbf{r}) = \mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 : \ \mathbf{x},\mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right] > \frac{3}{4} + \epsilon. \tag{3.1}$$

Anche questo è un rilassamento del caso generale in cui abbiamo sostituito $1/2$ con $3/4$ (per motivi che saranno chiari a breve). Dato un particolare valore di $\mathbf{x}$, definiamo una stima di come $\mathscr{A}$ sia in grado di predire il bit $\mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \pmod 2$ per uno specifico valore di $\mathbf{x}$ (mediando su tutti i possibili valori di $\mathbf{r}$):

$$S(\mathbf{x}) = \mathbb{P}\left[\mathscr{A}(f(\mathbf{x}),\mathbf{r}) = \mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 : \ \mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right].$$

Quindi l'Eq. (3.1) equivale a

$$\mathbb{E}\left[S(\mathbf{x})\right] > \frac{3}{4} + \epsilon. \tag{3.2}$$

D'altra parte è semplice mostrare che

$$\mathbb{P}\left[\mathbf{x} \text{ soddisfa } S(\mathbf{x}) > \frac{3}{4} + \frac{\epsilon}{2} : \ \mathbf{r} \stackrel{\$}{\leftarrow} \mathbb{Z}_2^n\right] > \frac{\epsilon}{2}. \tag{3.3}$$

Infatti, se ciò *non* si verificasse, condizionando sul fatto che $\mathbf{x}$ soddisfi $S(\mathbf{x}) > 3/4 + \epsilon/2$ oppure no, possiamo scrivere

$$\mathbb{E}\left[S(\mathbf{x})\right] \leq \frac{\epsilon}{2} \cdot 1 + \left(1 - \frac{\epsilon}{2}\right) \cdot \left(\frac{3}{4} + \frac{\epsilon}{2}\right) < \frac{3}{4} + \frac{\epsilon}{2},$$

il che contraddice l'Eq. (3.2).

Nel seguito mostreremo come costruire $\mathscr{B}$ in grado di invertire $f(\mathbf{x})$ solo se $\mathbf{x}$ soddisfa $S(\mathbf{x}) > 3/4 + \epsilon/2$: siccome sappiamo che la frazione di $\mathbf{x}$ per cui ciò si verifica è *non trascurabile* (cf. Eq. (3.3)), ciò implica che la probabilità con cui $\mathscr{B}$ inverte $f(\cdot)$ è non trascurabile, contro l'ipotesi che $f(\cdot)$ sia unidirezionale. Pertanto nel seguito assumeremo che $\mathbf{x}$ soddisfa $S(\mathbf{x}) > 3/4 + \epsilon/2$.

Dunque, dato $\mathbf{y} = f(\mathbf{x})$ da invertire, $\mathscr{B}$ è definito come segue (l'intuizione relativa al funzionamento di $\mathscr{B}$ è data subito dopo la sua descrizione):

1. Ripeti per ogni $j = 1, \ldots, \kappa \left(= O(\log n/\epsilon^2)\right)$: estrai a caso $\mathbf{r}_j \stackrel{\$}{\leftarrow} \mathbb{Z}_2^n$ e lancia $\mathscr{A}(\mathbf{y}, \mathbf{r}_j)$ ed $\mathscr{A}(\mathbf{y}, \mathbf{r}_j \oplus \mathbf{e}_i)$. Calcola $\mathscr{A}(\mathbf{y}, \mathbf{r}_j) \oplus \mathscr{A}(\mathbf{y}, \mathbf{r}_j \oplus \mathbf{e}_i)$.
2. Poni $\mathbf{x}[i]$ pari alla maggioranza dei κ valori ottenuti al passo (1).
3. Ripeti i passi (1) e (2) per ogni $i = 1, \ldots, n$, recuperando così $\mathbf{x} = (\mathbf{x}[1], \ldots, \mathbf{x}[n])$.

L'osservazione chiave dietro il funzionamento di $\mathscr{B}$ è che per ogni $\mathbf{r} \stackrel{\$}{\leftarrow} \mathbb{Z}_2^n$

$$\begin{aligned}\mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \ \oplus \ (\mathbf{r} \oplus \mathbf{e}_i)^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 &= \left(\bigoplus_{j \neq i} \mathbf{r}[j] \cdot \mathbf{x}[j] \oplus \mathbf{r}[i] \cdot \mathbf{x}[i]\right) \\ &\quad \oplus \left(\bigoplus_{j \neq i} \mathbf{r}[j] \cdot \mathbf{x}[j] \oplus (1 - \mathbf{r}[i]) \cdot \mathbf{x}[i]\right) \\ &= \mathbf{x}[i].\end{aligned}$$

Siccome $\mathbf{r}$ ed $\mathbf{r} \oplus \mathbf{e}_i$ sono uniformemente casuali quando $\mathbf{r}$ è uniformemente casuale, per ogni indice i fissato, e per ogni $\mathbf{x}$ che soddisfa $S(\mathbf{x}) > 3/4 + \epsilon/2$,

possiamo concludere:

$$\mathbb{P}\left[\mathscr{A}(\mathbf{y},\mathbf{r}) \neq \mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 : \ \mathbf{y} = f(\mathbf{x}), \mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right] < \frac{1}{4} - \frac{\epsilon}{2}$$
$$\mathbb{P}\left[\mathscr{A}(\mathbf{y},\mathbf{r} \oplus \mathbf{e}_i) \neq (\mathbf{r} \oplus \mathbf{e}_i)^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 : \ \mathbf{y} = f(\mathbf{x}), \mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right] < \frac{1}{4} - \frac{\epsilon}{2}.$$

Pertanto, con probabilità almeno $1 - 2(1/4 - \epsilon/2) = 1/2 + \epsilon$, l'attaccante $\mathscr{A}$ indovinerà in entrambi i casi e quindi $\mathscr{B}$ prevederà il valore di $\mathbf{x}[i]$ correttamente con probabilità $1/2 + \epsilon$.

Notare che ciò è valido, indipendentemente, *per ciascun* indice j nei passi (1) e (2) della descrizione di $\mathscr{B}$ (ovvero per ognuno dei vettori $\mathbf{r}_j$). Ripetere per ogni $j = 1, \ldots, \kappa$ e prendere infine la maggioranza delle risposte come ipotesi per $\mathbf{x}[i]$, ci permette di amplificare il vantaggio di $\mathscr{B}$ per la predizione di $\mathbf{x}[i]$ invocando il limite di Chernoff (cf. Teorema A.5). Informalmente tale limite afferma che la probabilità che κ esperimenti indipendenti sia "ϵ-lontana" dal suo valore atteso è dell'ordine $e^{-\Omega(\kappa\epsilon^2)}$. Definiamo la variabile aleatoria binaria Z_j (per $j = 1, \ldots, \kappa$), che assume valore 1 quando la predizione di $\mathscr{A}$ è corretta in *entrambi* i casi considerati nella descrizione di $\mathscr{B}$. Ovviamente tutte le Z_j sono indipendenti ed inoltre abbiamo mostrato $\mathbb{E}[Z_j] \geq 1/2 + \epsilon$. Posto $Z = \sum_{j=1}^{\kappa} Z_j$, per la linearità del valore atteso, abbiamo anche $\mathbb{E}[Z] \geq \kappa(1/2 + \epsilon)$ e quindi applicando il limite di Chernoff possiamo concludere $\mathbb{P}[Z < \kappa/2] \leq e^{-\Omega(\kappa\epsilon^2)}$. Siccome $\kappa = O(\log n^2/\epsilon^2)$ troviamo che la maggioranza delle risposte è la predizione corretta per $\mathbf{x}[i]$ con probabilità almeno $1 - 1/n^2$.

Ripetendo il ragionamento per ogni $i = 1, \ldots, n$ (usando l'opportuno vettore $\mathbf{e}_i$ di volta in volta), la probabilità che almeno una predizione di $\mathbf{x}[i]$ sia sbagliata è al più $n/n^2 = 1/n$. Quindi $\mathscr{B}$ recupera l'intero $\mathbf{x}$ con probabilità $1 - 1/n$ che è non trascurabile, contro l'ipotesi che $f(\cdot)$ sia unidirezionale.

Il caso generale. Purtroppo l'Eq. (3.1) non è vera in generale, ma possiamo solamente assumere:

$$\mathbb{P}\left[\mathscr{A}(f(\mathbf{x}),\mathbf{r}) = \mathbf{r}^{\mathsf{T}} \cdot \mathbf{x} \bmod 2 : \ \mathbf{x},\mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right] > \frac{1}{2} + \epsilon. \tag{3.4}$$

Come vedremo ciò complica ulteriormente la dimostrazione. Come già nel caso precedente, è immediato mostrare che l'Eq. (3.4) è equivalente a

$$\mathbb{E}[S(\mathbf{x})] > \frac{1}{2} + \epsilon,$$

dove $S(\mathbf{x})$ mantiene esattamente il significato che aveva in precedenza. In questo caso però ci concentreremo sui valori $\mathbf{x}$ tali che $S(\mathbf{x}) > 1/2 + \epsilon/2$. Un argomento simile a quello usato nel caso precedente mostra:

$$\mathbb{P}\left[\mathbf{x} \text{ soddisfa } S(\mathbf{x}) > \frac{1}{2} + \frac{\epsilon}{2} : \ \mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n\right] > \frac{\epsilon}{2},$$

quindi ancora una volta possiamo assumere che $\mathbf{x}$ soddisfa $S(\mathbf{x}) > 1/2 + \epsilon/2$ senza alcuna perdita di generalità, in quanto la frazione degli $\mathbf{x}$ per cui ciò è verificato è non trascurabile.

Dato $\mathbf{y} = f(\mathbf{x})$ da invertire, $\mathscr{B}$ è definito come segue (l'intuizione relativa al funzionamento di $\mathscr{B}$ è data subito dopo la sua descrizione):

1. Per ogni $j = 1, \ldots, \kappa\,(= O(\log n/\epsilon))$ estrai a caso $\mathbf{r}_j \xleftarrow{\$} \mathbb{Z}_2^n$ e cerca di indovinare (tirando a caso) i valori $b_j = \mathbf{r}_j^{\mathsf{T}} \cdot \mathbf{x}$.

2. Per ogni sottoinsieme $\mathcal{J} \subseteq \{1, \ldots, \kappa\}$ definiamo $\mathbf{r}_{\downarrow\mathcal{J}} = \bigoplus_{j\in\mathcal{J}} \mathbf{r}_j$ e $b_{\downarrow\mathcal{J}} = \bigoplus_{j\in\mathcal{J}} b_j$. Per ognuno dei 2^κ sottoinsiemi non vuoti $\mathcal{J} \subseteq \{1, \ldots, \kappa\}$, lancia quindi $\mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)$.

3. Poni $\mathbf{x}[i]$ pari alla maggioranza dei valori $b_{\downarrow\mathcal{J}} \oplus \mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)$.

4. Ripeti i passi precedenti per ogni $i = 1, \ldots, n$, recuperando così $\mathbf{x} = (\mathbf{x}[1], \ldots, \mathbf{x}[n])$.

Per prima cosa, come vedremo tra un attimo, $\mathscr{B}$ ha successo solo se *tutte* le n predizioni dei bit b_j al passo (1) sono corrette. Questo ci fa perdere immediatamente un fattore 2^κ nella riduzione. Sia $\mathcal{E}$ l'evento che $\mathscr{B}$ indovini tutti i valori b_j correttamente. Vogliamo mostrare, analizzando la descrizione di $\mathscr{B}$, che per ogni i e per ogni $\mathbf{x}$ che soddisfi $S(\mathbf{x}) > \epsilon/2 + 1/2$ risulta

$$\mathbb{P}\left[\mathscr{B}(\mathbf{y}) = \mathbf{x}[i] : \ \mathbf{y} = f(\mathbf{x}) \wedge \mathcal{E}\right] \geq 1 - \frac{1}{n^2}.$$

Questo sarà sufficiente per dimostrare l'asserto a patto che $2^{-\kappa}$ sia non trascurabile (ovvero a patto che κ non sia troppo grande), in quanto

$$\begin{aligned}\mathbb{P}\left[\mathscr{B}(\mathbf{y}) = \mathbf{x} : \ \mathbf{y} = f(\mathbf{x})\right] &\geq \mathbb{P}\left[\mathcal{E}\right] \cdot \mathbb{P}\left[\mathscr{B}(\mathbf{y}) = \mathbf{x} : \ \mathbf{y} = f(\mathbf{x}) \wedge \mathcal{E}\right] \\ &\geq \frac{1}{2^\kappa} \cdot \left(1 - \frac{1}{n}\right).\end{aligned}$$

L'osservazione fondamentale è che

$$\mathbf{r}_{\downarrow\mathcal{J}}^{\mathsf{T}} \cdot \mathbf{x} = \left(\bigoplus_{j \in \mathcal{J}} \mathbf{r}_j \right)^{\mathsf{T}} \cdot \mathbf{x} = \bigoplus_{j \in \mathcal{J}} \mathbf{r}_j^{\mathsf{T}} \cdot \mathbf{x} = \bigoplus_{j \in \mathcal{J}} b_j = b_{\downarrow\mathcal{J}}. \tag{3.5}$$

Quindi, $\mathscr{B}$ non fa altro che invocare $\mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)$ per tutti i 2^κ sottoinsiemi non vuoti $\mathcal{J} \subseteq \{1, \ldots, \kappa\}$ ed infine porre $\mathbf{x}[i]$ pari alla maggioranza dei valori $b_{\downarrow\mathcal{J}} \oplus \mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)$. Il punto è che, quando $\mathscr{A}$ indovina, si ha

$$\begin{aligned} b_{\downarrow\mathcal{J}} \ \oplus \ \mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i) &= b_{\downarrow\mathcal{J}} \oplus (\mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)^{\mathsf{T}} \cdot \mathbf{x} \\ &= [\text{usando l'Eq. (3.5)}] \\ &= \mathbf{r}_{\downarrow\mathcal{J}}^{\mathsf{T}} \cdot \mathbf{x} \oplus (\mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)^{\mathsf{T}} \cdot \mathbf{x} \\ &= \mathbf{e}_i^{\mathsf{T}} \cdot \mathbf{x} = \mathbf{x}[i], \end{aligned}$$

così che anche il valore predetto per $\mathbf{x}[i]$ è corretto.

Resta da dimostrare che, per un valore di κ non troppo grande, $\mathscr{B}$ indovina con probabilità almeno $1 - 1/n^2$. Sia $Z(\mathcal{J})$ la variabile aleatoria binaria che vale 1 quando la predizione di $\mathscr{A}$ relativa ad un dato insieme $\mathcal{J}$ è corretta, cioè quando $\mathscr{A}(\mathbf{y}, \mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i) = (\mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i)^{\mathsf{T}} \cdot \mathbf{x}$. Chiaramente $\mathscr{B}$ sbaglia a predire $\mathbf{x}[i]$ quando la maggioranza di $Z(\mathcal{J})$ è 0 cioè se $Z = \sum_{\mathcal{J} \neq \emptyset} Z(\mathcal{J}) < 2^{\kappa-1}$. Calcoliamo il valore atteso di $Z(\mathcal{J})$ per un dato $\mathcal{J}$. Siccome $\mathbf{x}$ è tale che $S(\mathbf{x}) > 1/2 + \epsilon/2$ e poiché $\mathbf{r}_{\downarrow\mathcal{J}}$ (e quindi anche $\mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i$) è uniformemente casuale in $\mathbb{Z}_2^n$, abbiamo $\mathbb{E}[Z(\mathcal{J})] > 1/2 + \epsilon/2$ e quindi $\mathbb{E}[Z] > 2^{\kappa-1}(1 + \epsilon)$.

A questo punto saremmo tentati ad invocare il limite di Chernoff come in precedenza. Purtroppo ciò non è possibile, in quanto chiaramente le $Z(\mathcal{J})$ *non sono indipendenti*. Però se $\mathcal{I}$ e $\mathcal{J}$ sono due insiemi *distinti*, i vettori $\mathbf{r}_{\downarrow\mathcal{I}}$ ed $\mathbf{r}_{\downarrow\mathcal{J}}$ sono indipendenti, il che implica che anche $\mathbf{r}_{\downarrow\mathcal{I}} \oplus \mathbf{e}_i$ ed $\mathbf{r}_{\downarrow\mathcal{J}} \oplus \mathbf{e}_i$ lo sono, e quindi ne segue che $Z(\mathcal{I})$ e $Z(\mathcal{J})$ sono indipendenti. Tecnicamente si dice che le variabili aleatorie $Z(\mathcal{J})$ sono *indipendenti a coppie*. Questo ci permette di utilizzare la disuguaglianza di Chebyshev (cf. Lemma A.4). Ci basta osservare che

$$\begin{aligned} \mathbb{V}[Z(\mathcal{J})] = \mathbb{E}\left[Z(\mathcal{J})^2\right] - (\mathbb{E}[Z(\mathcal{J})])^2 = \mathbb{E}[Z(\mathcal{J})] - (\mathbb{E}[Z(\mathcal{J})])^2 = \\ = \mathbb{E}[Z(\mathcal{J})](1 - \mathbb{E}[Z(\mathcal{J})]) = \frac{1 - \epsilon^2}{4}, \end{aligned}$$

e quindi, usando il fatto che le variabile aleatoria $Z(\mathcal{J})$ sono indipendenti a coppie (cf. Lemma A.2), otteniamo $\mathbb{V}[Z] = 2^\kappa \cdot \frac{1-\epsilon^2}{4}$. Applicando la disuguaglianza

di Chebyshev, possiamo concludere

$$\begin{aligned}\mathbb{P}\left[Z < 2^{\kappa-1}\right] &\leq \mathbb{P}\left[|Z - \mathbb{E}[Z]| > 2^{\kappa-1}\epsilon\right] \\ &\leq \frac{\mathbb{V}[Z]}{(2^{\kappa-1}\cdot\epsilon)^2} \leq \frac{4\cdot 2^{\kappa}\cdot(1-\epsilon^2)}{4\cdot 2^{2\kappa}\epsilon^2} \\ &\leq \frac{1}{2^{\kappa}\epsilon^2}.\end{aligned}$$

Siccome $\kappa = \log(n^2/\epsilon^2) = O(\log(n/\epsilon))$, otteniamo proprio che la probabilità che $\mathscr{B}$ indovini un singolo $\mathbf{x}[i]$ è $1 - 1/n^2$ (condizionando sull'evento $\mathcal{E}$ definito sopra). L'ultima cosa fondamentale da osservare è che $2^{-\kappa} = O(\epsilon^2/n^2)$ è non trascurabile, poiché ϵ è non trascurabile. Questo conclude la dimostrazione del teorema di Goldreich e Levin. □

3.4 Alcune costruzioni di PRG

Esistono diversi approcci, con efficienza variabile, per costruire un oggetto che soddisfi la Definizione 3.2. Nel seguito distinguiamo le costruzioni teoriche (il cui scopo è quello di ottenere qualche forma di sicurezza dimostrabile) dalle costruzioni pratiche (per lo più euristiche, senza sicurezza dimostrabile ma tipicamente con migliore efficienza).

Costruzioni teoriche. Mostriamo prima come sia possibile costruire un PRG che amplifica il suo input di un solo bit, i.e. tale che $\ell(n) = n + 1$. Come vedremo un PRG di questo tipo può essere usato per ottenere un'amplificazione di randomicità polinomiale.

Håstad, Impagliazzo, Levin e Luby [Hås+99] hanno mostrato come costruire un PRG da una *qualsiasi* funzione unidirezionale. Tuttavia la prova di questo fatto fondamentale è molto complessa ed esula dagli scopi del testo. Ci limiteremo invece a mostrare come sia possibile costruire un PRG da ogni permutazione unidirezionale (*One-Way Permutation*, OWP).

Definizione 3.6 (Permutazione unidirezionale). La funzione $f : \{0,1\}^n \to \{0,1\}^n$ è una permutazione unidirezionale (t,ϵ)-sicura se:

(i) $f(\cdot)$ permuta gli elementi di $\{0,1\}^n$;

(ii) $f(\cdot)$ è una funzione unidirezionale (t,ϵ)-sicura (cf. Definizione 3.4). ■

Sia ora $f : \{0,1\}^n \to \{0,1\}^n$ una OWP e sia $\chi(\cdot)$ un predicato estremo per $f(\cdot)$. L'idea è quella di sfruttare il fatto che dato $f(x)$ per un x a caso, è difficile predire $\chi(x)$ con probabilità significativamente migliore di $1/2$. Informalmente ciò significa che $\chi(x)$ è un bit pseudocasuale. D'altra parte, poiché $f(\cdot)$ è una permutazione, la stringa $f(x)$ è a distribuzione uniforme (per un x a caso) e quindi la funzione $G(x) = (f(x), \chi(x))$ è di fatto un PRG.

Teorema 3.7 (OWP $\Rightarrow$ PRG). *Sia $f : \{0,1\}^n \to \{0,1\}^n$ una OWP (t,ϵ)-sicura. Allora la funzione $G(x) = (f(x), \chi(x))$ è un PRG (t,ϵ)-sicuro con $\ell(n) = n+1$.*

Dimostrazione. Sia $\mathscr{D}$ un attaccante PPT per $G(\cdot)$ ed indichiamo con

$$\begin{aligned}\epsilon(n) &= \mathsf{IP}\left[\mathscr{D}(G(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right] - \mathsf{IP}\left[\mathscr{D}(\omega) = 1 : \ \omega \xleftarrow{\$} \{0,1\}^{n+1}\right] \\ &= \mathsf{IP}\left[\mathscr{D}(f(x), \chi(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right] - \mathsf{IP}\left[\mathscr{D}(\omega) = 1 : \ \omega \xleftarrow{\$} \{0,1\}^{n+1}\right],\end{aligned}$$

la probabilità con cui $\mathscr{D}$ distingue la distribuzione $G(U_n)$ da U_{n+1}. Siccome $f(\cdot)$ è una permutazione, possiamo scrivere

$$\begin{aligned}\mathsf{IP}\left[\mathscr{D}(\omega) = 1 : \ \omega \xleftarrow{\$} \{0,1\}^{n+1}\right] &= \mathsf{IP}\left[\mathscr{D}(\omega, \omega') = 1 : \ \omega \xleftarrow{\$} \{0,1\}^n, \omega' \xleftarrow{\$} \{0,1\}\right] \\ &= \mathsf{IP}\left[\mathscr{D}(f(x), \omega') = 1 : \ x \xleftarrow{\$} \{0,1\}^n, \omega' \xleftarrow{\$} \{0,1\}\right] \\ &= \frac{1}{2} \cdot \mathsf{IP}\left[\mathscr{D}(f(x), \chi(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right] \\ &\qquad + \frac{1}{2} \cdot \mathsf{IP}\left[\mathscr{D}(f(x), 1 - \chi(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right],\end{aligned}$$

dove l'ultima uguaglianza segue dal fatto che ω' è uguale ad $\chi(x)$ con probabilità $1/2$. Pertanto

$$\begin{aligned}\epsilon(n) = \frac{1}{2}\Bigg(&\mathsf{IP}\left[\mathscr{D}(f(x), \chi(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right] \\ &- \mathsf{IP}\left[\mathscr{D}(f(x), 1 - \chi(x)) = 1 : \ x \xleftarrow{\$} \{0,1\}^n\right]\Bigg).\end{aligned}$$

Consideriamo ora un attaccante $\mathscr{A}$ che, data in input la stringa $y = f(x)$, prova a predire il valore di $\chi(x)$ come segue:

1. Scegli $\omega' \xleftarrow{\$} \{0,1\}$.

2. Lancia $\mathscr{D}(y, \omega')$. Se $\mathscr{D}$ restituisce 1 ritorna ω', altrimenti ritorna $1 - \omega'$.

Manipolando,

$$\begin{aligned}
&\mathbb{P}\left[\mathscr{A}(f(x)) = \chi(x) :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&= \frac{1}{2} \cdot \mathbb{P}\left[\mathscr{A}(f(x)) = \chi(x) \mid \omega' = \chi(x) :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&\qquad + \frac{1}{2} \cdot \mathbb{P}\left[\mathscr{A}(f(x)) = \chi(x) \mid \omega' \neq \chi(x) :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&= \frac{1}{2}\Bigg(\mathbb{P}\left[\mathscr{D}(f(x), \chi(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&\qquad + \mathbb{P}\left[\mathscr{D}(f(x), 1 - \chi(x)) = 0 :\ x \xleftarrow{\$} \{0,1\}^n\right]\Bigg) \\
&= \frac{1}{2}\Bigg(\mathbb{P}\left[\mathscr{D}(f(x), \chi(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&\qquad + \left(1 - \mathbb{P}\left[\mathscr{D}(f(x), 1 - \chi(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right]\right)\Bigg) \\
&= \frac{1}{2} + \frac{1}{2}\Bigg(\mathbb{P}\left[\mathscr{D}(f(x), \chi(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right] \\
&\qquad - \mathbb{P}\left[\mathscr{D}(f(x), 1 - \chi(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right]\Bigg) \\
&= \frac{1}{2} + \epsilon(n).
\end{aligned}$$

Ovviamente $\mathscr{A}$ è eseguibile in tempo polinomiale. Siccome poi $\chi(\cdot)$ è un predicato hard-core per $f(\cdot)$, abbiamo che $\epsilon(n)$ deve essere trascurabile. Applicando un argomento simile segue che anche $-\epsilon(n)$ deve essere trascurabile. Pertanto,

$$\left|\mathbb{P}\left[\mathscr{D}(G(x)) = 1 :\ x \xleftarrow{\$} \{0,1\}^n\right] - \mathbb{P}\left[\mathscr{D}(\omega) = 1 :\ \omega \xleftarrow{\$} \{0,1\}^{n+1}\right]\right| \leq \epsilon(n)$$

è trascurabile e $G(\cdot)$ è un PRG, come desiderato. □

Possiamo usare il risultato del Teorema 3.7 per costruire un PRG con fattore d'espansione polinomiale $\ell(n) = \text{poly}(n)$. L'idea di base è la seguente. Dato

un seme iniziale $x \in \{0,1\}^n$, questo è usato in $G(\cdot)$ per produrre $n+1$ bit pseudocasuali. Di questi, un bit è usato come output e gli altri n bit sono usati come nuovo seme per $G(\cdot)$. La procedura può essere iterata per ottenere un numero arbitrario di bit pseudocasuali. Intuitivamente, il motivo per cui parte dell'output può essere usata come nuovo seme è che tali bit sono pseudocasuali, e quindi praticamente indistinguibili da n bit veramente casuali. Tale costruzione è molto elegante, ma poco pratica in termini d'efficienza.

Blum-Blum-Shab. Esistono soluzioni più efficienti. Una delle più famose è il generatore Blum-Blum-Shab [BBS86], basato sulla difficoltà del problema della fattorizzazione di certi interi. (Per la comprensione di quanto segue è necessaria un po' di confidenza con alcuni concetti di base in teoria dei numeri, cf. Appendice B.) Sia N un intero della forma $N = p \cdot q$ con $p, q \equiv 3 \pmod 4$ (N è detto un *intero di Blum*). Ciò garantisce che ogni residuo quadratico in $\mathbb{Z}_N^*$ ha una radice quadrata che è a sua volta un residuo quadratico (cf. Lemma B.22). L'idea di base è quella di definire la sequenza di interi:

$$x_{i+1} \equiv x_i^2 \pmod N, \tag{3.6}$$

e prelevare ad ogni passo la parità oppure il bit meno significativo del risultato. Notare che usando il teorema di Eulero (cf. Teorema B.10), ciascun valore x_i può essere calcolato direttamente come

$$x_i \equiv \left(x_0^{2^i \pmod{(p-1)(q-1)}}\right) \pmod N.$$

Questa costruzione elegante è sicura quando si assume *l'ipotesi della residuosità quadratica* (cf. Definizione 6.9). Informalmente, ciò significa ipotizzare che sia impossibile stabilire in tempo polinomiale se $x \in \mathbb{Z}_N^*$ è un residuo quadratico o meno. Come vedremo nel Capitolo 6 in effetti tale problema è del tutto equivalente a quello di fattorizzare N (cf. Teorema 6.4) il che è ritenuto difficile per opportuni valori dei primi p e q (cf. Appendice C.2). Pertanto, se fattorizzare certi interi è difficile (come si crede), il generatore Blum-Blum-Shab genererà bit privi di sequenze ricorrenti che possano essere scoperti efficientemente. Per un'analisi concreta della sicurezza asintotica si veda [SS05].

Esistono una serie di altre costruzioni teoriche, con efficienza variabile, ad esempio basate sull'ipotesi RSA [FS00; SPW06] (cf. Paragrafo 6.2) e sul problema decisionale di Diffie-Hellman [PS98; Gen05; FSS07] (cf. Paragrafo 6.3).

Costruzioni Pratiche. Nel mondo reale si usano costruzioni molto più efficienti, che però hanno lo svantaggio di non avere sicurezza dimostrabile. Si veda [FS03, Capitolo 10] per una panoramica.

3.5 Estrattori di randomicità

Finora abbiamo studiato il concetto di pseudorandomicità. Un approccio alternativo è quello di affidarsi ad alcuni fenomeni fisici la cui natura sembra impredicibile. Purtroppo neanche tali sorgenti sono veramente casuali, ma tipicamente presentano un qualche grado di correlazione (per questo motivo si parla di *sorgenti deboli di randomicità*). Ha senso allora chiedersi se esiste una procedura per *estrarre randomicità*: data una sorgente debole di randomicità vogliamo estrarre da essa quanta più randomicità vera possibile. La risposta a questa domanda ha portato al concetto di *estrattore*.

Estrattori deterministici. Il modo più immediato di definire un estrattore è il seguente:

Definizione 3.7 (Estrattore deterministico). Siano m, n interi tali che $m \leq n$ interi ed $\epsilon > 0$ un parametro. La mappa $\mathsf{Ext} : \{0,1\}^n \to \{0,1\}^m$ è un ϵ-estrattore deterministico per la variabile aleatoria $X \in \{0,1\}^n$, se $\Delta(\mathsf{Ext}(X), U_m) \leq \epsilon$. ■

Non è difficile mostrare che una condizione *necessaria* per estrarre m bit (statisticamente) indistinguibili da m bit veramente casuali, è che per ogni determinazione x della variabile aleatoria X si abbia $\mathbb{P}[X = x] \leq 2^{-m}$. In termini più precisi, è necessario che X abbia entropia minima (cf. Appendice A.3) $\mathbf{H}_\infty(X) \geq m$ (cf. Esercizio 3.11).

Applicando il metodo probabilistico,[21] [AS00] si può dimostrare che, per ogni m, n ed ϵ, esistono ϵ-estrattori in grado di estrarre m bit casuali da ogni sorgente X con entropia minima $m + O(\log n - \log \epsilon)$. Dal punto di vista pratico, però, siamo interessati a costruire un estrattore esplicitamente (possibilmente anche in modo efficiente).

La nozione di estrattore deterministico risale a von Neumann [Neu63]. Consideriamo un processo Bernoulliano con parametro τ, ovvero una sequenza di variabili aleatorie *indipendenti* $X_1, X_2, X_3, \ldots \in \{0,1\}$ tali che per ogni i si

[21] Il metodo probabilistico è una tecnica (non costruttiva) inventata da Paul Erdös per dimostrare l'*esistenza* di alcuni oggetti matematici (ad esempio un grafo con certe proprietà).

abbia $\mathbb{P}[X_i = 1] = \tau$ e $\mathbb{P}[X_i = 0] = 1 - \tau$. Quando il parametro τ soddisfa $\tau_0 \leq \tau \leq 1 - \tau_0$, la distribuzione X è una sorgente di von Neumann con soglia τ_0. Consideriamo la seguente procedura che estrae $m = 1$ bit casuali da n campioni indipendenti prelevati da una sorgente di von Neumann: dividiamo gli n campioni a coppie e scandiamo le coppie una alla volta; quando troviamo una coppia composta da due bit *differenti* ci fermiamo e ritorniamo il primo dei due bit.

Non è difficile accorgersi che in questo modo la probabilità che l'estrattore ritorni 0 oppure 1 è la stessa ed è pari a $\tau(1-\tau)$. Inoltre la probabilità che $n/2$ coppie non abbiano mai due simboli diversi è $(\tau_0^2 + (1 - \tau_0)^2)^{n/2} \leq \epsilon$, quando $\tau_0 \geq \frac{\log(1/\epsilon)}{n}$. L'approccio può essere esteso per estrarre un qualsiasi numero di bit [Per92].

Purtroppo l'ipotesi che i campioni siano completamente indipendenti è molto forte ed, in genere, non verificata nel caso delle sorgenti deboli di randomicità. Santha e Vazirani [SV86] hanno considerato una naturale estensione, in cui si richiede solamente che, per ogni $1 \leq i \leq n$ ed ogni possibile prefisso $x_1, \ldots, x_{i-1} \in \{0,1\}$, si abbia

$$\tau_0 \leq \mathbb{P}[X_i = 1 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 1 - \tau_0.$$

Questa definizione è molto più naturale e sicuramente più vicina a descrivere una sorgente debole di randomicità. Comunque, gli stessi Santha e Vazirani hanno dimostrato che non esiste un estrattore deterministico in grado di estrarre un singolo bit da tale sorgente! Ciò ha portato al concetto di *estrattore con seme*. Menzioniamo, tuttavia, che è possibile costruire estrattori deterministici per altre classi di sorgenti diverse da quelle considerate da Santha e Vazirani. Si vedano [Blu86; TV00; Bou07; Sha11] per alcuni esempi.

Estrattori con seme. Un estrattore con seme [NZ96] assume che si abbia a disposizione una variabile aleatoria uniforme di dimensione d piccola:

Definizione 3.8 (Estrattore con seme). Siano m, n, d interi ed $\epsilon > 0$ un parametro. La mappa $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ è un (k, ϵ)-estrattore con seme, se per ogni variabile aleatoria $X \in \{0,1\}^n$ con $\mathbf{H}_\infty(X) \geq k$ risulta $\Delta((\mathsf{Ext}(X), U_d), U_m) \leq \epsilon$. ■

L'esempio più famoso di estrattore con seme è basato sul concetto di *funzione hash indipendente a coppie*:[22]

[22] Si parla anche di funzioni hash "non crittografiche" per distinguere dalle funzioni hash resistenti alle collisioni, più complesse da realizzare (cf. Capitolo 4).

Definizione 3.9 (Funzioni hash indipendenti a coppie). Una famiglia $\mathcal{H}$ di funzioni $h : \{0,1\}^n \to \{0,1\}^\ell$ è indipendente a coppie se, per ogni coppia $x, y \in \{0,1\}^n$ distinta (cioè tale che $x \neq y$) e per ogni $\alpha, \beta \in \{0,1\}^\ell$, risulta

$$\mathbb{P}\left[h(x) = \alpha \wedge h(y) = \beta : \ h \in \mathcal{H}\right] = \frac{1}{(2^\ell)^2}.$$

■

Notare che se $h(\cdot)$ fosse una funzione puramente casuale il requisito della Definizione 3.9 sarebbe valido in generale: dati N input distinti, la probabilità di ogni singola configurazione sarebbe esattamente $2^{-\ell \cdot N}$. Pertanto, la definizione di indipendenza a coppie è un rilassamento per $L = 2$, e la speranza è che tale requisito possa essere soddisfatto da funzioni molto più semplici da costruire delle funzioni casuali. La nomenclatura "indipendenza a coppie" deriva dal fatto che, quando la Definizione 3.9 è soddisfatta, le variabili aleatorie $Z_x = h(x)$ (per $x \in \{0,1\}^n$) sono indipendenti a coppie, ovvero per ogni $x \neq y$ le variabili aleatorie Z_x e Z_y sono statisticamente indipendenti.

Descriviamo di seguito una costruzione esplicita. Consideriamo la funzione $h_{a,b} : \ \mathbb{Z}_{2^n} \to \mathbb{Z}_{2^n}$ (quindi una permutazione) definita come $h_{a,b}(x) = a \cdot x + b$ con $a, b \in \mathbb{Z}_{2^n}$. Mostriamo che l'insieme $\mathcal{H} = \{h_{a,b}\}$ è indipendente a coppie. Fissiamo una coppia di elementi nel dominio $x, y \in \mathbb{Z}_{2^n}$ (con $x \neq y$) ed una coppia di elementi nel codominio $\alpha, \beta \in \mathbb{Z}_{2^n}$ e consideriamo la probabilità che $h_{a,b}(x) = \alpha$ e $h_{a,b}(y) = \beta$ ovvero

$$\begin{cases} a \cdot x + b = \alpha \\ a \cdot y + b = \beta \end{cases} \quad \Rightarrow \quad \begin{cases} a = (\alpha - \beta) \cdot (x - y)^{-1} \\ b = \alpha - (\alpha - \beta) \cdot (x - y)^{-1} \cdot x. \end{cases}$$

Siccome esiste una sola coppia (a, b) che soddisfa tale sistema, ogni configurazione è equiprobabile:

$$\mathbb{P}\left[h_{a,b}(x) = \alpha \wedge h_{a,b}(y) = \beta : \ a, b \xleftarrow{\$} \mathbb{Z}_{2^n}\right] = \frac{1}{2^{2n}}.$$

Non è complesso verificare che per ogni $\ell < n$ la troncatura ai primi ℓ bit definisce un insieme di funzioni hash indipendenti a coppie da $\{0,1\}^n$ a $\{0,1\}^\ell$.

Impagliazzo, Levin e Luby [ILL89] hanno mostrato come usare una funzione hash indipendente a coppie per costruire un estrattore con seme. Tale costruzione è quasi ottima per lunghezze del seme non troppo piccole.

Lemma 3.8 (Lemma dell'hash residuo). *Se la famiglia $\mathcal{H}$ di funzioni $h : \{0,1\}^n \to \{0,1\}^\ell$ è indipendente a coppie, con $\ell = k - 2\log(1/\epsilon) - O(1)$, allora $\mathsf{Ext}(x, h) = (h, h(x))$ è un $(k, \epsilon/2)$-estrattore con seme.*

Con un piccolo abuso di notazione il valore h restituito dall'estrattore indica l'*intero* corrispondente all'indice della funzione $h \in \mathcal{H}$; quindi la lunghezza del seme è $d = \log(\#\mathcal{H})$ e si prende $\#\mathcal{H}$ una potenza di 2. D'altra parte la lunghezza dell'output $m = d + \ell$ è molto vicina a $k + d$ per ϵ costante, così che l'estrattore estrae praticamente tutta la randomicità contenuta in X.

Dimostrazione. Indicheremo con H la variabile aleatoria relativa all'indice $h \in \mathcal{H}$. La dimostrazione usa il concetto di *probabilità di collisione*. Data una variabile aleatoria X la sua probabilità di collisione è $Col(X) = \sum_{x \in \mathcal{X}} \mathbb{P}[X = x]^2$, ovvero la probabilità che per $x, x' \xleftarrow{\$} \mathcal{X}$ si abbia $x = x'$. Per prima cosa mostreremo che $Col(\mathsf{Ext}(X, H))$ è piccola. Quindi, useremo questo fatto per mostrare che l'output dell'estrattore è statisticamente vicino alla distribuzione uniforme U_m, i.e. $\Delta(\mathsf{Ext}(X, H), U_m) \leq \epsilon/2$.

La prima osservazione è che siccome $\mathbf{H}_\infty(X) \geq k$ abbiamo $Col(X) \leq 2^{-k}$. Se infatti indichiamo con $p_{\max}$ la massima probabilità $\mathbb{P}[X = x]$, possiamo scrivere:

$$Col(X) = \sum_{x \in \mathcal{X}} \mathbb{P}[X = x]^2 \leq p_{\max} \sum_{x \in \mathcal{X}} \mathbb{P}[X = x] = p_{\max} = \frac{1}{2^k}.$$

Quindi

$$\begin{aligned}
Col(\mathsf{Ext}(X, H)) &= \mathbb{P}\left[(h, h(x)) = (h', h'(x')) : \ x, x' \xleftarrow{\$} X, h, h' \xleftarrow{\$} H\right] \\
&= \mathbb{P}[h = h'] \cdot \mathbb{P}[(h, h(x)) = (h', h'(x')) \mid h = h'] \\
&= \mathbb{P}[h = h'] \cdot \mathbb{P}[h(x) = h(x')] \\
&= \mathbb{P}[h = h'] \cdot \Big(\mathbb{P}[x = x'] \\
&\qquad + \mathbb{P}[x \neq x'] \cdot \mathbb{P}[h(x) = h(x') \mid x \neq x']\Big) \\
&\leq \left[\text{siccome } Col(X) \leq 2^{-k}\right] \\
&\leq \frac{1}{2^d} \cdot \left(\frac{1}{2^k} + \mathbb{P}[h(x) = h(x') \mid x \neq x']\right) \\
&\leq [\text{siccome i valori } h \text{ sono indipendenti a coppie}] \\
&\leq \frac{1}{2^d}\left(\frac{1}{2^k} + \frac{1}{2^\ell}\right) \\
&= [\text{usando } k - \ell = 2\log(1/\epsilon) + O(1)] \\
&= \frac{1}{2^{d+\ell}}\left(\frac{1}{2^{2\log(1/\epsilon)+O(1)}} + 1\right)
\end{aligned}$$

$$\leq \frac{\epsilon^2+1}{2^{d+\ell}}. \tag{3.7}$$

Sia ora V_m una variabile aleatoria arbitraria su $\{0,1\}^m$. Useremo il seguente fatto generale relativo alla distanza Euclidea $\|V_m - U_m\|_2^2$ (cf. Eq. (A.4)):

$$\begin{aligned}
\|V_m - U_m\|_2^2 &= \sum_{u\in\{0,1\}^m} \left(\mathbb{P}[V_m = u] - \mathbb{P}[U_m = u]\right)^2 \\
&= \sum_{u\in\{0,1\}^m} \mathbb{P}[V_m = u]^2 + \sum_{u\in\{0,1\}^m} \mathbb{P}[U_m = u]^2 \\
&\qquad - 2\sum_{u\in\{0,1\}^m} \mathbb{P}[V_m = u]\cdot\mathbb{P}[U_m = u] \\
&= Col(V_m) + \frac{1}{2^{d+\ell}} - \frac{2}{2^{d+\ell}} = Col(V_m) - \frac{1}{2^{d+\ell}}.
\end{aligned}$$

Quindi scegliendo $V_m = (H, h(X))$, abbiamo trovato:

$$\begin{aligned}
\|(H,h(X)),U_m\|_2^2 &= Col((H,h(X))) - \frac{1}{2^{d+\ell}} \\
&\leq [\text{sostituendo l'Eq. (3.7)}] \\
&\leq \frac{\epsilon^2+1}{2^{d+\ell}} - \frac{1}{2^{d+\ell}} = \frac{\epsilon^2}{2^{d+\ell}}.
\end{aligned}$$

Infine, usando il fatto che per ogni $\mathbf{v} \in \mathbb{R}^m$ risulta $\|\mathbf{v}\|_1 \leq \sqrt{m}\|\mathbf{v}\|_2$ (cf. Appendice A, Eq. (A.5)) concludiamo:

$$\begin{aligned}
\Delta((H,h(X)),U_m) &= \frac{1}{2}\|(H,h(X)) - U_m\|_1 \\
&\leq \frac{1}{2}\sqrt{2^{d+\ell}}\|(H,h(X)) - U_m\|_2 \\
&\leq \frac{1}{2}\sqrt{2^{d+\ell}}\cdot\sqrt{\frac{\epsilon^2}{2^{d+\ell}}} = \frac{\epsilon}{2}.
\end{aligned}$$

□

Applicando nuovamente il metodo probabilistico, si può dimostrare che per ogni n, k, ϵ esiste un (k,ϵ)-estrattore con seme di lunghezza $\log(n-k)+2\log(\epsilon)+O(1)$ che ritorna $m = k+d-2\log(1/\epsilon)-O(1)$ bit. Diversi lavori in letteratura tentano di costruire esplicitamente tali estrattori [Lu+03; GUV09; Dvi+09].

Estrattori a sorgenti multiple. Siccome per usare un estrattore con seme abbiamo comunque bisogno di una piccola stringa casuale, potremmo dire di aver solo spostato il problema senza averlo risolto veramente. Un modo possibile per rilassare questo requisito forte è, ad esempio, assumere che il seme non sia completamente casuale, ma abbia abbastanza entropia minima. In particolare, quando tale entropia minima è la stessa contenuta in X, ciò equivale a considerare un estrattore deterministico che utilizza due (o più) sorgenti indipendenti [CG88; BIW06; Raz05; Rao09; Bar+10].

Definizione 3.10 (Estrattore a sorgenti multiple). Siano n, s, m, k interi ed ϵ un parametro. La mappa $\mathsf{Ext} : (\{0,1\}^n)^s \to \{0,1\}^m$ è un estrattore-(k, ϵ) ad s sorgenti, se per ogni sequenza di s variabili aleatorie indipendenti $X_i \ldots, X_s$ che soddisfano $\mathbf{H}_\infty(X_i) \geq k$ (per ogni $i = 1, \ldots, s$), risulta $\Delta(\mathsf{Ext}(X_1, \ldots, X_s), U_m) \leq \epsilon$. ■

Il caso di due sorgenti è di particolare interesse in quanto assumere che due sorgenti deboli di randomicità diverse siano indipendenti, descrive bene quello che plausibilmente avviene in realtà. Applicando il metodo probabilistico, è possibile mostrare che esistono estrattori a sorgenti multiple ad esempio con $s = 2$ e $k = O(\log n + \log(1/\epsilon))$. Tuttavia un raggiungimento esplicito di questo limite è ben lontano. Nel 1985, Chor e Goldreich hanno mostrato che il prodotto scalare modulo due $\mathsf{Ext}(x, y) = \sum_{i=1}^{n} x_i \cdot y_i \pmod 2$ è un estrattore a due sorgenti con errore piccolo, a patto che k sia abbastanza più grande di $n/2$. Nel 2005, Bourgain [Bou05] ha migliorato la soglia di entropia a $k = (1/2 - \alpha)n$ per una piccola costante α.

In effetti non è complesso vedere che il problema di costruire un estrattore a due sorgenti è equivalente a quello di costruire alcuni grafi — detti grafi di Ramsey — per cui i metodi attualmente noti non sono affatto soddisfacenti [Gop06].

Due esempi di estrattori a due sorgenti sono indicati di seguito:

- *Estrattore di Hadamard.* Sia $\mathbb{F}$ un campo finito. Si può mostrare [CG88; Rao07] che per ogni $\delta > 0$ la mappa $\mathsf{Ext}_{\mathsf{Had}} : \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ definita come $\mathsf{Ext}_{\mathsf{Had}}(L, R) = \langle L, R \rangle$, essendo $\langle \cdot, \cdot \rangle$ il prodotto scalare in $\mathbb{F}$, è un $(k_{\mathsf{Had}}, k_{\mathsf{Had}}, \epsilon_{\mathsf{Had}})$-estrattore a due sorgenti per

$$k_{\mathsf{Had}} > \left(\frac{1}{2} + \delta\right) n \log(\#\mathbb{F}) \qquad \epsilon_{\mathsf{Had}} = (\#\mathbb{F})^{(n+1)/2} 2^{-k_{\mathsf{Had}}},$$

  con $\#\mathbb{F} = \Omega(n)$.

- *Estrattore DL.* Sia p un primo, g un generatore di $\mathbb{Z}_p^*$ e $q > 1$ un intero che divide $p-1$. La mappa $\mathsf{Ext}_{\mathsf{DL}} : \mathbb{Z}_p \times \mathbb{Z}_p \to \mathbb{Z}_q$ definita come $\mathsf{Ext}_{\mathsf{DL}}(L, R) = \log_g(L - R) \pmod q$ è un $(k_{\mathsf{DL}}, k_{\mathsf{DL}}, \epsilon_{\mathsf{DL}})$-estrattore a due sorgenti per

$$k_{\mathsf{DL}} > \left(\frac{\lfloor \log p + 1 \rfloor}{2} \right) + \log \epsilon_{\mathsf{DL}}^{-1} + \log q \qquad \epsilon_{\mathsf{DL}} > 0.$$

  Osserviamo che per calcolare questo estrattore è necessario valutare logaritmi discreti in $\mathbb{Z}_p$, il che solitamente è infattibile (cf. Appendice C.3). Per una scelta accurata dei parametri p e q, tuttavia, ciò può essere fatto in modo efficiente; si veda [CG88] per i dettagli.

Esercizi

Esercizio 3.1. Consideriamo il problema della cena organizzata dal Cappellaio Matto, descritto all'inizio del capitolo. Qual è la configurazione che massimizza, nei due tavoli, il numero di coppie di invitati che non si conoscono? Quali sono le prestazioni dell'algoritmo probabilistico che assegna gli invitati a caso? E quelle dell'algoritmo deterministico che assegna il prossimi invitato al tavolo dove il numero di coppie di sconosciuti è minima? Giustificare la risposta.

Esercizio 3.2. Siano $X = \{X_n\}_{n\in\mathbb{N}}$ ed $Y = \{Y_n\}_{n\in\mathbb{N}}$ due insiemi di distribuzioni.

1. Dimostrare che X ed Y sono statisticamente ϵ-indistinguibili, se e solo se per ogni insieme $\mathcal{S} \subseteq \{0,1\}^*$, si ha

$$\Delta_{\mathcal{S}}(n) = |\mathbb{P}[X_n \in \mathcal{S}] - \mathbb{P}[Y_n \in \mathcal{S}]| \leq \epsilon(n).$$

   (Suggerimento: dimostrare che $\Delta(X_n, Y_n)$ *è uguale a* $\max_{\mathcal{S}} \Delta_{\mathcal{S}}(n)$*.)*

2. Usare il risultato al punto precedente per mostrare che se X ed Y sono statisticamente indistinguibili, allora sono anche computazionalmente indistinguibili. *(Suggerimento: per ogni funzione* $f : \{0,1\}^* \to \{0,1\}$*, definire* $\mathcal{S}_f = \{x : f(x) = 1\}$*.)*

Esercizio 3.3. Siano $X = \{X_n\}_{n\in\mathbb{N}}$ ed $Y = \{Y_n\}_{n\in\mathbb{N}}$ due insiemi di distribuzioni computazionalmente indistinguibili. Dimostrare che per ogni funzione PPT f, gli insiemi di distribuzioni $X' = \{f(X_n)\}_{n\in\mathbb{N}}$ ed $Y' = \{f(Y_n)\}_{n\in\mathbb{N}}$ sono computazionalmente indistinguibili.

Esercizio 3.4. Mostrare che un attaccante computazionalmente illimitato può sempre violare la Definizione 3.2. (In altri termini, nessun PRG può avere sicurezza incondizionata.)

Esercizio 3.5. Consideriamo il seguente esperimento mentale, che definisce la sicurezza di un generatore pseudocasuale $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ contro un attaccante PPT $\mathscr{A}$. Si sceglie un seme casuale $x \xleftarrow{\$} \{0,1\}^n$ e si calcola $y_0 = G(x)$. Quindi si estrae un valore casuale $y_1 \xleftarrow{\$} \{0,1\}^{\ell(n)}$. Per un valore casuale $b \xleftarrow{\$} \{0,1\}$, il valore y_b è mostrato all'avversario, che deve indovinare b. Diciamo che G è (t,ϵ)-sicuro se nessun avversario PPT $\mathscr{A}$ eseguibile in tempo t ha vantaggio maggiore di $1/2 + \epsilon$ nell'esperimento di cui sopra. Mostrare che questa definizione è equivalente alla Definizione 3.2.

Esercizio 3.6. Sia $G_1 : \{0,1\}^n \to \{0,1\}^{2n}$ un PRG che raddoppia la lunghezza, ed indichiamo con $(y, y') = G_1(x)$ le due metà dell'output di G_1. Consideriamo ora la funzione $G_2 : \{0,1\}^n \to \{0,1\}^{4n}$, definita come $G_2(x) = (G_1(y), G_1(y'))$. In generale possiamo considerare $G_r : \{0,1\}^n \to \{0,1\}^{r \cdot n}$, definito come $G_r(x) = (G_{r-1}(y), G_{r-1}(y'))$. Supponendo che G_1 sia (t, ϵ)-sicuro, determinare se G_r è un PRG e con che parametri.

Esercizio 3.7. Dimostrare che l'esistenza dei generatori pseudocasuali con fattore di espansione $\ell(n) = 2n$ implica l'esistenza delle funzioni unidirezionali. *(Suggerimento: definire $f(x, y) = G(x)$, dove $|x| = |y|$.)*

Esercizio 3.8. Siano f, g due funzioni unidirezionali.

1. Dire se $f'(x_1||x_2) = f(x_1)||x_2$ è unidirezionale (per $|x_1| = |x_2|$).
2. Dire se $g'(x_1||x_2) = x_1||g(x_2)$ è unidirezionale (per $|x_1| = |x_2|$).
3. Dire se $h(x) = f(x)||g(x)$ è unidirezionale.

Esercizio 3.9. Sia f una funzione unidirezionale che preserva la lunghezza. Indichiamo con $\chi(i, \mathbf{x}) = x_i$, l'$i$-esimo bit di $\mathbf{x}$ (per ogni $1 \leq i \leq |\mathbf{x}|$). Dimostrare che la funzione

$$f'(\mathbf{x}) = f(\mathbf{x})||\chi(1, \mathbf{x})||1$$

è unidirezionale, ma che la funzione $\chi(1, \cdot)$ non è un predicato estremo di f'.

Esercizio 3.10. Date due distribuzioni D, D', consideriamo la seguente distanza

$$\Delta_t(D, D') = \max_{\mathscr{A}} \left| \mathbb{P}\left[\mathscr{A}(x) = 1 : \ x \xleftarrow{\$} D\right] - \mathbb{P}\left[\mathscr{A}(x') = 1 : \ x' \xleftarrow{\$} D'\right] \right|,$$

dove il massimo è preso su tutti gli algoritmi $\mathscr{A}$ con tempo di esecuzione t. distribuzioni arbitrarie. Osserviamo che D, D' sono (t, ϵ)-indistinguibili se e solo se $\Delta_t(D, D') \leq \epsilon$.

1. Sia $\mathscr{A}'$ un algoritmo randomizzato con tempo d'esecuzione t' ed indichiamo con $\mathscr{A}'(D)$ (risp. $\mathscr{A}'(D')$) la distribuzione ottenuta prendendo $x \xleftarrow{\$} D$ (risp. $x' \xleftarrow{\$} D'$) e calcolando $\mathscr{A}'(x)$ (risp. $\mathscr{A}'(x')$). Mostrare che $\Delta_t(\mathscr{A}'(D), \mathscr{A}'(D')) \leq \Delta_{t+t'}(D, D')$.
2. Mostrare che per ogni distribuzione D'' si ha

$$\Delta_t(D, D'') \leq \Delta_t(D, D') + \Delta_t(D', D'').$$

3. Indichiamo con $t = \infty$ il caso in cui $\mathscr{A}$ non è limitato computazionalmente. Mostrare che $\Delta_\infty(D, D') = \Delta(D, D')$, essendo $\Delta(D, D')$ la distanza statistica tra D e D'. Mostrare che per ogni t, si ha $\Delta_t(D, D') \leq \Delta_\infty(D, D')$.

4. Sia $\mathcal{E}$ un evento, ed indichiamo con $D|\mathcal{E}$ la distribuzione di probabilità ottenuta condizionando sull'evento $\mathcal{E}$, ovvero $\mathbb{P}[X = x : x \xleftarrow{\$} D|\mathcal{E}] = \mathbb{P}[X = x|\mathcal{E} : x \xleftarrow{\$} D]$. Mostrare che,

$$\Delta_t(D, D') \leq \Delta_t(D|\mathcal{E}, D'|\mathcal{E}) + \mathbb{P}\left[\overline{\mathcal{E}}\right].$$

Esercizio 3.11. Mostrare che è impossibile estrarre $m \leq n$ bit da $X \in \{0,1\}^n$, se $\mathbf{H}_\infty(X) < n$.

Esercizio 3.12. Mostrare che per ogni variabile aleatoria $X \in \{0,1\}^n$ ed ogni funzione $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, se $\mathsf{Ext}(X, U_d)$ è ϵ-vicina alla distribuzione uniforme, allora x è $O(\epsilon)$-vicina ad una variabile aleatoria X' tale che $\mathbf{H}_\infty(X') \geq m - d - 1$.

Esercizio 3.13. La mappa $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ è un estrattore-(k, ϵ) di Rényi, se per ogni variabile aleatoria $X \in \{0,1\}^n$ con entropia di Rényi almeno k (cf. Definizione A.4), risulta che $\mathsf{Ext}(X, U_d)$ ha entropia di Rényi almeno $m - \epsilon$. Dimostrare che un tale estrattore è anche un $(k, O(\sqrt{\epsilon}))$-estrattore con seme secondo la Definizione 3.10.

Letture consigliate

[AS00] John Alon e Joel H. Spencer. *The Probabilsitic Method*. John Wiley e Sons, 2000.

[Bar+10] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov e Avi Wigderson. "Simulating Independence: New Constructions of Condensers, Ramsey Graphs, Dispersers, and Extractors". In: *J. ACM* 57.4 (2010).

[BBS86] Lenore Blum, Manuel Blum e Mike Shub. "A Simple Unpredictable Pseudo-Random Number Generator". In: *SIAM J. Comput.* 15.2 (1986), pp. 364–383.

[BIW06] Boaz Barak, Russell Impagliazzo e Avi Wigderson. "Extracting Randomness Using Few Independent Sources". In: *SIAM J. Comput.* 36.4 (2006), pp. 1095–1118.

[Blu86] Manuel Blum. "Independent unbiased coin flips from a correlated biased source-a finite stae Markov chain". In: *Combinatorica* 6.2 (1986), pp. 97–108.

[Bou05] Jean Bourgain. "More on the sum-product phenomenon in prime fields and its applications". In: *International Journal of Number Theory* 1 (2005), pp. 1–32.

[Bou07] Jean Bourgain. "On the construction of affine extractors". In: *Geometric And Functional Analysis,* 17 (2007), pp. 33–57.

[CG88] Benny Chor e Oded Goldreich. "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity". In: *SIAM J. Comput.* 17.2 (1988), pp. 230–261.

[Dvi+09] Zeev Dvir, Swastik Kopparty, Shubhangi Saraf e Madhu Sudan. "Extensions to the Method of Multiplicities, with Applications to Kakeya Sets and Mergers". In: *FOCS*. 2009, pp. 181–190.

[FS00] Roger Fischlin e Claus-Peter Schnorr. "Stronger Security Proofs for RSA and Rabin Bits". In: *J. Cryptology* 13.2 (2000), pp. 221–244.

[FS03] Niels Ferguson e Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, 2003.

[FSS07] Reza Rezaeian Farashahi, Berry Schoenmakers e Andrey Sidorenko. "Efficient Pseudorandom Generators Based on the DDH Assumption". In: *Public Key Cryptography*. 2007, pp. 426–441.

[Gen05] Rosario Gennaro. "An Improved Pseudo-Random Generator Based on the Discrete Logarithm Problem". In: *J. Cryptology* 18.2 (2005), pp. 91–110.

[GL89] Oded Goldreich e Leonid A. Levin. "A Hard-Core Predicate for all One-Way Functions". In: *STOC*. 1989, pp. 25–32.

[Gol01] Oded Goldreich. *Foundations of Cryptography, Vol. 1: Basic Tools*. Cambridge University Press, 2001.

[Gop06] Parikshit Gopalan. "Constructing Ramsey Graphs from Boolean Function Representations". In: *IEEE Conference on Computational Complexity*. 2006, pp. 115–128.

[GUV09] Venkatesan Guruswami, Christopher Umans e Salil P. Vadhan. "Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes". In: *J. ACM* 56.4 (2009).

[Hal70] Tord Hall. *Carl Friedrich Gauss, a biography*. MIT Press, 1970.

[Hås+99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin e Michael Luby. "A Pseudorandom Generator from any One-way Function". In: *SIAM J. Comput.* 28.4 (1999), pp. 1364–1396.

[IL89] Russell Impagliazzo e Michael Luby. "One-way Functions are Essential for Complexity Based Cryptography (Extended Abstract)". In: *FOCS*. 1989, pp. 230–235.

[ILL89] Russell Impagliazzo, Leonid A. Levin e Michael Luby. "Pseudorandom Generation from one-way functions (Extended Abstract)". In: *STOC*. 1989, pp. 12–24.

[Imp95] Russell Impagliazzo. "A Personal View of Average-Case Complexity". In: *Structure in Complexity Theory Conference*. 1995, pp. 134–147.

[Lu+03] Chi-Jen Lu, Omer Reingold, Salil P. Vadhan e Avi Wigderson. "Extractors: optimal up to constant factors". In: *STOC*. 2003, pp. 602–611.

[Neu63] John von Neumann. "Various Techniques for Use in Connection with Random Digits". In: *von Neumann's Collected Works*. Vol. 5. Pergamon, 1963, pp. 768–770.

[NZ96] Noam Nisan e David Zuckerman. "Randomness is Linear in Space". In: *J. Comput. Syst. Sci.* 52.1 (1996), pp. 43–52.

[Per92] Yuval Peres. "Iterating von Neumann's procedure for extracting random bits". In: *Ann. Statist.* 20 (1992), pp. 590–597.

[PS98] Sarvar Patel e Ganapathy S. Sundaram. "An Efficient Discrete Log Pseudo Random Generator". In: *CRYPTO*. 1998, pp. 304–317.

[Rao07] Anup Rao. "An Exposition of Bourgain's 2-Source Extractor". In: *Electronic Colloquium on Computational Complexity (ECCC)* 14.034 (2007).

[Rao09] Anup Rao. "Extractors for a Constant Number of Polynomially Small Min-Entropy Independent Sources". In: *SIAM J. Comput.* 39.1 (2009), pp. 168–194.

[Raz05] Ran Raz. "Extractors with Weak Random Seeds". In: *STOC*. 2005, pp. 11–20.

[Sha11] Ronen Shaltiel. “Weak Derandomization of Weak Algorithms: Explicit Versions of Yao’s Lemma”. In: *Computational Complexity* 20.1 (2011), pp. 87–143.

[SPW06] Ron Steinfeld, Josef Pieprzyk e Huaxiong Wang. “On the Provable Security of an Efficient RSA-Based Pseudorandom Generator”. In: *ASIACRYPT*. 2006, pp. 194–209.

[SS05] A. Sidorenko e B. Schoenmakers. “Concrete Security of the Blum-Blum-Shub Pseudorandom Generator”. In: *IMA Int. Conf.* 2005, pp. 355–375.

[SV86] M. Santha e U. V. Vazirani. “Generating quasi-random sequences from semi-random sources”. In: *Journal of Computer and Systems Sciences* 33 (1986), pp. 75–87.

[TV00] Luca Trevisan e Salil P. Vadhan. “Extracting Randomness from Samplable Distributions”. In: *FOCS*. 2000, pp. 32–42.

[Yao82] Andrew Chi-Chih Yao. “Theory and Applications of Trapdoor Functions (Extended Abstract)”. In: *FOCS*. 1982, pp. 80–91.

Capitolo 4

Funzioni hash

«Che curiosa sensazione!» disse Alice. «Devo star accorciandomi come un telescopio.» E così era infatti: [...] «Perché potrebbe andare a finire, sai,» si disse «che mi consumi tutta come una candela. E che aspetto avrei allora?» E cercò d'immaginarsi come doveva apparire la fiamma d'una candela dopo che la candela si è estinta, perché ricordava di non aver mai visto una cosa del genere.

Lewis Carroll, Le Avventure di Alice nel Paese delle Meraviglie [Car65]

In questo capitolo ci occupiamo di una famiglia di funzioni il cui ruolo in crittografia è fondamentale: le funzioni *hash*. Una funzione hash è una funzione non iniettiva, che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita. Possiamo interpretare l'hash di una stringa $x \in \{0,1\}^*$ come un'*impronta digitale* da associare alla stringa stessa. In generale una funzione hash prende come input una stringa a lunghezza arbitraria, e la *comprime* in una stringa più corta (tipicamente qualche centinaio di bit). Un'applicazione tipica è nel contesto della memorizzazione di dati nei calcolatori. Data una funzione hash $H(\cdot)$ con codominio $\{0,1\}^n$ si inizializza una tabella a dimensione n, si calcola l'hash della stringa x e si memorizza il risultato nella cella indicizzata da $H(x)$. In questo modo, la struttura dati assicura tempi di lettura e scrittura costanti. La speranza, qui, è che la funzione hash *minimizzi il numero di collisioni* (ovvero distribuisca uniformemente gli elementi nella tabella), in quanto una collisione risulta in due elementi memorizzati nella stessa cella: un numero molto elevato di collisioni renderebbe poco efficiente la lettura di alcuni elementi.

Le funzioni hash hanno gli usi più svariati in crittografia. Ad esempio, esse sono usate nel contesto dell'integrità di messaggio (cf. Capitoli 7 e 8) ed, in generale, nel contesto di ogni primitiva crittografica con sicurezza dimostrabile nel *modello dell'oracolo casuale* (cf. Paragrafo 4.4). Tuttavia, come vedremo, i requisiti sono più stringenti che nel contesto delle strutture dati, in quanto ad esempio non basta minimizzare le collisioni, ma bisogna cercare di evitarle del tutto. La differenza fondamentale è che, nel contesto crittografico, dobbia-

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_4,

mo preoccuparci della Regina Rossa, il cui scopo è proprio quello di trovare collisioni.

> *Come costruire una funzione hash per cui sia "difficile" trovare collisioni?*

Guida per il lettore. Dopo aver definito formalmente i requisiti di sicurezza per una funzione hash (Paragrafo 4.1), descriveremo lo schema di Merkle e Damgård per costruire una funzione hash usando una funzione di compressione (Paragrafo 4.2). Quindi nel Paragrafo 4.3 descriveremo alcune funzioni hash molto usate in pratica. Infine discuteremo le caratteristiche salienti del modello dell'oracolo casuale, che sta assumendo un ruolo sempre più importante e riconosciuto nelle dimostrazioni crittografiche (Paragrafo 4.4).

4.1 Requisiti di sicurezza

In generale considereremo funzioni hash con dominio infinito e codominio finito (o comunque che comprino il loro input). Siccome il codominio è più piccolo del dominio, le collisioni sono inevitabili. (Notare che se non dovessimo comprimere l'input, sarebbe banale trovare funzioni prive di collisioni, ad esempio la funzione identità.) Il nostro scopo è progettare la funzione hash in modo che trovare una collisione sia "difficile". Consideriamo un insieme di funzioni indicizzate da un parametro $s \in \mathcal{S}$ detto chiave:

$$\mathcal{H} = \{H^s(\cdot) : \ s \in \mathcal{S}\},$$

dove $H^s(x) = H(s, x)$. La "chiave" s non è una usuale chiave crittografica. Come vedremo s ha il solo scopo di indicizzare una particolare funzione all'interno di $\mathcal{H}$; in particolare il suo valore *non* deve essere segreto. (Per enfatizzare questo fatto, l'indice s è indicato come apice.)

Definizione 4.1 (Funzione hash). Siano $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{S}$ insiemi arbitrari. Una funzione hash $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}, H)$ è una coppia di algoritmi definiti come segue:

- `Generazioni degli indici.` Dato in ingresso il parametro statistico di sicurezza n restituisce l'indice $s \leftarrow \mathsf{Gen}(1^n)$ (con $s \in \mathcal{S}$), che indicizza una particolare funzione $H^s(\cdot)$ in $\mathcal{H}$. Assumeremo che il valore di n sia implicito in s.
- `Algoritmo di hash.` Data una stringa $x \in \mathcal{X}$ ed una chiave $s \in \mathcal{S}$ restituisce $y = H^s(x) \in \mathcal{Y}$. ■

Come anticipato, la caratteristica principale che cercheremo in una funzione hash è quella di essere *resistente alle collisioni*. Più in generale, possiamo considerare tre diversi attacchi per una funzione hash:

1. *Attacco della pre-immagine.* Dato $s \in \mathcal{S}$ ed $y \in \mathcal{Y}$ con $y = H^s(x)$ per $x \xleftarrow{\$} \mathcal{X}$, la Regina Rossa cerca $x' \in \mathcal{X}$ tale che $H^s(x') = y$.

2. *Attacco della pre-immagine secondaria.* Dato $s \in \mathcal{S}$ ed $x \in \mathcal{X}$, la Regina Rossa cerca $x' \neq x$ tale che $H^s(x') = H^s(x)$.

3. *Ricerca di collisioni.* Dato $s \in \mathcal{S}$, la Regina Rossa cerca una coppia di valori *distinti* $x, x' \in \mathcal{X}$ tali che $H^s(x) = H^s(x')$.

Notare che una funzione hash resistente all'attacco della pre-immagine è, di fatto, una funzione unidirezionale (cf. Definizione 3.4).

Resistenza alle collisioni. Non è difficile mostrare che la resistenza alle collisioni è il requisito più forte per una funzione hash. Fissiamo un indice $s \in \mathcal{S}$. Osserviamo innanzitutto che ogni funzione hash resistente alle collisioni è anche resistente all'attacco della pre-immagine secondaria. Infatti, se fosse possibile trovare una collisione per un x *fissato*, sarebbe possibile anche trovarla per un x *a scelta*. Analogamente si può vedere che una funzione hash resistente all'attacco della pre-immagine secondaria è resistente anche all'attacco della pre-immagine. Infatti, se fosse possibile invertire $y = H^s(x)$ trovando un x' tale che $H^s(x') = y$, allora si potrebbe prendere un valore x, calcolare $y = H^s(x)$ e poi invertire, trovando così x'. Poiché le funzioni hash comprimono l'input, con alta probabilità si avrà $x \neq x'$.

Possiamo quindi preoccuparci solamente del requisito di resistenza alle collisioni, che formalizziamo di seguito.[23] Modelleremo una funzione hash come un *oracolo casuale*. Intuitivamente, ciò significa che l'unico modo per conoscere il valore $H^s(x)$, per un dato $x \in \mathcal{X}$, è chiedere all'oracolo. Più precisamente, assumeremo che, per ogni $x \in \mathcal{X}$, il valore $y = H^s(x)$ è distribuito uniformemente in $\mathcal{Y}$.

Definizione 4.2 (Funzione hash resistente alle collisioni). Sia $n \in \mathbb{N}$ un parametro statistico di sicurezza. Diremo che la funzione hash $\Pi_{\mathsf{HASH}} =$

[23] In realtà, esistono diverse sfumature per le nozioni di resistenza all'attacco della pre-immagine e della pre-immagine secondaria, e non tutte sono implicate dalla resistenza alle collisioni. Si rimanda a [RS04] per una trattazione esaustiva ed uno studio delle relazioni tra queste diverse nozioni.

(Gen, H) è (t, Q, ϵ)-resistente alle collisioni se, per ogni avversario PPT $\mathscr{A}$ eseguibile in tempo t e che effettua Q richieste ad $H(\cdot)$, risulta

$$\mathbb{P}\left[\mathscr{A}^{H^s(\cdot)}(1^n, s) = (x, x') : \ s \leftarrow \mathsf{Gen}(1^n), H^s(x) = H^s(x'), x \neq x'\right] \leq \epsilon(n).$$

■

Complessità degli attacchi a forza bruta. Valutiamo di seguito la complessità e la probabilità di successo degli attacchi a forza bruta contro una funzione hash.

Iniziamo dall'attacco della pre-immagine. Data in input la funzione $H^s(\cdot)$ ed un valore $y \in \mathcal{Y}$, vogliamo trovare $x' \in \mathcal{X}$ tale che $H^s(x') = y$. Sia $\mathcal{Q} \subseteq \mathcal{X}$, un qualsiasi sottoinsieme di $\mathcal{X}$ tale che $\#\mathcal{Q} = Q$. Per ogni valore $x' \in \mathcal{Q}$ calcoliamo $H^s(x')$: se $H^s(x') = y$ restituiamo x', soluzione del problema della pre-immagine. Sia $\mathcal{Y} = \{0,1\}^\ell$. Siccome per ogni stringa x' che proviamo la probabilità di ottenere il valore di y fissato è $1/2^\ell$, la probabilità di fallire Q volte è $(1 - 1/2^\ell)^Q$ e la probabilità media di successo è:

$$\epsilon(Q) = 1 - \left(1 - \frac{1}{2^\ell}\right)^Q \approx \frac{Q}{2^\ell},$$

quando $Q \ll 2^\ell$.

Consideriamo ora l'attacco della pre-immagine secondaria. Dati la funzione $H^s(\cdot)$ ed un valore $x \in \mathcal{X}$ come input, vogliamo trovare $x' \neq x$ tale che $H^s(x') = H^s(x)$. Calcoliamo $y = H^s(x)$ e scegliamo $\mathcal{Q} \subseteq \mathcal{X} \setminus \{x\}$, con $\#\mathcal{Q} = Q - 1$. Per ogni valore $x' \in \mathcal{Q}$ calcoliamo quindi $H^s(x')$: se $H^s(x') = y$ restituiamo x', soluzione del problema della pre-immagine secondaria. Assumiamo ancora $\mathcal{Y} = \{0,1\}^\ell$. Siccome per ogni stringa x' che proviamo la probabilità di ottenere il valore y fissato è $1/2^\ell$, la probabilità di fallire $Q - 1$ volte è $(1 - 1/2^\ell)^{Q-1}$ e la probabilità media di successo

$$\epsilon(Q) = 1 - \left(1 - \frac{1}{2^\ell}\right)^{Q-1} \approx \frac{Q-1}{2^\ell}.$$

Resta da considerare la ricerca di una collisione. In questo caso, data la funzione $H^s(\cdot)$ come input, vorremmo trovare $x, x' \in \mathcal{X}$ (con $x \neq x'$) tali che $H^s(x) = H^s(x')$. Scegliamo $\mathcal{Q} \subseteq \mathcal{X}$, con $\#\mathcal{Q} = Q$. Quindi calcoliamo $y = H^s(x)$ per ogni elemento $x \in \mathcal{Q}$: se esistono due stringhe x, x' tali che $y = y'$ ed $x \neq x'$, la coppia (x, x') è la collisione cercata.

Per analizzare la probabilità di successo di questo algoritmo ricorreremo al *paradosso del compleanno* che richiamiamo qui brevemente. Sia $\mathcal{Y}$ un insieme con $\#\mathcal{Y} = N$ elementi: ci chiediamo qual è la probabilità che, scelti Q elementi a caso in $\mathcal{Y}$, almeno due di essi siano uguali.[24] Sia p_0 la probabilità che *non* si riesca a trovare una collisione e supponiamo $Q \ll N$. Fissati due elementi in $\mathcal{Y}$, questi saranno differenti con probabilità $1 - 1/N$. Fissato un terzo elemento, la probabilità che esso sia diverso dai due precedenti è $1 - 2/N$. Continuando con questo ragionamento otteniamo

$$p_0 = \left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\cdots\left(1 - \frac{Q-1}{N}\right).$$

Calcolando il logaritmo ed usando l'approssimazione $\ln(X+1) \approx X$ per $X \ll 1$ abbiamo[25]

$$\ln(p_0) \approx -\frac{1}{N} - \frac{2}{N} - \cdots - \frac{Q-1}{N} = -\frac{1}{N}\sum_{i=1}^{Q-1} i = -\frac{1}{N}\frac{Q(Q-1)}{2} \approx -\frac{Q^2}{2N}.$$

Quindi $1 - p_0 \approx 1 - e^{-Q^2/2N}$. Se vogliamo probabilità di successo maggiore di $1/2$, dobbiamo porre:

$$\begin{aligned} 1 - p_0 \geq \frac{1}{2} \quad &\Rightarrow \quad 1 - e^{-\frac{Q^2}{2N}} \geq \frac{1}{2} \quad \Rightarrow \quad e^{-\frac{Q^2}{2N}} \leq \frac{1}{2} \\ &\Rightarrow \quad Q \geq \sqrt{2\ln 2}\sqrt{N} \approx 1.17\sqrt{N} = \Theta(\sqrt{N}). \end{aligned}$$

Ne segue che la probabilità di successo media è $\epsilon(Q) \geq 1/2$ quando $O(Q) = O(2^{\ell/2})$.

Abbiamo così dimostrato il seguente:

Teorema 4.1 (Paradosso del compleanno). *Sia $\mathcal{Y}$ un insieme con $\#\mathcal{Y} = N$ elementi. Se gli elementi sono prelevati da $\mathcal{Y}$ a caso, in media troveremo una collisione con probabilità migliore di $1/2$ dopo aver selezionato $Q \approx 1.17\sqrt{N} = \Theta(\sqrt{N})$ elementi.*

[24] Questo è esattamente ciò che succede nell'algoritmo descritto se consideriamo $N = 2^\ell$. Il nome del paradosso deriva del fatto che quando $N = 365$ (il numero di giorni in un anno), stiamo valutando la probabilità che, scelte a caso Q persone, ce ne siano almeno due nate lo stesso giorno. Come vedremo bastano $Q = 23$ persone affinché ciò accada con probabilità superiore ad $1/2$, il che a prima vista può sembrare paradossale.

[25] Si ricordi l'espressione per la serie aritmetica

$$\sum_{i=1}^{Q} i = \frac{Q^2 + Q}{2}.$$

Crittosistema 4.1. La costruzione di Merkle-Damgård $\Pi_{\mathsf{MD}} = (\mathsf{Gen}, H)$

Sia $\Pi_{\mathsf{cmps}} = (\mathsf{Gen}, \mathsf{cmps})$ una funzione di compressione con dominio $\{0,1\}^{2\ell}$ e codominio $\{0,1\}^{\ell}$.

- `Generazione delle chiavi.` L'algoritmo di generazione delle chiavi è lo stesso nelle due costruzioni. Dato il parametro di sicurezza n restituisce l'indice $s \leftarrow \mathsf{Gen}(1^n)$ che indicizza le funzioni $H^s(\cdot)$ e $\mathsf{cmps}^s(\cdot)$ (cf. anche Definizione 4.1).
- `Algoritmo di hash.` Data la chiave s come input ed una stringa $x \in \{0,1\}^*$ di lunghezza $L < 2^{\ell(n)}$, si procede come segue:
  - Sia $B = \lfloor \frac{L}{\ell} \rfloor$ il numero di blocchi lunghi ℓ in x (eventualmente riempire la stringa x sulla destra con tanti 0 quanti sono necessari a rendere la sua lunghezza un multiplo di ℓ). Suddividere x in blocchi, vale a dire $x = x_1 \,||\, \ldots \,||\, x_B$. Porre $x_{B+1} = L$, codificando L come stringa binaria ad ℓ bit.
  - Porre $z_0 = 0^{\ell}$. Per $i = 1, 2, \ldots, B+1$ calcolare $z_i = \mathsf{cmps}^s(z_{i-1} \,||\, x_i)$.
  - Ritornare $H^s(x) = z_{B+1}$.

4.2 La costruzione di Merkle-Damgård

Descriviamo qui la metodologia di Merkle-Damgård[26] per costruire una funzione hash resistente alle collisioni a partire da una *qualsiasi* funzione di compressione per stringhe a lunghezza *fissa* (resistente alle collisioni). Una funzione di compressione per stringhe a lunghezza fissa, mappa stringhe binarie lunghe $\ell'(n) > \ell(n)$ in stringhe binarie lunghe $\ell(n)$. (Nel seguito considereremo il caso $\ell' = 2\ell$, ma quanto detto può essere esteso al caso in cui ℓ' è arbitrario.) Lo schema di Merkle-Damgård consente di estendere il campo d'azione della funzione di compressione $\Pi_{\mathsf{cmps}} = (\mathsf{Gen}, \mathsf{cmps})$ a stringhe di lunghezza arbitraria, ottenendo un funzione hash $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}, H)$ con dominio $\{0,1\}^*$ e codominio $\{0,1\}^{\ell}$. La costruzione di Merkle-Damgård è mostrata nel Crittosistema 4.1 (cf. anche Fig. 4.1). Assumeremo che la lunghezza dell'input sia un multiplo intero di ℓ. Se così non fosse, è sempre possibile operare un "riempimento" (*padding* in inglese) aggiungendo, ad esempio, tanti valori 0 sulla destra fintanto che la

[26] Lo schema è stato scoperto indipendentemente da Ralph Merkle[Mer89] ed Ivan Damgård[Dam89].

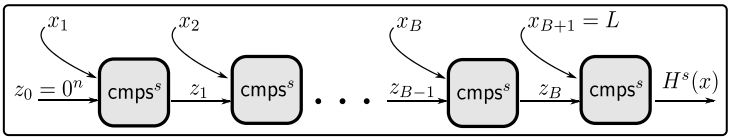


Fig. 4.1. Una rappresentazione grafica della costruzione di Merkle-Damgård

lunghezza non è un multiplo di ℓ.[27] Notare che l'elemento $z_0 = 0^\ell$ in Fig. 4.1, può essere sostituito da un vettore di inizializzazione (*Initialization Vector*, IV) arbitrario. La condizione $L < 2^{\ell(n)}$ è necessaria, in quanto avremo bisogno di codificare L con ℓ bit. Ciò equivale a supporre che i messaggi da comprimere non abbiano lunghezza esponenziale, il che è ragionevole in pratica.

L'intuizione dietro al fatto che Π_{MD} è resistente alle collisioni, risiede nel fatto che se due stringhe x, x' collidono in $H^s(\cdot)$, esse generano una collisione anche in $\mathsf{cmps}^s(\cdot)$.

Teorema 4.2 (Π_{MD} è resistente alle collisioni). *Se Π_{cmps} è una funzione di compressione per stringhe a lunghezza fissa (t, Q, ϵ)-resistente alle collisioni, allora la funzione hash Π_{MD} è (t, Q, ϵ)-resistente alle collisioni.*

Dimostrazione. Come anticipato mostreremo che, per ogni s, una collisione in $H^s(\cdot)$ genera una collisione in $\mathsf{cmps}^s(\cdot)$, contro l'ipotesi che quest'ultima sia resistente alle collisioni. Siano x, x' due stringhe lunghe rispettivamente L, L' bit tali che $H^s(x) = H^s(x')$. Indichiamo con $x_1 \,||\, \ldots \,||\, x_B$ e con $x'_1 \,||\, \ldots \,||\, x'_{B'}$ le suddivisioni in blocchi (eventualmente con riempimento) di x ed x'. Notare che per costruzione si ha $x_{B+1} = L$ ed $x'_{B'+1} = L'$. Distingueremo due casi.

Il caso $L \neq L'$. In questo caso si avrà $z_{B+1} = \mathsf{cmps}^s(z_B||L)$ e $z'_{B'+1} = \mathsf{cmps}^s(z'_{B'}||L')$. Poiché però $H^s(x) = H^s(x')$ deve essere $\mathsf{cmps}^s(z_B||L) = \mathsf{cmps}^s(z'_{B'}||L')$. Ma $L \neq L'$ e quindi $z_B||L$ e $z'_{B'}||L'$ sono due stringhe *distinte* che collidono in $\mathsf{cmps}^s(\cdot)$.

Il caso $L = L'$. In questo caso per costruzione $B = B'$ ed $x_{B+1} = x'_{B+1}$. Siccome $x \neq x'$ ma $|x| = |x'|$ deve esistere almeno un indice $1 \leq i \leq B$ tale che $x_i \neq x'_i$. Sia $i^* \leq B + 1$ l'indice più grande per cui $z_{i^*-1}||x_{i^*} \neq z'_{i^*-1}||x'_{i^*}$. Se

[27] Incontreremo la necessità di usare funzioni di riempimento diverse volte nel testo.

$i^* = B+1$, allora $z_B||x_{B+1}$ e $z'_B||x'_{B+1}$ sono due stringhe *distinte* che collidono in $\mathsf{cmps}^s(\cdot)$ in quanto

$$\begin{aligned}\mathsf{cmps}^s(z_B \mid\mid x_{B+1}) &= z_{B+1} = H^s(x) = H^s(x') \\ &= z'_{B+1} = \mathsf{cmps}^s(z'_B \mid\mid x'_{B+1}).\end{aligned}$$

Se invece $i^* \leq B$, poiché i^* è massimo e siccome deve esistere una collisione, $z_{i^*} = z'_{i^*}$. Quindi ancora una volta $z_{i^*-1}||x_{i^*}$ e $z'_{i^*-1}||x'_{i^*}$ sono due stringhe *distinte* che collidono in $\mathsf{cmps}^s(\cdot)$. □

Sulla costruzione delle funzioni di compressione. Il Teorema 4.2 implica che il problema di costruire una funzione hash resistente alle collisioni (per stringhe a lunghezza arbitraria), può essere ricondotto al problema di costruire una funzione di compressione (per stringhe a lunghezza fissa) resistente alle collisioni. Resta aperta la questione di come costruire tali funzioni di compressione.

Un approccio possibile (che discutiamo di seguito) è quello di utilizzare un cifrario a blocco (cf. Definizione 2.1). Come vedremo nel paragrafo successivo è anche possibile costruire la funzione di compressione senza utilizzare nessun'altra primitiva (tuttavia quest'approccio, in genere, non ha sicurezza dimostrabile). L'idea di usare un cifrario a blocco come funzione di compressione risale a Rabin [Rab78]. In effetti un cifrario a blocco si può ritenere una funzione di compressione in quanto prende come input una chiave ed un testo in chiaro e restituisce un testo cifrato che di solito ha la stessa lunghezza del testo in chiaro. Usare un cifrario a blocco solitamente è meno efficiente (in termini di velocità) che usare una costruzione ad-hoc, ma ha il vantaggio di ottenere sicurezza dimostrabile.

Come visto, la costruzione di Merkle-Damgård ha una struttura iterativa del tipo $z_i = \mathsf{cmps}^s(z_{i-1}||x_i)$. Sia $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ un cifrario a blocco con $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^\ell$. Preneel, Govaerts e Vandewalle [PGV93] hanno

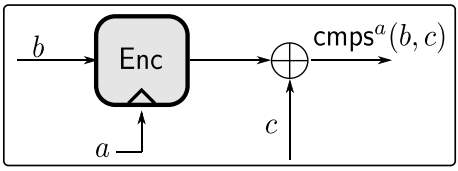


Fig. 4.2. Funzione di compressione a partire da un cifrario a blocco

considerato lo schema seguente (cf. Fig. 4.2):

$$\mathsf{cmps}^a(b,c) = \mathsf{Enc}_a(b) \oplus c \qquad a,b,c \in \{x_i, z_{i-1}, x_i \oplus z_{i-1}, \gamma\},$$

dove γ è una stringa fissata lunga ℓ bit. In pratica è possibile scegliere per l'input e la chiave del cifrario in Fig. 4.2, uno dei quattro possibili valori $x_i, z_{i-1}, x_i \oplus z_{i-1}, \gamma$; uno di questi valori è, inoltre, addizionato all'output del cifrario. Ciò genera $4^3 = 64$ possibili configurazioni. Di queste, 12 sono state congetturate sicure, mentre per le restanti è stato mostrato un attacco esplicito. Successivamente Black, Rogaway e Shrimpton [BRS02] hanno dato prova formale di tale congettura e Stam [Sta08] ha analizzato alcune generalizzazioni. (Si veda anche [Bla+10].) Generalmente ogni dimostrazione di sicurezza per una funzione di compressione basata su un cifrario a blocco ha luogo in un modello ideale detto "modello del cifrario ideale" (*Ideal Cipher Model*). In questo modello, si assume che il cifrario sia scelto uniformemente a caso tra tutti i possibili cifrari (per una data lunghezza della chiave e dell'input) ed inoltre si ipotizza che l'avversario abbia accesso ad un oracolo che restituisca la cifratura (risp. la decifratura) di messaggi (risp. crittotesti) arbitrari a sua scelta.[28] Si veda [Hir06], per un'altra costruzione con sicurezza dimostrabile.

Black, Cochran e Shrimpton [BCS05], hanno mostrato che è impossibile costruire una funzione di compressione usando una sola invocazione del cifrario a blocco usato al suo interno. In generale, un numero minore di invocazioni del cifrario risulta in prestazioni migliori per la funzione hash, ma genera anche un output più corto (il che ovviamente all'estremo non è desiderabile).

Una congettura dovuta a Stam [Sta08] afferma che se una funzione di compressione da $\ell + m$ bit ad ℓ bit fa r chiamate ad una primitiva $f(\cdot)$ con input ad n bit, è possibile trovare una collisione (con alta probabilità) attraverso

$$Q = r \cdot 2^{(rn-m)/(r+1)}$$

invocazioni della primitiva $f(\cdot)$. Il caso $r = 1$ è stato dimostrato da Steinberg [Ste10].

4.3 Funzioni hash nel mondo reale

In questo paragrafo descriviamo alcune costruzioni di funzioni hash molto usate in pratica.

[28] Solitamente queste ipotesi non sono verificate quando si rimpiazza il cifrario ideale con un cifrario a blocco concreto. Pertanto il valore di una prova nel modello della primitiva ideale è simile al valore di una prova nel modello dell'oracolo casuale (cf. Paragrafo 4.4). In effetti i due modelli sono equivalenti [Cor+05; DP06; HKT11].

SHA-1. L'algoritmo SHA-0, acronimo di algoritmo hash sicuro (*Secure Hash Algorithm*, SHA), è stato ideato dal NIST e da NSA nel 1993. Già nel 1995 sono state corrette alcune debolezze che hanno portato all'algoritmo SHA-1 [Nat93]. SHA-1 produce stringhe di 160 bit, con lunghezza dell'input limitata a $2^{64} - 1$ bit.

La prima operazione sull'input x è un riempimento $\mathsf{pad}(\cdot)$ atto a rendere la lunghezza della stringa un multiplo di 512 bit. Sia $d = (447 - |x|) \bmod 512$ e w la stringa binaria ottenuta da x aggiungendo tanti 0 sulla sinistra fino a quando $|w| = 64$ bit. Poniamo

$$u = x \mid\mid \mathsf{pad}(x) = x \mid\mid 1 \mid\mid 0^d \mid\mid w.$$

La lunghezza della stringa u così ottenuta è un multiplo di 512 bit. Possiamo quindi dividere u in L blocchi da 512 bit come in $u = u_1 \mid\mid \ldots \mid\mid u_L$. Definiamo ora le seguenti costanti esadecimali:

$$\begin{aligned} H_0 &= 67452301 & H_1 &= \text{EFCDAB89} \\ H_2 &= 98\text{BADCFE} & H_3 &= 10325476 \\ H_4 &= \text{C3D2E1F0}. \end{aligned}$$

Il resto dell'algoritmo procede in modo iterativo come segue:

1. Per ogni $i = 1, \ldots, L$ dividi ciascun blocco u_i in 16 parole da 32 bit, come in $u_i = w_0 \mid\mid \ldots \mid\mid w_{15}$.

   (a) Per ogni $j = 16, \ldots, 79$ calcola:

   $$w_j \leftarrow \mathsf{ROTL}_1(w_{j-3} \oplus w_{j-8} \oplus w_{j-14} \oplus w_{j-16}),$$

   avendo indicato con $\mathsf{ROTL}_k(x)$ la permutazione circolare a sinistra di k posizioni della stringa x.[29]

   (b) Effettua le sostituzioni:

   $$\begin{aligned} A &\leftarrow H_0 & B &\leftarrow H_1 \\ C &\leftarrow H_2 & D &\leftarrow H_3 \\ E &\leftarrow H_4. \end{aligned}$$

[29] Una permutazione circolare di una stringa, mischia i simboli che la compongono scalandoli verso destra o verso sinistra di un certo numero di posizioni. Ad esempio $\mathsf{ROTL}_2(1234) = 3412$.

(c) Per ogni $j = 0, \ldots, 79$ poni (si veda la parte finale dell'algoritmo per la definizione delle funzioni $f_j(\cdot)$ e delle costanti F_j)

$$\text{tmp} \leftarrow \mathsf{ROTL}_5(A) + f_j(B, C, D) + E + w_j + F_j,$$

(dove + qui è la somma modulo 2^{32}) e permuta le variabili come segue:

$$\begin{aligned} E &\leftarrow D & D &\leftarrow C \\ C &\leftarrow \mathsf{ROTL}_{30}(C) & B &\leftarrow A \\ A &\leftarrow \text{tmp}. \end{aligned}$$

(d) Aggiorna i valori $H_1, \ldots, H_4$:

$$\begin{aligned} H_0 &\leftarrow H_0 + A & H_1 &\leftarrow H_1 + B \\ H_2 &\leftarrow H_2 + C & H_3 &\leftarrow H_3 + D \\ H_4 &\leftarrow H_4 + E. \end{aligned}$$

2. Ritorna $H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4$.

Le funzioni $f_j(\cdot)$ e le costanti F_j sono definite come segue:

$$f_j(B, C, D) = \begin{cases} (B \wedge C) \vee (\overline{B} \wedge D) & \text{se } 0 \le j \le 19 \\ B \oplus C \oplus D & \text{se } 20 \le j \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{se } 40 \le j \le 59 \\ B \oplus C \oplus D & \text{se } 60 \le j \le 79 \end{cases}$$

$$F_j = \begin{cases} \text{5A827999} & \text{se } 0 \le j \le 19 \\ \text{6ED9EBA1} & \text{se } 20 \le j \le 39 \\ \text{8F1BBCDC} & \text{se } 40 \le j \le 59 \\ \text{CA62C1D6} & \text{se } 60 \le j \le 79. \end{cases}$$

Essenzialmente, la funzione calcolata nel ciclo più interno (quello che varia su j) definisce la funzione di compressione utilizzata in SHA-1. Il ciclo più esterno è la sua iterazione come previsto dalla costruzione di Merkle-Damgård.

MD5. L'algoritmo *Message Digest* (MD) fa parte di una serie di algoritmi progettati da Rivest. MD-5 ha rimpiazzato il suo predecessore MD-4 nel 1991.

Tab. 4.1. Confronto tra SHA-1 ed MD-5. La velocità di emissione è misurata implementando le due primitive in codice C++ su un core Celeron ad 850 MHz

Parametro	MD-5	SHA-1
Lunghezza dell'impronta	128 bit	160 bit
Unità di base	512 bit	512 bit
Numero di passi	64 (in 4 round)	80 (in 4 round)
Massima lunghezza dell'input	∞	$2^{64} - 1$ bit
Costanti additive	64	4
Velocità di emissione	26 Mbps	48 Mbps

L'output prodotto è costituito da 128 bit. L'operazione di riempimento è la stessa definita per SHA-1 ed ha lo scopo di rendere la lunghezza del messaggio un multiplo di 512 bit. L'algoritmo lavora su uno stato di 128 bit, diviso in 4 parole da 32 bit. Tali valori sono inizializzati con delle costanti. Lo schema itera la funzione di compressione interna come previsto dalla costruzione di Merkle-Damgård, manipolando lo stato in funzione dei diversi blocchi da 512 bit in $u = x||\mathsf{pad}(x)$. Non descriviamo i dettagli dell'algoritmo, rimandando il lettore interessato direttamente allo standard [Riv92]. La Tab. 4.1 mostra un confronto con SHA-1.

Verso una nuova generazione di funzioni hash. Sono ormai diversi anni che MD-5 è stato violato, pertanto il suo utilizzo è sconsigliato. La caduta è iniziata nel 2004, quando sono state scoperte alcune debolezze, che ne hanno messo in dubbio la sicurezza [Wan+04; BCH06]. L'attacco definitivo è dovuto a Stevens et al. [SLW07; Ste+09]: al costo di 2^{39} chiamate alla funzione di compressione interna di MD-5, è possibile costruire prefissi P, P' e suffissi S, S' tali che le stringhe concatenate $P \,||\, S$ e $P' \,||\, S'$ collidano. La potenza dell'attacco risiede nel fatto che il prefisso può essere scelto arbitrariamente (a differenza degli attacchi precedenti).

Per quanto riguarda SHA, la prima collisione in SHA-0 è stata trovata nel 2004 [Bih+05]. Il lavoro attualmente richiesto per trovare una collisione in SHA-1 è dell'ordine di 2^{53} [WYY05]. Gli stessi principi si applicano ad alcune versioni successive dell'algoritmo (appartenenti alla famiglia SHA-2) che generano impronte digitali lunghe 224, 256, 384 e 512 (contro i 160 bit di SHA-1).

Nel 2007 il NIST ha indetto una competizione per il progetto di un nuovo algoritmo di hash, il cui nome sarà SHA-3 e che dovrebbe essere pronto nel 2012.[30]

4.4 Modello dell'oracolo casuale

Vedremo diverse applicazioni delle funzioni hash nei capitoli successivi, ad esempio nel contesto della confidenzialità (cf. Capitolo 6) e dell'integrità di messaggio (cf. Capitoli 7 e 8). Come sappiamo l'approccio della sicurezza computazionale basa la sicurezza delle primitive crittografiche sull'ipotesi che alcuni problemi computazionali siano intrattabili date le risorse a disposizione di un attaccante. Purtroppo a volte da sola questa ipotesi non è sufficiente (o comunque conduce a costruzioni inefficienti).

Per offrire un piano intermedio tra sicurezza formale ed efficienza si è soliti introdurre un modello idealizzato, in cui le primitive crittografiche possono essere dimostrate sicure. L'esempio più famoso di tale metodologia è il *modello dell'oracolo casuale*, introdotto da Bellare e Rogaway [BR93] nei primi anni 90'. In questo modello si suppone l'esistenza di un oracolo casuale $H(\cdot)$ interrogabile da tutti: dato un input x l'oracolo restituisce il valore $H(x)$ e l'unico modo per calcolare $H(x)$ è inviare x all'oracolo. Si è soliti parlare di *modello standard* quando non è necessario utilizzare un oracolo casuale e quindi una primitiva può essere dimostrata sicura senza tale ipotesi.

Più precisamente nel modello dell'oracolo casuale si fanno le seguenti ipotesi:

1. Si assume che le richieste all'oracolo siano *locali*, nel senso che se qualcuno interroga l'oracolo con input x, la stringa x resta segreta e nessuno è in grado di determinare nemmeno che la richiesta ha avuto luogo.

2. L'oracolo è *consistente*, ovvero se l'oracolo restituisce y in corrispondenza della stringa x, allora restituirà sempre e comunque y quando viene interrogato per x.

3. La funzione $H(\cdot)$ è scelta a caso (una volta per tutte) tra tutte quelle possibili. Supponiamo che $H(\cdot)$ mappi n bit in $\ell(n)$ bit. Una funzione di questo tipo è rappresentabile con una stringa lunga $2^n\ell(n)$ bit e quindi ci sono $2^{2^n\ell(n)}$ possibili mappe. Scegliere $H(\cdot)$ a caso significa scegliere una di queste mappe a caso tra tutte le mappe possibili.

[30] Si veda `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

Notare che, vista la dimensione, è decisamente impossibile fare ciò in pratica. Equivalentemente, è possibile pensare che la funzione $H(\cdot)$ sia costruita "al volo". Data una richiesta (x_i, y_i) si controlla se la stringa x_i è stata già richiesta: in caso affermativo si restituisce la stessa stringa usata in precedenza, altrimenti l'output è scelto a caso in $\{0,1\}^{\ell(n)}$. Sebbene ciò sia del tutto indifferente dal punto di vista di chi interroga l'oracolo, costruire $H(\cdot)$ "al volo" è sensibilmente diverso da supporre che $H(\cdot)$ sia disponibile una volta per tutte quando una primitiva è inizializzata.

4. In una prova per riduzione è possibile "programmare" l'oracolo opportunamente, in modo che i valori restituiti siano utilizzabili in modo conveniente (purché essi appaiano uniformemente casuali).

Incontreremo alcuni esempi concreti di dimostrazioni di sicurezza nel modello dell'oracolo casuale più avanti nel testo.

Critiche e punti a favore. Nella realtà, in fase d'implementazione, dobbiamo sostituire l'oracolo con una funzione concreta $\widehat{H}$, ad esempio SHA-1. Da qui la domanda: *qual è il valore di una dimostrazione di sicurezza nel modello dell'oracolo casuale quando tale schema è utilizzato nel mondo reale?* La risposta è per alcuni versi controversa.

Un celebre risultato negativo dovuto a Canetti, Goldreich ed Halevi [CGH04], mostra che esistono primitive crittografiche (cf. Esercizio 7.6 ed Esercizio 8.10) che sono sicuri nel modello dell'oracolo casuale, ma che sono *in*sicuri *per ogni possibile implementazione concreta* dell'oracolo stesso! Tuttavia, è importante sottolineare che questi schemi sono completamente "artificiali" e non esiste nessun attacco concreto ad un sistema sicuro nel modello dell'oracolo utilizzato in pratica.

Il punto è che non è affatto chiaro cosa significhi in pratica per una funzione reale emulare un oracolo casuale. Ad esempio un oracolo casuale è completamente diverso da una funzione pseudocasuale: mentre la seconda è una funzione con chiave che può essere valutata solo una volta che la chiave è nota e sembra puramente casuale altrimenti, il primo è una funzione senza chiave che chiunque può valutare e che nonostante ciò deve sembrare puramente casuale in un senso che non è ben definibile.

Per tutti questi motivi, una dimostrazione nel modello standard è considerata in genere più valida di una dimostrazione nel modello dell'oracolo casuale. Tuttavia, gli schemi con sicurezza dimostrabile nel modello dell'oracolo casuale sono spesso molto più efficienti; inoltre si tende a pensare che una dimostrazione

nel modello dell'oracolo è sempre meglio che non avere alcuna dimostrazione. Una prova in questo modello idealizzato, infatti, dà un senso di validità allo schema, sottolineando che le uniche sue debolezze sono quelle che possono sorgere rimpiazzando l'oracolo con una funzione concreta. Quindi, se abbiamo modo di ritenere che tale funzione è "sufficientemente buona", possiamo confidare in un certo senso che lo schema sia sicuro. Dal punto di vista teorico sarebbe auspicabile formalizzare il requisito di essere "sufficientemente buona". Si veda [MRH04] per una possibile formalizzazione.

In conclusione usare uno schema che ha una prova formale nel modello dell'oracolo è decisamente meglio che usarne uno che non ha alcuna dimostrazione di sicurezza! Tuttavia se esiste uno schema equivalente sicuro nel modello standard quest'ultimo è preferibile (almeno a parità di efficienza).

Esercizi

Esercizio 4.1. Dare una definizione formale di resistenza all'attacco della pre-immagine secondaria. Quindi, mostrare che ogni famiglia di funzioni hash resistente alle collisioni è anche resistente all'attacco della pre-immagine secondaria.

Esercizio 4.2. Dare una definizione formale di resistenza all'attacco della pre-immagine. Quindi, mostrare che ogni famiglia di funzioni hash resistente all'attacco della pre-immagine secondaria è anche resistente all'attacco della pre-immagine.

Esercizio 4.3. Costruire una funzione hash che sia resistente all'attacco della pre-immagine, ma non resistente alle collisioni.

Esercizio 4.4. Costruire una funzione hash che sia resistente all'attacco della pre-immagine secondaria, ma non resistente alle collisioni.

Esercizio 4.5. Sia $H : \{0,1\}^* \to \{0,1\}^{256}$ una funzione hash tale che:

$$m \equiv m' \pmod{232} \Rightarrow H(m) = H(m').$$

1. Dire quante possibili stringhe di 256 bit hanno una pre-immagine.
2. Dato m, trovare una seconda pre-immagine. Calcolare la complessità.
3. Dato $H(m)$, trovare una pre-immagine. Calcolare la complessità.

Esercizio 4.6. Per una stringa $x = x_1, \ldots, x_n$, la funzione di parità Φ è definita come $\Phi(x) = \bigoplus_{i=1}^{n} x_i$. Sia $H : \{0,1\}^* \to \{0,1\}^{128}$ una funzione hash tale che:

$$\Phi(m) = \Phi(m').$$

1. Dato m, trovare una seconda pre-immagine. Calcolare la complessità.
2. Trovare una collisione. Calcolare in modo approssimato la probabilità di trovare una collisione avendo a disposizione 2^{64} tentativi.

Esercizio 4.7. Siano (Gen_1, H_1) e (Gen_2, H_2) due funzioni hash. Considerare la funzione hash (Gen, H) che lancia Gen_1, Gen_2 (indipendentemente) per ottenere indici s_1, s_2 e calcola l'hash come $H^{s_1,s_2}(m) = H^{s_1}(m)||H^{s_2}(m)$. Stabilire in che condizioni (Gen, H) è resistente alle collisioni, all'attacco della pre-immagine ed all'attacco della pre-immagine secondaria.

Esercizio 4.8. Sia (Gen, H) una funzione hash resistente alle collisioni. Consideriamo la funzione hash $(\mathsf{Gen}, \widehat{H})$ definita come $\widehat{H}^s(m) = H^s(H^s(m))$. Stabilire se $(\mathsf{Gen}, \widehat{H})$ è resistente alle collisioni.

Esercizio 4.9. Considerare le seguenti modifiche della costruzione di Merkle-Damgård e dire se lo schema risultante è sicuro o meno.

1. L'ultimo blocco è omesso, ovvero si ritorna z_B invece che z_{B+1}.
2. Invece di ritornare $\mathsf{cmps}^s(z_B, L)$, si ritorna $z_B||L$.

Esercizio 4.10. Sia Π_{HASH} una famiglia di funzioni hash con dominio $\mathcal{X}$. Consideriamo il seguente esperimento. Si sceglie $s \leftarrow \mathsf{Gen}(1^n)$. L'attaccante ritorna $(x_1, \ldots, x_n, y) \in \mathcal{X}^n \times \mathcal{Y}$ ed ha successo se y divide $\prod_{i=1}^{n} H(x_i)$, ma $y \neq H(x_i)$ per ogni $i = 1, \ldots, n$. Diciamo che Π_{HASH} è (t, Q, ϵ)-*intrattabile per divisione*, se nessun attaccante eseguibile in tempo t che effettua Q richieste, ha successo nell'esperimento con probabilità migliore di ϵ.
Mostrare che se Π_{HASH} è intrattabile per divisione, allora è anche resistente alle collisioni.

Letture consigliate

[BCH06] John Black, Martin Cochran e Trevor Highland. “A Study of the MD5 Attacks: Insights and Improvements”. In: *FSE*. 2006, pp. 262–277.

[BCS05] John Black, Martin Cochran e Thomas Shrimpton. “On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions”. In: *EUROCRYPT*. 2005, pp. 526–541.

[Bih+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet e William Jalby. “Collisions of SHA-0 and Reduced SHA-1”. In: *EUROCRYPT*. 2005, pp. 36–57.

[Bla+10] John Black, Phillip Rogaway, Thomas Shrimpton e Martijn Stam. “An Analysis of the Blockcipher-Based Hash Functions from PGV”. In: *J. Cryptology* 23.4 (2010), pp. 519–545.

[BR93] Mihir Bellare e Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *First ACM Conference on Computer and Communications Security*. Fairfax: ACM, 1993, pp. 62–73.

[BRS02] John Black, Phillip Rogaway e Thomas Shrimpton. “Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV”. In: *CRYPTO*. 2002, pp. 320–335.

[Car65] Lewis Carroll. *Alice’s Adventures in Wonderland*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1865.

[CGH04] Ran Canetti, Oded Goldreich e Shai Halevi. “The random oracle methodology, revisited”. In: *J. ACM* 51.4 (2004), pp. 557–594.

[Cor+05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud e Prashant Puniya. “Merkle-Damgård Revisited: How to Construct a Hash Function”. In: *CRYPTO*. 2005, pp. 430–448.

[Dam89] Ivan Damgård. “A Design Principle for Hash Functions”. In: *CRYPTO*. 1989, pp. 416–427.

[DP06] Yevgeniy Dodis e Prashant Puniya. “On the Relation Between the Ideal Cipher and the Random Oracle Models”. In: *TCC*. 2006, pp. 184–206.

[Hir06] Shoichi Hirose. “Some Plausible Constructions of Double-Block-Length Hash Functions”. In: *FSE*. 2006, pp. 210–225.

[HKT11] Thomas Holenstein, Robin Künzler e Stefano Tessaro. “The equivalence of the random oracle model and the ideal cipher model, revisited”. In: *STOC*. 2011, pp. 89–98.

[Mer89] Ralph C. Merkle. “One Way Hash Functions and DES”. In: *CRYPTO*. 1989, pp. 428–446.

[MRH04] Ueli M. Maurer, Renato Renner e Clemens Holenstein. “Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology”. In: *TCC*. 2004, pp. 21–39.

[PGV93] Bart Preneel, René Govaerts e Joos Vandewalle. “Hash Functions Based on Block Ciphers: A Synthetic Approach”. In: *CRYPTO*. 1993, pp. 368–378.

[Rab78] Michael O. Rabin. “Digitalized Signatures”. In: *Foundations of Secure Computation*. A cura di Richard A. DeMillo, David P. Dobkin, Anita K. Jones e Richard J. Lipton. Academic Press, 1978, pp. 155–168.

[Riv92] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*. Internet Request for Comments. RFC 1321. Apr. 1992.

[RS04] Phillip Rogaway e Thomas Shrimpton. “Cryptographic Hash Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. In: *FSE*. 2004, pp. 371–388.

[SLW07] Marc Stevens, Arjen K. Lenstra e Benne de Weger. “Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities”. In: *EUROCRYPT*. 2007, pp. 1–22.

[Sta08] Martijn Stam. “Beyond Uniformity: Better Security/Efficiency Tradeoffs for Compression Functions”. In: *CRYPTO*. 2008, pp. 397–412.

[Ste+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik e Benne de Weger. “Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate”. In: *CRYPTO*. 2009, pp. 55–69.

[Ste10] John P. Steinberger. “Stam’s Collision Resistance Conjecture”. In: *EUROCRYPT*. 2010, pp. 597–615.

[Wan+04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai e Hongbo Yu. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, Report 2004/199. `http://eprint.iacr.org/`. 2004.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin e Hongbo Yu. “Finding Collisions in the Full SHA-1”. In: *CRYPTO*. 2005, pp. 17–36.

[Nat93] National Institute of Standards and Technology (NIST). *FIPS Publication 180: Secure Hash Standard (SHS)*. 1993.

Capitolo 5

Cifrari simmetrici

Ci sono 10 tipi di persone. Quelli che capiscono i numeri binari e quelli che non li capiscono.

Barzelletta anonima

Come abbiamo accennato nell'introduzione (cf. Paragrafo 1.1), uno dei requisiti fondamentali nel contesto della comunicazione sicura è la *confidenzialità*: Alice ed il Bianconiglio vogliono poter comunicare (scambiando alcuni messaggi) in modo che, seppur la Regina Rossa intercetti i messaggi inviati sul canale insicuro, il contenuto della comunicazione resti in qualche modo "segreto". D'altra parte, vorremmo che il Bianconiglio sia in grado di "interpretare" i messaggi inviati da Alice (e viceversa).

Già sappiamo che un modo per risolvere questo problema è attraverso l'uso di un cifrario simmetrico (cf. Definizione 2.1). Nel Capitolo 2, abbiamo definito il concetto di sicurezza incondizionata per cifrari simmetrici, ed abbiamo incontrato una costruzione esplicita di schema di cifratura incondizionatamente sicuro. Purtroppo, abbiamo anche visto che esistono alcune limitazioni intrinseche relative all'uso di tali schemi, sopratutto in termini di efficienza (cf. Paragrafo 2.3).

In questo capitolo sposteremo la nostra attenzione sul concetto di *sicurezza computazionale* (cf. Definizione 1.4) per cifrari simmetrici, ovvero assumeremo che la Regina abbia risorse computazionali limitate (nel senso definito nel Paragrafo 1.3). Siccome il requisito di sicurezza computazionale è più debole di quello di sicurezza incondizionata, la speranza è quella di guadagnare qualcosa in termini di efficienza. In altri termini ci chiediamo:

> *Supponiamo che Alice ed il Bianconiglio condividano un segreto. Possono comunicare attraverso un canale insicuro, celando il contenuto dei messaggi alla Regina che controlla il canale? Come definire la sicurezza computazionale in quest'ambito?*

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_5, © Springer-Verlag Italia 2012

Guida per il lettore. La struttura del capitolo è quindi la seguente. Nel Paragrafo 5.1 daremo la definizione formale di sicurezza computazionale per cifrari simmetrici e nel Paragrafo 5.2 vedremo alcune costruzioni teoriche di cifrari simmetrici computazionalmente sicuri. Nel Paragrafo 5.3 descriveremo gli standard DES ed AES, due cifrari simmetrici molto usati in pratica.

Un cifrario simmetrico elabora input a lunghezza fissa; per questo motivo i cifrari simmetrici sono detti anche *cifrari a blocco*. Quando vogliamo usare un cifrario simmetrico per cifrare un messaggio a lunghezza arbitraria, non possiamo quindi farlo direttamente. Per risolvere questo problema si utilizza un *modo operativo*: ci occuperemo di questo nel Paragrafo 5.4. Un'alternativa è quella di usare un *cifrario a flusso*, come vedremo nel Paragrafo 5.5.

5.1 Nozioni di sicurezza per cifrari simmetrici

Consideriamo un cifrario simmetrico $\Pi_{\mathsf{SKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ come nella Definizione 2.1. Nel Capitolo 2 abbiamo definito il concetto di sicurezza incondizionata: Π_{SKE} è incondizionatamente sicuro se, per ogni coppia di messaggi $m_0, m_1 \in \mathcal{M}$ e per ogni chiave $k \leftarrow \mathsf{Gen}(1^n)$, la distribuzione delle variabili aleatorie $\mathsf{Enc}_k(m_0)$ ed $\mathsf{Enc}_k(m_1)$ è *identica*. Come visto, la conseguenza fondamentale del teorema di Shannon è che la sicurezza incondizionata richiede di usare un'*unica* chiave per cifrare un *singolo* messaggio; inoltre tale chiave deve essere almeno tanto lunga quanto il messaggio da cifrare. Siccome ciò non è molto efficiente, vorremmo trovare un modo per cifrare un numero arbitrario di messaggi usando *la stessa chiave* k, mantenendo allo stesso tempo un livello accettabile di sicurezza. Inoltre, vorremmo che la chiave fosse possibilmente corta e che si possa utilizzarla per cifrare messaggi arbitrariamente lunghi. Come anticipato, l'idea è quella di limitare il potere computazionale della Regina, passando dal concetto di sicurezza incondizionata a quello della *sicurezza computazionale*. Considereremo quindi solo attacchi eseguibili in tempo polinomiale, ovvero modelleremo la Regina come un algoritmo PPT (cf. Paragrafo 1.3).

Prima di procedere oltre, sottolineiamo due limitazioni inerenti ogni cifrario che usa una chiave a lunghezza n molto più corta della lunghezza dei messaggi che cifra: (i) in tempo 2^n l'attaccante può enumerare tutte le possibili chiavi ed usarle per decifrare il crittotesto in 2^n modi diversi, uno dei quali è quello corretto e (ii) l'avversario può indovinare la chiave con probabilità 2^{-n}. Quello descritto non è altro che l'attacco a forza bruta. Comunque, già quando $n = 128$, nessuna strategia è allarmante per Alice ed il Bianconiglio: anche

disponendo del computer più potente della Terra, Alice ed il Bianconiglio sarebbero già morti di vecchiaia prima che la Regina termini l'enumerazione di tutte le 2^{128} chiavi. Inoltre la probabilità che la Regina indovini la chiave è molto più bassa della probabilità che Alice sia colpita da una meteora!

Indistinguibilità e sicurezza semantica. La prima definizione formale di confidenzialità è dovuta a Goldwasser e Micali [GM84]. Tale definizione usa il concetto di *indistinguibilità* (cf. Definizione 3.1), introducendo un gioco tra la Regina (che attacca il cifrario) ed il Brucaliffo (cf. Fig. 1.2). Informalmente, diremo che un cifrario ha cifrature indistinguibili se, per ogni chiave $k \leftarrow \mathsf{Gen}(1^n)$ e per ogni coppia di messaggi $m_0, m_1 \in \mathcal{M}$ a scelta dell'attaccante, nessun avversario PPT $\mathscr{A}$ può distinguere $\mathsf{Enc}_k(m_0)$ da $\mathsf{Enc}_k(m_1)$ con vantaggio non trascurabile. Più formalmente, dato un cifrario simmetrico $\Pi_{\mathsf{SKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ come nella Definizione 2.1, consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$:
1. $k \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}(1^n)$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_k(m_b)$;
4. $b' \leftarrow \mathscr{A}(c_b)$;
5. restituisci 1 se e solo se $b' = b$ e $|m_0| = |m_1|$.

In pratica l'avversario sceglie due messaggi m_0 ed m_1 basandosi solo sulla conoscenza del parametro di sicurezza n, quindi il Brucaliffo estrae un bit casuale $b \xleftarrow{\$} \{0,1\}$ e cifra il messaggio m_b con la chiave k. La Regina ha successo se riesce ad indovinare il valore del bit b scelto dal Brucaliffo (notare che tale valore può sempre essere indovinato con probabilità $1/2$). (Osserviamo che, in generale, nell'esperimento è necessario controllare che $|m_0| = |m_1|$; altrimenti, se Π_{SKE} è in grado di cifrare messaggi a lunghezza variabile, è banale vincere il gioco ispezionando la lunghezza del crittotesto sfida.) Diremo che il cifrario Π_{SKE} è sicuro, se il valore di b è impredicibile.

Definizione 5.1 (Sicurezza IND per cifrari simmetrici). Diremo che il cifrario Π_{SKE} è IND-(t,ϵ)-sicuro (semanticamente sicuro) se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

La quantità ϵ è anche detta *vantaggio* di $\mathscr{A}$ nell'esperimento $\mathbf{Exp}^{\text{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$. Ovviamente, affinché il cifrario sia sicuro, dobbiamo richiedere che il parametro ϵ sia trascurabile (nel parametro di sicurezza n).

Dobbiamo a questo punto chiederci se è possibile soddisfare questa definizione *senza fare nessun'altra ipotesi*. Purtroppo la risposta a questa domanda è negativa, come mostrato dal seguente:

Lemma 5.1 (Sicurezza IND "pura" $\Rightarrow$ P $\neq$ NP). *Se esistono cifrari semanticamente sicuri per cui $\#\mathcal{K} < \#\mathcal{M}$, allora* $\mathbf{P} \neq \mathbf{NP}$.

Dimostrazione. Osserviamo che se $\#\mathcal{K} \geq \#\mathcal{M}$ esistono cifrari incondizionatamente sicuri, come mostrato da Shannon (cf. Teorema 2.3). Mostriamo che invece quando $\#\mathcal{K} < \#\mathcal{M}$, se $\mathbf{P} = \mathbf{NP}$, un cifrario semanticamente sicuro *non* può esistere. Intuitivamente, il motivo è che se $\mathbf{P} = \mathbf{NP}$ l'avversario può provare tutte le chiavi in tempo polinomiale. Consideriamo il seguente linguaggio:

$$\mathcal{L} = \bigcup_{n \in \mathbb{N}} \left\{ (\mathsf{Enc}_k(m), m) : \ k \in \{0,1\}^n, m \in \{0,1\}^{n+1} \right\},$$

composto da tutte le coppie (c, m) tali che $c \leftarrow \mathsf{Enc}_k(m)$, per $\mathcal{K} = \{0,1\}^n$ ed $\mathcal{M} = \{0,1\}^{n+1}$. Ovviamente $\mathcal{L} \in \mathbf{NP}$ (infatti la chiave k costituisce l'indizio). Se $\mathbf{P} = \mathbf{NP}$, allora $\mathcal{L}$ è anche decidibile in tempo polinomiale.

Consideriamo ora un attaccante $\mathscr{A}$ nell'esperimento $\mathbf{Exp}^{\text{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$. L'avversario sceglie $m_0, m_1 \in \{0,1\}^{n+1}$ e li invia all'oracolo. Quest'ultimo estrae $k \xleftarrow{\$} \{0,1\}^n$ e $b \xleftarrow{\$} \{0,1\}$ ed invia ad $\mathscr{A}$ il messaggio $c_b = \mathsf{Enc}_k(m_b)$. Quindi, l'attaccante decide che $b = 0$ se $(c_b, m_0) \in \mathcal{L}$, altrimenti decide che $b = 1$. L'osservazione chiave è che $\mathscr{A}$ *non* indovina b solo quando $b = 1$ e $(c_b, m_0) \in \mathcal{L}$, il che avviene se esiste k' tale che $\mathsf{Enc}_k(m_1) = \mathsf{Enc}_{k'}(m_0)$. Ma la probabilità che ciò accada è $\leq (\#\mathcal{K})/(\#\mathcal{M}) = 1/2$ e quindi $\mathscr{A}$ viola la sicurezza semantica del cifrario con probabilità $> 1/2$. □

La conseguenza fondamentale del Lemma 5.1 è che, nell'ambito della sicurezza computazionale, dobbiamo accontentarci di dimostrare risultati condizionali del tipo: "se un certo problema è difficile, allora lo schema è sicuro". In realtà, come sarà chiaro in seguito, la maggior parte delle costruzioni crittografiche è basata sull'esistenza delle funzioni unidirezionali (cf. Definizione 3.4), che a sua volta implica $\mathbf{P} \neq \mathbf{NP}$. È un importante problema aperto costruire crittosistemi la cui sicurezza sia basata *direttamente* sulla congettura $\mathbf{P} \neq \mathbf{NP}$ o su una simile congettura in teoria della complessità.

Sicurezza semantica. Abbiamo formalizzato la confidenzialità attraverso il concetto d'indistinguibilità. Restano alcuni punti oscuri: la definizione data va bene per tutte le applicazioni? Perché la Regina sceglie solamente due messaggi? E ancora: perché il Brucaliffo sceglie uno di essi esattamente con probabilità 1/2 (ovvero, cosa cambia se b non è uniforme)? Motivati da queste osservazioni, Goldwasser e Micali [GM84] hanno dato anche un'altra definizione (apparentemente) molto più generale di sicurezza, detta *sicurezza semantica*. Non daremo la definizione formale, ci basti sapere che tale nozione è abbastanza generale da catturare ogni possibile scenario d'interesse. Essenzialmente essa dice che la conoscenza del crittotesto non è di alcun aiuto ad un attaccante: tutto ciò che un attaccante può ricavare dato il testo cifrato e l'informazione a priori su di esso (ad esempio la lingua in cui il testo è scritto), può essere "simulato" in tempo polinomiale *senza avere accesso al crittotesto stesso*. (Questo paradigma della "simulazione" è usato spesso in crittografia, si vedano ad esempio i Capitoli 13 e 14.)

Qualche anno più tardi Micali, Rackoff e Sloan [MRS88] hanno mostrato che la Definizione 5.1 (ed un'altra definizione data da Yao [Yao82]) *è in realtà completamente equivalente* a quella di sicurezza semantica. Possiamo quindi tranquillamente lavorare con la più semplice delle due, in quanto dimostrare che un cifrario è IND-sicuro implica anche che esso è semanticamente sicuro.

Sicurezza contro attacchi a messaggio scelto. Nello scenario di attacco che abbiamo considerato nella Definizione 5.1, la Regina si limita a scegliere due messaggi e deve quindi riuscire a distinguerne la cifratura. In un attacco a messaggio scelto (*Chosen Plaintext Attack*, CPA) doteremo l'avversario di più potere, dandogli accesso ad un *oracolo di cifratura*. Consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$:
1. $k \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}^{\mathsf{Enc}_k(\cdot)}(1^n)$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_k(m_b)$;
4. $b' \leftarrow \mathscr{A}^{\mathsf{Enc}_k(\cdot)}(c_b)$;
5. restituisci 1 se e solo se $b' = b$ e $|m_0| = |m_1|$.

In pratica, la differenza con la definizione precedente è che, sia prima che dopo aver scelto i messaggi $m_0, m_1 \in \mathcal{M}$, la Regina può sceglierne altri a suo piacere e richiedere di vedere la corrispondente cifratura sperando di imparare qualcosa che l'aiuti poi ad indovinare il bit b, nascosto nel crittotesto sfida. Notare che la

scelta dei messaggi inviati all'oracolo di cifratura può essere *adattiva*, nel senso che la scelta del prossimo messaggio può dipendere dalle risposte ricevute nelle richieste precedenti.

Definizione 5.2 (Sicurezza CPA per cifrari simmetrici). Diremo che il cifrario Π_{SKE} è CPA-(t, Q, ϵ)-sicuro se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t che effettua Q richieste all'oracolo di cifratura, risulta:

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

Osserviamo che *un cifrario deterministico non può mai essere CPA-sicuro*. Infatti, se Π_{SKE} è deterministico, la Regina può scegliere m_0, m_1 arbitrariamente, ed indovinare b semplicemente richiedendone la cifratura al Brucaliffo: siccome il cifrario è deterministico, uno dei crittotesti ottenuti sarà uguale al crittotesto sfida c_b. Ne segue che un cifrario CPA-sicuro *deve essere randomizzato*.

Sicurezza contro attacchi a crittotesto scelto. In un attacco a crittotesto scelto (*Chosen Chipertext Attack*, CCA), la Regina ha anche accesso ad un *oracolo di decifratura*. Ciò rappresenta la possibilità che, nella realtà, un avversario può imparare il testo in chiaro corrispondente ad alcuni testi cifrati. Consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$:

1. $k \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}^{\mathsf{Enc}_k(\cdot),\mathsf{Dec}_k(\cdot)}(1^n)$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_k(m_b)$;
4. $b' \leftarrow \mathscr{A}^{\mathsf{Enc}_k(\cdot),\mathsf{Dec}_k(\cdot)}(c_b)$;
5. restituisci 1 se e solo se:
   (i) il crittotesto c_b non è stato mai inviato a $\mathsf{Dec}_k(\cdot)$;
   (ii) $|m_0| = |m_1|$;
   (iii) $b' = b$.

In pratica la differenza con l'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$ è che l'attaccante, oltre a mantenere accesso all'oracolo di cifratura $\mathsf{Enc}_k(\cdot)$, può anche inviare testi cifrati $c \in \mathcal{C}$ all'oracolo di decifratura $\mathsf{Dec}_k(\cdot)$, ottenendo come risposta il corrispondente testo in chiaro $m = \mathsf{Dec}_k(c)$. Sebbene l'accesso ad entrambi gli oracoli sia mantenuto sia prima che dopo aver scelto i due messaggi $m_0, m_1 \in \mathcal{M}$

(sulla base delle risposte ricevute fino a quel momento), ovviamente l'avversario *non può* interrogare l'oracolo di decifratura in corrispondenza del crittotesto sfida c_b (altrimenti otterrebbe m_b e potrebbe facilmente distinguere la cifratura di m_0 da quella di m_1).

Definizione 5.3 (Sicurezza CCA per cifrari simmetrici). Diremo che il cifrario Π_{SKE} è CCA-(t, Q, ϵ)-sicuro se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t che effettua Q richieste agli oracoli di cifratura e decifratura, risulta:

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

La nozione di sicurezza CCA è stata introdotta per la prima volta da Rackoff e Simon [RS91] ed è motivata da attacchi a sistemi reali, come avremo modo di vedere anche nel prossimo capitolo. A volte faremo riferimento ad una nozione più debole di sicurezza (detta CCA1), in cui l'attaccante ha accesso agli oracoli di cifratura e decifratura *solo prima di aver visto il crittotesto sfida*. Si veda anche [Sho98] per alcuni esempi relativi all'importanza della sicurezza CCA.

Vedremo una costruzione di cifrario simmetrico CCA-sicuro nel Capitolo 7, quando analizzeremo un paradigma generale per ottenere sicurezza CCA a partire da ogni cifrario simmetrico CPA-sicuro e da un codice autenticatore di messaggio sicuro (cf. Teorema 7.3).

5.2 Costruzioni teoriche

Il primo esempio di cifrario simmetrico computazionalmente sicuro che studieremo, è basato su un qualsiasi PRG (cf. Definizione 3.2). Lo schema Π_{PRG} è mostrato nel Crittosistema 5.1. Sia G un generatore pseudocasuale con fattore d'espansione ℓ, cioè $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$. L'algoritmo di generazione delle chiavi estrae la chiave k uniformemente a caso in $\{0,1\}^n$. Quindi, per cifrare il messaggio $m \in \{0,1\}^{\ell(n)}$, si calcola $c = m \oplus G(k)$. Dato c e la chiave condivisa k, è sufficiente calcolare $c \oplus G(k)$ per recuperare m. Possiamo mostrare quanto segue:

Teorema 5.2 (Π_{PRG} è IND-sicuro). *Se G è un generatore pseudocasuale (t, ϵ)-sicuro, il cifrario Π_{PRG} è IND-(t, ϵ)-sicuro.*

Dimostrazione. La prova è per riduzione: dato un attaccante $\mathscr{A}$ in grado di violare la sicurezza del cifrario, costruiremo un avversario $\mathscr{D}$ (che usa $\mathscr{A}$ al

Crittosistema 5.1. Il cifrario Π_{PRG} attraverso un PRG $G(\cdot)$

Posto $\mathcal{K} = \{0,1\}^n$ ed $\mathcal{M} = \mathcal{C} = \{0,1\}^{\ell(n)}$, sia $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ un PRG.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n come input genera $k \leftarrow \mathsf{Gen}(1^n)$, dove $k \xleftarrow{\$} \{0,1\}^n$.
- `Cifratura.` Dato un messaggio $m \in \{0,1\}^{\ell(n)}$ ed una chiave $k \in \{0,1\}^n$ calcola il testo cifrato
$$c = \mathsf{Enc}_k(m) = m \oplus G(k).$$
- `Decifratura.` Dato un crittotesto $c \in \{0,1\}^{\ell(n)}$ ed una chiave $k \in \{0,1\}^n$ calcola
$$m = \mathsf{Dec}_k(c) = G(k) \oplus c.$$

suo interno) in grado di distinguere l'output del PRG da una stringa casuale, contro l'ipotesi che quest'ultimo sia sicuro. L'avversario $\mathscr{D}$ riceve come input una stringa $\zeta \in \{0,1\}^{\ell(n)}$ e deve distinguere se essa proviene da $G(U_n)$ oppure da $U_{\ell(n)}$. Per fare ciò, $\mathscr{D}$ simula l'esperimento $\mathbf{Exp}^{\mathrm{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$ per $\mathscr{A}$ come segue:

1. Lancia $\mathscr{A}(1^n)$ per ottenere la coppia di messaggi $m_0, m_1 \in \{0,1\}^{\ell(n)}$.
2. Scegli un bit casuale $b \xleftarrow{\$} \{0,1\}$ e calcola $c_b = m_b \oplus \zeta$.
3. Invia c_b ad $\mathscr{A}$ e sia b' il bit deciso da $\mathscr{A}$. Se $b' = b$ restituisci 1, altrimenti restituisci 0.

Possiamo distinguere due casi, a seconda della natura dell'oracolo di $\mathscr{D}$.

L'oracolo di $\mathscr{D}$ è $U_{\ell(n)}$. In questo caso la stringa ζ è uniformemente casuale in $\{0,1\}^{\ell(n)}$. Ciò significa essenzialmente che il crittotesto che vede $\mathscr{A}$ è come se fosse stato cifrato con il cifrario OTP (cf. Crittosistema 2.1). Ovviamente, $\mathscr{A}$ non ha modo di predire b con probabilità migliore di $1/2$, in quanto OTP ha segretezza perfetta. Ne segue

$$\mathbb{P}\left[\mathscr{D}(U_{\ell(n)}) = 1\right] \leq \frac{1}{2}.$$

L'oracolo di $\mathscr{D}$ è $G(U_n)$. In questo caso la simulazione dell'esperimento effettuata da $\mathscr{D}$ è perfetta, cioè i valori che $\mathscr{A}$ vede nella riduzione sono distribuiti esattamente come nell'esperimento $\mathbf{Exp}^{\text{ind}}_{\mathsf{SKE},\Pi_{\mathsf{SKE}}}(\mathscr{A}, n)$. Siccome abbiamo ipotizzato che $\mathscr{A}$ ha vantaggio ϵ non trascurabile nel violare Π_{PRG}, possiamo concludere:

$$\mathbb{P}\left[\mathscr{D}(G(U_n)) = 1\right] \geq \frac{1}{2} + \epsilon.$$

Ma allora

$$\left|\mathbb{P}\left[\mathscr{D}(G(U_n)) = 1\right] - \mathbb{P}\left[\mathscr{D}(U_{\ell(n)}) = 1\right]\right| \geq \epsilon(n),$$

contro l'ipotesi che $G(\cdot)$ sia (t, ϵ)-sicuro. □

Osserviamo che nel cifrario Π_{PRG} il crittotesto è ottenuto producendo un flusso pseudocasuale di bit da addizionare (modulo 2) al testo in chiaro: i cifrari che operano in questa maniera sono anche detti *cifrari a flusso* (cf. Paragrafo 5.5). Siccome abbiamo già visto nel paragrafo 3.4 che i PRG esistono se e solo se le OWF esistono, l'esistenza di queste ultime implica l'esistenza di cifrari simmetrici IND-sicuri.

Sulla cifratura di messaggi multipli. Osserviamo che la Definizione 5.1 non dice nulla sulla sicurezza di un cifrario simmetrico quando la *stessa chiave* k è usata per cifrare *più di due messaggi*. In effetti, il motivo principale per cui siamo passati al contesto della sicurezza computazionale è proprio la possibilità di usare la stessa chiave (possibilmente corta) per cifrare un numero arbitrario di messaggi (a patto che la chiave sia perfettamente uniforme e nota solo ad Alice ed il Bianconiglio). Bisogna quindi estendere la Definizione 5.1 al caso di *cifratura di messaggi multipli*. L'idea è quella di modificare l'esperimento in modo che la Regina scelga due *vettori di messaggi* $\mathbf{m}_0 = (m_0^1, \ldots, m_0^L), \mathbf{m}_1 = (m_1^1, \ldots, m_1^L)$, con $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}^L$ e riceva dal Brucaliffo il vettore di testi cifrati corrispondente ad uno tra $\mathbf{m}_0$ e $\mathbf{m}_1$. In altri termini, il crittotesto sfida è ora:

$$\mathbf{c}_b = \left(\mathsf{Enc}_k(m_b^1), \ldots, \mathsf{Enc}_k(m_b^L)\right).$$

Lo scopo della Regina è sempre quello di indovinare il bit b.

Purtroppo la Definizione 5.1, *non* implica, in generale, sicurezza nel caso di cifratura di messaggi multipli. Per rendersi conto di ciò, è sufficiente mostrare un esempio di cifrario Π_{SKE} IND-sicuro che può essere violato quando usato nel contesto della cifratura di messaggi multipli. Sia $L = 2$ e consideriamo

il cifrario Π_{PRG} del Crittosistema 5.1. Supponiamo che i vettori scelti da $\mathscr{A}$ nell'esperimento siano:

$$\mathbf{m}_0 = \{0^n, 0^n\} \qquad \mathbf{m}_1 = \{0^n, 1^n\}.$$

Quando $\mathscr{A}$ riceve il vettore sfida $\mathbf{c}_b = (c_b^1, c_b^2)$ controlla semplicemente se $c_b^1 = c_b^2$: se questo è il caso decide per $b = 0$, altrimenti per $b = 1$. Il problema è che il cifrario Π_{PRG} è *deterministico*: lo stesso testo in chiaro è trasformato ogni volta nello stesso crittotesto. Pertanto $\mathscr{A}$ indovina b con probabilità 1. La morale è che *nessun cifrario deterministico può essere IND-sicuro quando usato per la cifratura di messaggi multipli.* Un discorso analogo vale per la sicurezza CPA. (In particolare, abbiamo già visto che un cifrario deterministico non può mai essere CPA-sicuro.)

Fortunatamente, nel caso di cifrari randomizzati è possibile mostrare che la Definizione 5.2 è sufficiente per ottenere sicurezza anche nel contesto della cifratura di messaggi multipli. (La dimostrazione di questo fatto è rimandata al Paragrafo 6.1.) Questo ci permette di fare un grande passo in avanti rispetto a quanto visto nel Capitolo 2: possiamo ora cifrare un numero arbitrario di messaggi usando *la stessa chiave* k.

Per dare un esempio concreto di cifrario CPA-sicuro, dobbiamo introdurre il concetto di *funzione pseudocasuale*.

Funzioni pseudocasuali. Una funzione $F(\cdot)$ è detta una *funzione con chiave* se $F : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$; il primo input è la chiave, il secondo è l'input vero e proprio. Fissata una chiave k, otteniamo così una funzione $F_k : \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$, dove $F_k(\cdot) = F(k, \cdot)$. In genere assumeremo funzioni F che preservano la lunghezza, nel senso che chiave, input ed output saranno tutti della stessa lunghezza $\ell(n) = n$ (ma quanto diremo è facilmente estendibile al caso generale in cui la funzione non preserva la lunghezza). Pertanto, fissando $k \in \{0,1\}^n$, la funzione $F_k(\cdot)$ è una mappa tra stringhe binarie lunghe n bit.

Per funzione pseudocasuale (*PseudoRandom Function*, PRF) intendiamo una funzione che "sembri casuale". Similmente a quanto abbiamo fatto per i PRG, possiamo formalizzare quest'intuizione attraverso il concetto di indistinguibilità: diremo che $F(\cdot)$ è pseudocasuale, se nessun attaccante PPT $\mathscr{D}$ è in grado di distinguerla dalla mappa $\$(\cdot)$ che implementa una vera funzione casuale. Più formalmente:

Definizione 5.4 (Funzione pseudocasuale). Sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una funzione con chiave che preserva la lunghezza, calcolabile in modo

Crittosistema 5.2. La trasformazione GGM

Trasforma qualsiasi generatore pseudocasuale $G : \{0,1\}^n \to \{0,1\}^{2n}$ in una funzione pseudocasuale $F_k : \{0,1\}^n \to \{0,1\}^n$ come segue.

- Siano $G_0(\cdot)$ (risp. $G_1(\cdot)$) la metà sinistra (risp. destra) dell'output di $G(\cdot)$.
- Data la chiave $k \xleftarrow{\$} \{0,1\}^n$ ed un input $x \in \{0,1\}^n$ restituisci

$$F_k(x_1, \ldots, x_n) = G_{x_n}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots).$$

efficiente. Diremo che F è una PRF (t, ϵ)-sicura se, per ogni attaccante PPT $\mathscr{D}$ eseguibile in tempo t, risulta:

$$\left| \mathbb{P}\left[\mathscr{D}^{F_k(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1\right] \right| \leq \epsilon(n),$$

dove $k \xleftarrow{\$} \{0,1\}^n$, e $\$(\cdot)$ è scelta a caso nell'insieme delle mappe tra stringhe di n bit.

A volte avremo bisogno di tenere in considerazione il numero di richieste $Q(n)$ che $\mathscr{D}$ effettua all'oracolo. Ovviamente, affinché $\mathscr{D}$ sia PPT, deve essere $t(n), Q(n) = \text{poly}(n)$. ■

Esiste una costruzione generale di PRF a partire da un qualsiasi PRG. La costruzione è detta GGM in onore dei suoi ideatori: Goldreich, Goldwasser e Micali [GGM84]. Sia $G : \{0,1\}^n \to \{0,1\}^{2n}$ un generatore pseudocasuale che raddoppia la lunghezza (cf. Definizione 3.2), ed indichiamo con $G(x) = (G_0(x), G_1(x))$ la metà sinistra e la metà destra dell'output di $G(x)$, con $|x| = |G_0(x)| = |G_1(x)| = n$. La trasformazione GGM è mostrata nel Crittosistema 5.2. In pratica possiamo immaginare di costruire un albero binario come segue (cf. anche Fig. 5.1). La radice dell'albero è la chiave segreta $k \xleftarrow{\$} \{0,1\}^n$ della nostra PRF. Dunque, per ogni nodo k' nell'albero, il parente sinistro di k' ha valore $G_0(k')$ ed il parente destro $G_1(k')$. Il valore $F_k(x)$ è calcolato esaminando l'espressione binaria di $x = (x_1, \ldots, x_n)$ e percorrendo l'albero dalla radice alle foglie in accordo a tale stinga (i.e., $x_i = 0$ significa andare a sinistra nell'albero e viceversa). Notare che la dimensione dell'albero è esponenziale in n (in particolare ci sono 2^n foglie), ma per calcolare l'output non è necessario costruire e memorizzare l'intero albero: bisogna solo calcolare i valori definiti dall'input x.

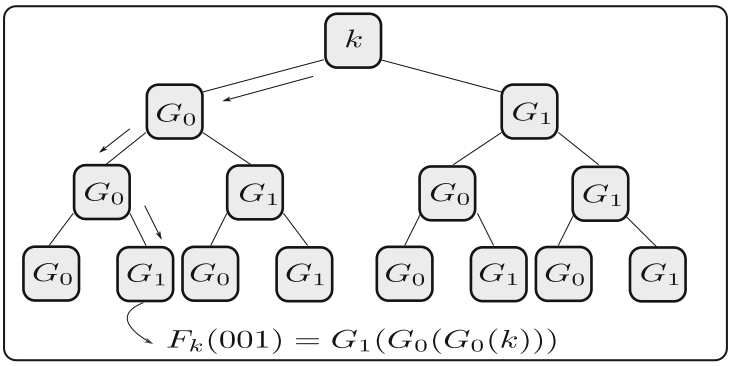


Fig. 5.1. Una rappresentazione grafica della trasformazione GGM

Teorema 5.3 (Trasformazione GGM, ovvero PRG ⇒ PRF). *Se $G(\cdot)$ è un PRG $(t_{\mathsf{PRG}}, \epsilon_{\mathsf{PRG}})$-sicuro, la trasformazione di Fig. 5.1 definisce una PRF $(t_{\mathsf{GGM}}, \epsilon_{\mathsf{GGM}})$-sicura, dove*

$$t_{\mathsf{PRG}} \approx t_{\mathsf{GGM}} \qquad \epsilon_{\mathsf{PRG}} = \epsilon_{\mathsf{GGM}}/n^2.$$

Dimostrazione. La dimostrazione usa l'argomento ibrido (cf. Teorema 3.1). Sia $\mathscr{D}$ un avversario PPT che ha accesso ad un oracolo che implementa $F_k(\cdot)$ (per una chiave k a caso) oppure una funzione veramente casuale $\$(\cdot)$ (scelta a caso nell'insieme delle funzioni che mappano n bit in n bit).

Sia $\mathcal{H}_n^i$ (per $0 \leq i \leq n$) la distribuzione su un albero binario con profondità n definita come segue: i valori fino al nodo i sono stringhe binarie scelte uniformemente a caso in $\{0,1\}^n$, mentre i valori dei nodi $j > i$ sono calcolati usando la trasformazione GGM. (Notare che a tale scopo tutti i valori contenuti nei nodi *fino* al livello $i-1$ sono irrilevanti). Osserviamo che $\mathcal{H}_n^n$ rappresenta una funzione puramente casuale $\$(\cdot)$ che mappa stringhe lunghe n bit in stringhe lunghe n bit, in quanto i valori di tutte le foglie (cioè i nodi a profondità n) sono scelti indipendentemente ed uniformemente a caso. D'altra parte $\mathcal{H}_n^0$ rappresenta esattamente l'albero definito dalla trasformazione GGM, in quanto solo la radice dell'albero è uniformemente casuale.

Applicando l'argomento ibrido sappiamo che, se esiste un attaccante PPT $\mathscr{D}$ in grado di distinguere la costruzione GGM da una funzione veramente casuale con vantaggio ϵ_{GGM}, allora deve esistere un indice i per cui gli ibridi $\mathcal{H}_n^i$ ed $\mathcal{H}_n^{i+1}$ possono essere distinti almeno con vantaggio $\epsilon_{\mathsf{GGM}}/n$. Mostriamo ora come costruire un attaccante PPT $\mathscr{D}'$ in grado di distinguere $G(U_n)$ dalla distribuzione

uniforme U_{2n}, contro l'ipotesi che $G(\cdot)$ sia un generatore pseudocasuale. Siccome $\mathscr{D}'$ non può memorizzare tutto l'albero di profondità n, lo costruirà "al volo" come descritto di seguito.

Sia $Q_{\mathsf{GGM}}(n) = \text{poly}(n)$ (risp. $Q_{\mathsf{PRG}} = \text{poly}(n)$) il numero di richieste che $\mathscr{D}$ (risp. $\mathscr{D}'$) effettua al suo oracolo nel tempo $t_{\mathsf{GGM}}(n)$ (risp. $t_{\mathsf{PRG}}(n)$). L'attaccante $\mathscr{D}'$ riceve $2n \cdot Q_{\mathsf{PRG}}(n)$ bit e deve stabilire se essi sono prodotti applicando $G(\cdot)$ (per $Q_{\mathsf{PRG}}(n)$ volte) oppure se costituiscono una stringa completamente casuale. (Sappiamo dal Corollario 3.4 che se questo è possibile, allora $G(\cdot)$ non può essere un PRG.) L'attaccante $\mathscr{D}'$ simula l'ambiente per $\mathscr{D}$ come segue:

1. Inizializza un albero binario vuoto ed estrai a caso $i \xleftarrow{\$} \{0, \ldots, n-1\}$.
2. Quando $\mathscr{D}$ effettua una richiesta $x \in \{0,1\}^n$ al suo oracolo, usa l'espressione binaria di $x = (x_1, \ldots, x_i, \ldots, x_n)$ per raggiungere un nodo a livello i dell'albero.
3. Usa uno dei $Q(n)$ campioni in input (ciascuno lungo $2n$ bit) ed assegna al parente sinistro del nodo individuato al passo precedente valore pari alla prima metà del campione in input ed al parente destro valore pari all'altra metà (n bit ciascuno). Continua a calcolare l'output usando la costruzione GGM.
4. Se in futuro una richiesta x di $\mathscr{D}$ conduce ad un nodo che già è stato riempito, rispondi in modo coerente ai valori già assegnati (senza ripetere il passo precedente). Altrimenti usa un nuovo campione di input come specificato al passo precedente.

Osserviamo che $\mathscr{D}'$ riempie l'albero binario *dinamicamente* a seconda delle richieste di $\mathscr{D}$, così che *non ha bisogno di memorizzare l'intero albero.* Inoltre se $\mathscr{D}$ è PPT anche $\mathscr{D}'$ lo è (con simile tempo d'esecuzione $t_{\mathsf{PRG}} \approx t_{\mathsf{GGM}}$). Possiamo allora distinguere due casi a seconda della natura dell'oracolo relativo a $\mathscr{D}'$.

L'oracolo di $\mathscr{D}'$ restituisce campioni da U_{2n}. In questo caso $\mathscr{D}'$ riceve dal suo oracolo una stringa completamente casuale lunga $2n \cdot Q_{\mathsf{PRG}}(n)$ bit. Pertanto l'albero simulato da $\mathscr{D}'$ è identico a quello nella distribuzione $\mathcal{H}_n^{i+1}$, in quanto tutti i valori a livello $i+1$ riempiti (dinamicamente) da $\mathscr{D}'$ sono casuali.

L'oracolo di $\mathscr{D}'$ restituisce campioni da $G(U_n)$. In questo caso $\mathscr{D}'$ riceve dal suo oracolo una stringa pseudocasuale lunga $2n \cdot Q_{\mathsf{PRG}}(n)$ bit (cioè $Q_{\mathsf{PRG}}(n)$ invocazioni di $G(\cdot)$ per un valore di seme scelto indipendentemente a caso ogni

volta). Pertanto l'albero simulato da $\mathscr{D}'$ è identico a quello nella distribuzione $\mathcal{H}_n^i$, in quanto tutti i valori a livello $i+1$ riempiti (dinamicamente) da $\mathscr{D}'$ sono pseudocasuali. (Notare che i semi usati per genere i valori pseudocasuali non sono noti a $\mathscr{D}'$, ma questo non fa alcuna differenza.)

Pertanto $\mathscr{D}'$ spera che l'indice i sia quello per cui $\mathscr{D}$ sia in grado di distinguere i due ibridi $\mathcal{H}_n^i$ ed $\mathcal{H}_n^{i+1}$: quando ciò si verifica $\mathscr{D}'$ può quindi distinguere $G(U_n)$ da U_{2n}. Il vantaggio di $\mathscr{D}'$ è pertanto $\epsilon_{\mathsf{PRG}} = \epsilon_{\mathsf{GGM}}/n^2$. Questo conclude la dimostrazione. □

Una funzione pseudocasuale è sufficiente per costruire un cifrario simmetrico CPA-sicuro. Sia $F(\cdot)$ una PRF che preserva la lunghezza. Consideriamo il cifrario simmetrico $\Pi_{\mathsf{PRF}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ del Crittosistema 5.3. Osserviamo che il cifrario è randomizzato, con spazio della randomicità $\Omega = \{0,1\}^n$. L'idea di fondo è quella di cifrare $m \in \mathcal{M}$ estraendo una stringa $\omega \xleftarrow{\$} \Omega$, calcolando $c = F_k(\omega) \oplus m$ e restituendo (ω, c). Il ricevitore legittimo può usare k per calcolare $F_k(\omega)$ e quindi recuperare $m = F_k(\omega) \oplus c$.

Intuitivamente, il cifrario è CPA-sicuro perché $F_k(\omega)$ appare completamente casuale ad un attaccante che osserva una coppia (ω, c), *fintanto che* ω *è un valore fresco non usato in altre cifrature* (o meglio fino a quando l'oracolo di cifratura non usa lo stesso valore ω per rispondere ad una delle richieste

Crittosistema 5.3. Il cifrario Π_{PRF} attraverso una PRF $F_k(\cdot)$

Posto $\mathcal{K} = \mathcal{M} = \Omega = \{0,1\}^n$ e $\mathcal{C} = \{0,1\}^{2n}$, sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una PRF.

- `Generazione delle chiavi.` Dato 1^n come input genera $k \leftarrow \mathsf{Gen}(1^n)$, dove $k \xleftarrow{\$} \{0,1\}^n$.
- `Cifratura.` Dato un messaggio $m \in \{0,1\}^n$ ed una chiave $k \in \{0,1\}^n$, estrai $\omega \xleftarrow{\$} \{0,1\}^n$ e poni

$$\mathsf{Enc}_k(m) = (\omega, F_k(\omega) \oplus m).$$

- `Decifratura.` Dato un crittotesto $(\omega, c) \in \{0,1\}^{2n}$ e la chiave k, calcola

$$m = \mathsf{Dec}_k(\omega, c) = F_k(\omega) \oplus c.$$

dell'attaccante). Comunque, si può dimostrare, che tale evento si verifica solo con probabilità trascurabile ed infatti:

Teorema 5.4 (Π_{PRF} è CPA-sicuro). *Se $F(\cdot)$ è una funzione pseudocasuale $(t_{\mathsf{PRF}}, \epsilon - Q/2^n)$-sicura, Π_{PRF} è CPA-(t, Q, ϵ)-sicuro, dove $t_{\mathsf{PRF}} \approx t$.*

Dimostrazione. Dato un attaccante PPT $\mathscr{A}$ che viola la sicurezza CPA del cifrario con vantaggio ϵ non trascurabile, mostriamo come costruire un attaccante PPT $\mathscr{D}$ in grado di distinguere $F_k(\cdot)$ da una funzione completamente casuale $\$(\cdot)$ (scelta a caso tra le funzioni che mappano n bit in n bit), contro l'ipotesi che $F(\cdot)$ sia una PRF. L'attaccante $\mathscr{D}$ ha accesso ad un oracolo che può essere interrogato con input $\omega \in \{0,1\}^n$, e che restituisce una stringa $\zeta \in \{0,1\}^n$ calcolata come $\zeta = F_k(\omega)$ (per una chiave casuale k) oppure $\zeta = \$(\omega)$. Per distinguere, $\mathscr{D}$ simula l'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_{\mathsf{PRF}}}(\mathscr{A}, n)$ per $\mathscr{A}$ come segue:

1. Lancia $\mathscr{A}(1^n)$. Quando $\mathscr{A}$ richiede la cifratura di un messaggio $m_i \in \{0,1\}^n$, estrai $\omega_i \xleftarrow{\$} \{0,1\}^n$ ed invia ω_i all'oracolo, ricevendo come risposta la stringa $\zeta_i \in \{0,1\}^n$. Restituisci quindi ad $\mathscr{A}$ la coppia $(\omega_i, \zeta_i \oplus m_i)$.

2. Quando $\mathscr{A}$ sceglie i due messaggi m_0, m_1, estrai $b \xleftarrow{\$} \{0,1\}$. Scegli $\omega \xleftarrow{\$} \{0,1\}^n$, invialo all'oracolo ed ottieni la risposta ζ. Inoltra ad $\mathscr{A}$ il crittotesto sfida $(\omega, \zeta \oplus m_b)$.

3. Continua a rispondere alle richieste di $\mathscr{A}$ come al passo (1). Quando $\mathscr{A}$ decide per il bit b', restituisci 1 se $b' = b$, altrimenti ritorna 0.

Notare che $\mathscr{D}$ è eseguibile in tempo $t_{\mathsf{PRF}} \approx t$. Possiamo distinguere due casi a seconda della natura dell'oracolo di $\mathscr{D}$.

L'oracolo di $\mathscr{D}$ è $\$(\cdot)$. In questo caso, ogni volta che $\mathscr{D}$ interroga il suo oracolo con un input ω_i, la stringa ζ_i che riceve come risposta è completamente casuale. Pertanto, i crittotesti che $\mathscr{A}$ vede nella simulazione sono stringhe uniformemente casuali in $\{0,1\}^{2n}$. L'unica cosa di cui dobbiamo preoccuparci è del caso sfortunato in cui il valore ω usato per simulare il crittotesto sfida sia già stato usato in precedenza (o verrà usato successivamente) per rispondere ad una delle richieste di $\mathscr{A}$ all'oracolo di cifratura. Indichiamo con $\mathcal{E}$ tale evento; siccome $\mathscr{A}$ effettua al più Q richieste all'oracolo di cifratura, la probabilità di

$\mathcal{E}$ è $\mathbb{P}[\mathcal{E}] = Q/2^n$. Pertanto:

$$\begin{aligned}\mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1\right] &= \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1 \wedge \mathcal{E}\right] + \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1 \wedge \overline{\mathcal{E}}\right] \\ &\leq \mathbb{P}[\mathcal{E}] + \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1 \mid \overline{\mathcal{E}}\right] \\ &\leq \frac{Q}{2^n} + \frac{1}{2}.\end{aligned}$$

L'oracolo di $\mathscr{D}$ implementa $F_k(\cdot)$. In questo caso, siccome le stringhe ζ_i sono calcolate usando $F_k(\cdot)$, l'avversario $\mathscr{D}$ simula perfettamente l'esperimento $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{SKE},\Pi_{\mathsf{PRF}}}(\mathscr{A}, n)$. Poiché abbiamo supposto che $\mathscr{A}$ è in grado di violare la sicurezza CPA del cifrario, abbiamo:

$$\mathbb{P}\left[\mathscr{D}^{F_k(\cdot)}(1^n) = 1\right] \geq \frac{1}{2} + \epsilon.$$

Mettendo tutto insieme abbiamo trovato

$$\left|\mathbb{P}\left[\mathscr{D}^{F_k(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1\right]\right| \geq \epsilon - \frac{Q}{2^n}.$$

Siccome per ipotesi ϵ non è trascurabile e $Q = Q(n) = \text{poly}(n)$, l'avversario $\mathscr{D}$ può distinguere i due oracoli con vantaggio non trascurabile contro l'ipotesi che la PRF sia sicura. Questo conclude la dimostrazione. □

Permutazioni pseudocasuali. Una permutazione pseudocasuale (*Pseudo-Random Permutation*, PRP o anche *cifrario a blocco*) è una funzione con chiave $P : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$ (calcolabile efficientemente) tale che per ogni chiave k la funzione $P_k(\cdot) = P(k, \cdot)$ è una *corrispondenza biunivoca*.[31] Assumeremo inoltre che $P_k^{-1}(\cdot)$ sia calcolabile efficientemente. (Anche in questo caso, considereremo per semplicità funzioni che preservano la lunghezza, ovvero $\ell(n) = n$.) Informalmente, diremo che una PRP è sicura se nessun attaccante PPT è in grado di distinguere $P_k(\cdot)$ (per una chiave casuale k) da una permutazione puramente casuale $\pi(\cdot)$ (scelta a caso dall'insieme delle permutazioni casuali tra stringhe di $\ell(n)$-bit). Più formalmente:

Definizione 5.5 (Permutazione pseudocasuale). Sia $P : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una permutazione con chiave calcolabile in modo efficiente. Diremo che

[31] Ovvero essa è tale che ad ogni elemento del dominio corrisponde uno ed un solo elemento del codominio (e viceversa).

$P(\cdot)$ è una *permutazione pseudocasuale* (t, ϵ)-sicura se, per ogni algoritmo PPT $\mathscr{D}$ eseguibile in tempo t, risulta:

$$\left| \mathbb{P}\left[\mathscr{D}^{P_k(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\pi(\cdot)}(1^n) = 1\right] \right| \leq \epsilon(n),$$

dove $k \xleftarrow{\$} \{0,1\}^n$ e $\pi(\cdot)$ è scelta a caso nell'insieme delle permutazioni casuali di stringhe lunghe n bit.

Quando l'avversario $\mathscr{D}$ ha anche accesso agli oracoli *inversi*, si parla di PRP *forte* (*Strong PseudoRandom Permutation*, SPRP). Diremo quindi che la funzione $P(\cdot)$ è una PRP forte (t, ϵ) sicura, se, per ogni algoritmo PPT $\mathscr{D}$ eseguibile in tempo t, risulta:

$$\left| \mathbb{P}\left[\mathscr{D}^{P_k(\cdot), P_k^{-1}(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\pi(\cdot), \pi^{-1}(\cdot)}(1^n) = 1\right] \right| \leq \epsilon(n),$$

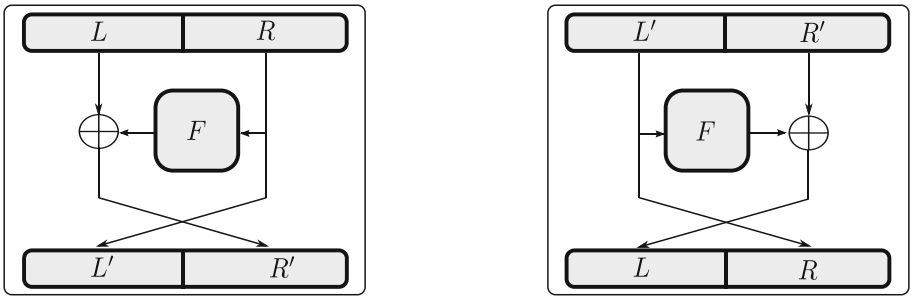
dove $k \xleftarrow{\$} \{0,1\}^n$ e $\pi(\cdot)$ è scelta a caso nell'insieme delle permutazioni casuali di stringhe di n bit. ■

Vogliamo ora mostrare che $P(\cdot)$ è una PRP *se e solo se* $P(\cdot)$ è una PRF. Un verso è banale:

Teorema 5.5 (PRP ⇒ PRF). *Se $P(\cdot)$ è una PRP $(t, \epsilon - Q^2/2^{n+1})$-sicura, allora essa è anche una PRF (t, ϵ)-sicura, dove $Q = \text{poly}(n)$ indica il numero di richieste eseguibili in tempo t.*

Dimostrazione. La dimostrazione usa l'argomento ibrido. Siccome $P(\cdot)$ è una PRP essa è indistinguibile da una permutazione veramente casuale $\pi(\cdot)$; d'altra parte affinché $P(\cdot)$ sia un PRF deve essere indistinguibile da una funzione veramente casuale $\$(\cdot)$. Pertanto, ci basta mostrare che $\pi(\cdot)$ è computazionalmente indistinguibile da $\$(\cdot)$ e, per transitività, ciò equivale a dire che $P(\cdot)$ è una PRF.

Immaginiamo allora un'attaccante $\mathscr{D}$ che interagisce con un oracolo che per ogni richiesta ω restituisce una tra $\pi(\omega)$ oppure $\$(\omega)$. L'attaccante deve stabilire se l'oracolo implementa $\pi(\cdot)$ oppure $\$(\cdot)$. Osserviamo che l'unica possibilità per $\mathscr{D}$ di distinguere i due casi è trovare una collisione (infatti l'esistenza di due elementi distinti che sono mappati nello stesso elemento è l'unica caratteristica che differenzia una funzione puramente casuale da una permutazione puramente casuale). Sia $\mathcal{E}$ l'evento per cui una delle Q richieste inviate da $\mathscr{D}$



(a) Permutazione di Feistel (b) Permutazione di Feistel inversa

Fig. 5.2. Round di base di una rete di Feistel

generi una collisione. Ovviamente:

$$\begin{aligned}\mathbb{P}[\mathcal{E}] &= [\text{sia } \mathcal{E}_i \text{ l'vento per cui l'input } i \text{ collide con l'input } j] \\ &\leq \sum_{1 \leq j \leq i \leq Q} \mathbb{P}[\mathcal{E}_i] \\ &= \binom{Q}{2} \frac{1}{2^n} = \frac{Q(Q-1)}{2^{n+1}} \approx \frac{Q^2}{2^{n+1}}.\end{aligned}$$

Siccome $\mathscr{D}$ è PPT, $Q = \text{poly}(n)$ e quindi il vantaggio di $\mathscr{D}$ è trascurabile. Ne segue che le permutazioni casuali sono indistinguibili dalle funzioni casuali. □

L'altro verso dell'implicazione è dovuto ad una costruzione molto famosa di Luby e Rackoff [LR85] basata sulla permutazione di Feistel, che introduciamo di seguito.

Reti di Feistel. La costruzione seguente è detta *permutazione di Feistel*, in onore del suo creatore. L'input è costituito da una stringa di $2n$ bit, separata in due blocchi di n bit, nel seguito indicati con L ed R. Come mostrato in Fig. 5.2(a) la metà di destra (ovvero la stringa R) viene portata subito in uscita e costituisce la metà sinistra dell'output; d'altra parte il valore R è usato come input per la funzione $F(\cdot)$ e la stringa così ottenuta è sommata modulo 2 con il valore L, per ottenere la metà destra dell'output. Un po' più formalmente, data una funzione $F : \{0,1\}^n \to \{0,1\}^n$ possiamo costruire una permutazione

$\Psi_F : \{0,1\}^{2n} \to \{0,1\}^{2n}$ come segue:

$$\Psi_F(L,R) = (R, F(R) \oplus L)$$
$$\Psi_F^{-1}(L,R) = (F(L) \oplus R, L)\,.$$

Notare che anche Ψ_F^{-1} è calcolabile efficientemente, come mostrato in Fig. 5.2(b). Una *rete di Feistel* è ottenuta iterando la permutazione di Feistel in diversi round; in generale ogni round farà uso di una diversa funzione $F_i(\cdot)$.

Definizione 5.6 (Rete di Feistel). Siano $F_1, \ldots, F_r : \{0,1\}^n \to \{0,1\}^n$ funzioni calcolabili in modo efficiente. La rete di Feistel ad r round è definita come:

$$\Psi_F^r(L,R) = \Psi_{F_r} \circ \ldots \circ \Psi_{F_1}(L,R) = \Psi_{F_r}\left(\Psi_{F_{r-1}}\left(\ldots\left(\Psi_{F_1}(L,R)\right)\ldots\right)\right).$$

Dato l'output $(L',R') = \Psi_F^r(L,R)$, possiamo invertire la trasformazione calcolando:

$$(L,R) = \Psi_{F_1}^{-1} \circ \ldots \circ \Psi_{F_r}^{-1}(L',R') = \Psi_{F_1}^{-1}\left(\ldots\left(\Psi_{F_{r-1}}^{-1}\left(\Psi_{F_r}^{-1}(L',R')\right)\right)\ldots\right).$$

■

Come vedremo le reti di Feistel sono alla base di diverse costruzioni di cifrari a blocco usate in pratica, tra cui il DES (cf. Paragrafo 5.3).

Il risultato di Luby e Rackoff è che, quando $F(\cdot)$ è una PRF (con o senza chiave), $r = 3$ round sono sufficienti per ottenere una PRP, ed $r = 4$ round generano una PRP forte.

Teorema 5.6 (PRF ⇒ PRP). *Siano $F_1, \ldots F_r : \{0,1\}^n \to \{0,1\}^n$ delle PRF (t,ϵ)-sicure. Allora $\Psi_F^3(L,R)$ è una PRP $(t, 3\cdot\epsilon + 3\cdot Q^2/2^{n+1})$-sicura, dove $Q = \mathrm{poly}(n)$ indica il numero di richieste eseguibili in tempo t.*

Dimostrazione. Presentiamo solo una bozza della dimostrazione, lasciamo al lettore i dettagli. Sia (L,R) l'input, ed indichiamo con (L_i,R_i) gli output del round i, per $i = 1,2,3$. Scriviamo esplicitamente gli output intermedi:

$$(L_1,R_1) = (R, F_1(R) \oplus L)$$
$$(L_2,R_2) = (R_1, F_2(R_1) \oplus L_1)$$
$$(L_3,R_3) = (R_2, F_2(R_2) \oplus L_2).$$

Per mostrare che $\mathscr{D}$ non è in grado di distinguere $\Psi_F^3(\cdot)$ da una permutazione casuale $\pi(\cdot)$ su stringhe di $2n$ bit, useremo una *sequenza di giochi* computazionalmente indistinguibili. Nel seguito, indicheremo con G_i la variabile aleatoria corrispondente all'output del gioco i-simo.

Giochi estremi. Cominciamo a definire i due giochi estremi G_0 e G_4. Il gioco G_0 non è altro che l'esperimento originale nel caso in cui l'oracolo di $\mathscr{D}$ implementa la costruzione di Luby e Rackoff. Pertanto:

$$\mathbb{P}[G_0 = 1] = \mathbb{P}\left[\mathscr{D}^{\Psi_F^3(\cdot)}(1^n) = 1\right].$$

Il gioco G_4 d'altra parte è l'esperimento originale nel caso in cui l'oracolo di $\mathscr{D}$ implementa la permutazione casuale $\pi(\cdot)$. Pertanto:

$$\mathbb{P}[G_4 = 1] = \mathbb{P}\left[\mathscr{D}^{\pi(\cdot)}(1^n) = 1\right].$$

Si tratta quindi di mostrare che $|\mathbb{P}[G_0 = 1] - \mathbb{P}[G_4 = 1]|$ è trascurabile come nell'enunciato del teorema. Per fare ciò definiremo alcuni giochi intermedi.

Gioco G_1. Il gioco G_1 è identico al gioco G_0, ma ora le funzioni $F_1(\cdot), F_2(\cdot)$, $F_3(\cdot)$ sono sostituite da funzioni completamente casuali $\$_1(\cdot), \$_2(\cdot), \$_3(\cdot)$. Un semplice argomento ibrido mostra che, se le PRF sono ϵ-sicure, non è possibile distinguere il gioco G_0 dal gioco G_1 con probabilità migliore di $3 \cdot \epsilon$.

Gioco G_2. Il gioco G_2 è identico al gioco G_1, con la differenza che la stringa R_2 è scelta a caso invece di essere calcolata come $R_2 = \$_2(R_1) \oplus L_1$. Notare che in questo modo l'output intermedio (L_2, R_2) è completamente casuale.

L'osservazione chiave è che i due giochi sono identici, a meno che non si trovi una collisione per R_1 in una delle Q richieste effettuate da $\mathscr{D}$. Sia $\mathcal{E}$ tale evento; non è difficile accorgersi che $\mathbb{P}[\mathcal{E}] \leq Q^2/2^{n+1}$. Consideriamo infatti due input $(L, R) \neq (L', R')$. Se $R = R'$ deve essere $L \neq L'$, e quindi:

$$R_1 = L \oplus \$_1(R) = L \oplus \$_1(R') \neq L' \oplus \$_1(R') = R'_1.$$

Pertanto quando $R = R'$ non si verificano mai collisioni in R_1. Supponiamo ora che $R \neq R'$. Se vogliamo avere $R_1 = R'_1$ si deve verificare $L \oplus L' = \$_1(R) \oplus \$_1(R')$. Siccome $R \neq R'$ e $\$_1(\cdot)$ è una funzione puramente casuale, la stringa $\$_1(R) \oplus \$_1(R')$ è completamente casuale, e quindi la probabilità che essa sia uguale ad un dato elemento $L \oplus L'$ è esattamente 2^{-n}. Siccome $\mathscr{D}$ effettua Q richieste, ci sono $\binom{Q}{2}$ coppie possibili e quindi la probabilità che si verifichi almeno una collisione è $\leq Q^2/2^{n+1}$ che è trascurabile in quanto $Q = \text{poly}(n)$.

Siccome $\$_2(\cdot)$ è una funzione puramente casuale ed i valori R_1 sono distinti se $\mathcal{E}$ non si verifica, tutti i valori $\$_2(R_1)$ sono casuali ed indipendenti. Ne segue che R_2 è casuale ed il gioco G_2 è indistinguibile dal gioco G_1.

Gioco G_3. Il gioco G_3 è identico al gioco G_2, con la differenza che la stringa R_3 è scelta a caso invece di essere calcolata come $R_3 = \$_3(R_2) \oplus L_2$. Notare che in questo modo l'output (L_3, R_3) è completamente casuale. Non è complesso vedere che la probabilità di distinguere questi due giochi è ancora una volta limitata dalla probabilità dell'evento $\mathcal{E}$ definito sopra.

Inoltre, come visto nella dimostrazione del Teorema 5.5, la probabilità di distinguere $\pi(\cdot)$ (ovvero la funzione usata nel gioco G_4) da una funzione veramente casuale (che corrisponde all'output generato in G_3), è al più $Q^2/2^{n+1}$.

Mettendo tutto insieme, abbiamo così ottenuto:

$$\mathbb{P}\left[\mathscr{D}^{\Psi_F^3(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\pi(\cdot)}(1^n) = 1\right] \leq 3 \cdot \epsilon + \frac{3 \cdot Q^2}{2^{n+1}}.$$

□

Per una prova che $r = 4$ round sono sufficienti ad ottenere una PRP forte, si rimanda direttamente a [LR85]. Negli anni successivi sono state analizzate diverse varianti della costruzione di Luby e Rackoff, richiedendo che le funzioni interne non siano indipendenti [Pie90], usando primitive più deboli delle PRF [Luc96; NR99; MP03; Mau+06], oppure dando all'avversario la possibilità di interrogare i blocchi interni singolarmente [RR00]. Recentemente, sono state anche definite *reti di Feistel generalizzate*, in cui ad esempio le lunghezze degli input possono essere sbilanciate. Si rimanda a [HR10] per una loro definizione ed analisi formale.

5.3 Cifrari simmetrici nel mondo reale

Come visto nel Paragrafo 2.1, i cifrari basati solamente su sostituzioni o permutazioni non sono sicuri a causa delle caratteristiche del linguaggio. Si potrebbe pensare di rendere tali cifrari sicuri usando più sostituzioni o permutazioni in successione, ma osserviamo che: (i) due sostituzioni sono equivalenti ad una sola sostituzione e (ii) due permutazioni sono equivalenti ad una sola permutazione. L'approccio usato in pratica è quello di alternare permutazioni e sostituzioni (come già suggerito da Shannon in [Sha49]) in una *rete di sostituzione-permutazione*: quest'idea è alla base della struttura di tutti i cifrari a blocco moderni.

Le sostituzioni, come vedremo implementate attraverso *scatole di sostituzione* (*Substitution boxes*, S-box), servono a creare *confusione*. La proprietà di confusione si riferisce alla caratteristica di rendere la relazione tra testo cifrato e chiave di cifratura la più complessa possibile. Lo scopo principale è far

sì che sia impossibile ricavare la chiave anche quando si è in possesso di un elevato numero di coppie testo in chiaro/testo cifrato (prodotte con la stessa chiave). In particolare, ciò richiede che cambiare un singolo bit della chiave cambi completamente il testo cifrato.

Le permutazioni, come vedremo implementate attraverso *scatole di permutazione* (*Permutation boxes*, P-box), contribuiscono a fare *diffusione*. La proprietà di diffusione si riferisce alla caratteristica del crittotesto di dissipare le proprietà di ridondanza statistica presenti nel testo in chiaro. In altri termini, la proprietà di non-uniformità della distribuzione delle singole lettere nel testo in chiaro (dovuta in prevalenza alle proprietà linguistiche dello stesso), deve essere molto meno evidente nel testo cifrato. Un cifrario con buona diffusione dovrebbe, ad esempio, essere tale che cambiare un solo bit del testo in chiaro genera un crittotesto completamente diverso, in maniera del tutto impredicibile. (Questa proprietà è anche detta *effetto a valanga* e praticamente richiede che cambiare un certo insieme di bit nel messaggio originale faccia sì che ogni bit del messaggio cifrato cambi con probabilità 1/2.)

La maggior parte dei cifrari a blocco odierni è basata sulla rete di Feistel descritta nel Paragrafo 5.1. Solitamente si considera la costruzione di Fig. 5.2 per un certo numero r di round: l'input è il testo in chiaro e l'output finale costituisce il testo cifrato. Le funzioni F_i sono sostituite da un'unica funzione F con chiave, utilizzata con valori di chiave indipendenti in ciascun round. (Come vedremo, solitamente, la sequenza delle sottochiavi da utilizzare nei diversi round è derivata a partire da un'*unica* chiave k.) A volte, dopo l'ultimo round, la parte destra e la parte sinistra sono scambiate; inoltre può essere presente una permutazione iniziale.

Se indichiamo con L_0 ed R_0 i due blocchi iniziali, possiamo rappresentare la permutazione di Feistel nell'i-simo round come in

$$\Psi_{F_{k_i}}(L_{i-1}, R_{i-1}) = (L_i, R_i) = (R_{i-1}, L_{i-1} \oplus F_{k_i}(R_{i-1}))\,,$$

per ogni $i = 0, \ldots, r$. Come sappiamo, la proprietà fondamentale della permutazione di Feistel è che per decifrare il blocco i-simo non serve invertire F, in quanto

$$\Psi^{-1}_{F_{k_i}}(L_i, R_i) = (R_i \oplus F_{k_i}(R_i), L_i) = (L_{i-1}, R_{i-1}),$$

come è immediato verificare (cf. anche Fig. 5.2). In generale, la sicurezza aumenta all'aumentare delle dimensioni del blocco base e della chiave k, del numero di round e della complessità associata ad F; d'altra parte queste caratteristiche rallentano il cifrario.

Lo Standard DES. Il *Data Encryption Standard (DES)* è uno dei cifrari a blocco più usati al mondo. La prima versione è stata progettata negli anni 1973-1974 dall'IBM, sulla base del cifrario di Feistel. Il cifrario è stato reso pubblico nel 1975 ed approvato come standard federale Americano nel 1976. Sembra che la National Security Agency (NSA) abbia giocato un ruolo particolare nello sviluppo del DES.[32] In particolare c'è il sospetto che l'NSA abbia indebolito la sicurezza del cifrario in modo da poter decifrare il contenuto dei messaggi cifrati. Uno dei progettisti, Alan Konehim, ha dichiarato: "Abbiamo inviato le S-box a Washington e ci sono tornate indietro completamente modificate!". L'SSCI (United States Senate Select Committee on Intelligence) nel 1978 ha deliberato:

> Nello sviluppo del DES, l'NSA ha convinto l'IBM che una dimensione ridotta della chiave fosse sufficiente, ha assistito indirettamente allo sviluppo delle S-box e ha certificato che l'algoritmo finale del DES è privo di ogni debolezza statistica o matematica.
>
> ...
>
> L'NSA non ha manomesso il progetto dell'algoritmo in nessun modo. L'IBM ha inventato e progettato l'algoritmo, preso tutte le decisioni che lo riguardavano e concordato che la dimensione scelta per la chiave era più che sufficiente per le applicazioni commerciali per cui il DES era stato pensato.

Alcuni dei sospetti sono stati confermati nel 1990, quando Eli Biham ed Adi Shamir hanno scoperto la *crittanalisi differenziale*, un metodo generale per attaccare i cifrari a blocco.[33] Si è scoperto che le S-box del DES erano molto resistenti a questo tipo di attacco, suggerendo che tali tecniche erano già note all'IBM negli anni 70. La conferma di ciò si è avuta nel 1994, quando uno dei progettisti, Don Coppersmith, ha pubblicato alcuni dei disegni originali del progetto. Stando a quanto Steven Levy ha dichiarato, l'IBM ha scoperto la crittanalisi differenziale e l'NSA ha chiesto di tenere la tecnica segreta per questioni di sicurezza nazionale.

[32] Si veda ad esempio `http://en.wikipedia.org/wiki/Data_Encryption_Standard` per approfondire la storia del DES.

[33] L'idea di base consiste nel confrontare certe differenze tra gli input con le corrispondenti differenze negli output. Sia n la lunghezza dei blocchi, e siano $\Delta_x, \Delta_y \in \{0,1\}^n$ due stringhe. Si dice che il differenziale (Δ_x, Δ_y) appare con probabilità p se per $x_1, x_2 \xleftarrow{\$} \{0,1\}^n$ tali che $x_1 \oplus x_2 = \Delta_x$, e per una chiave k scelta a caso, la probabilità che $F_k(x_1) \oplus F_k(x_2) = \Delta_y$ è p. Se F è una funzione veramente casuale, nessun differenziale deve apparire con probabilità maggiore di 2^{-n}. È possibile definire un attacco a messaggio scelto che sfrutta i differenziali per recuperare la chiave segreta.

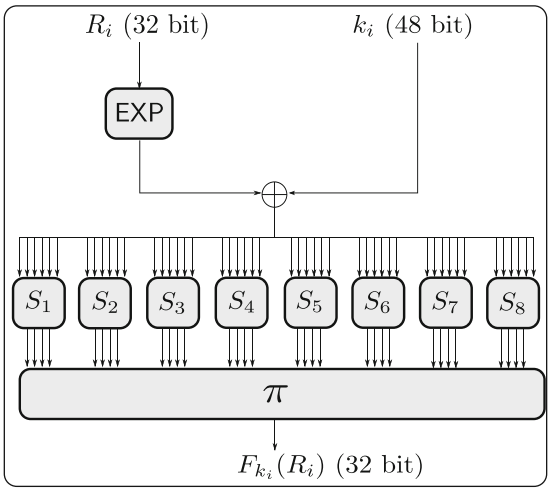


Fig. 5.3. La funzione F della rete di Feistel implementata dal DES

Descriviamo ora la struttura di base usata nel DES. Il cifrario implementa esattamente una rete di Feistel (cf. Fig. 5.2) con $r = 16$ round che processano blocchi di 64 bit; la sequenza delle sotto-chiavi (lunghe 48 bit) è derivata da un'unica chiave k a 56 bit. Per completare la descrizione bisogna definire la funzione $F_{k_i}(\cdot)$, per una data sotto-chiave, e l'algoritmo per generare le sotto-chiavi (detto nel seguito *gestione della chiave*).

La funzione $F_{k_i}(\cdot)$ prende in ingresso la metà destra dell'input al round i-simo, ovvero la stringa R_i. L'output $F_{k_i}(R_i)$ è esso stesso una stringa di 32 bit. Lo schema di riferimento è mostrato in Fig. 5.3. La stringa d'ingresso è prima di tutto espansa in una stringa di 48 bit attraverso una funzione di espansione EXP; il risultato $\mathsf{EXP}(R_i)$ è quindi sommato modulo 2 con la sottochiave k_i (anch'essa lunga 48 bit). I 48 bit risultanti sono divisi in 8 gruppi da 6 bit e ciascun gruppo costituisce l'input di una S-box. La S-box mappa i 6 bit che riceve come input in 6 bit di output, così che in totale si ottengono 32 bit. A tali bit è infine applicata una permutazione $\pi(\cdot)$.

Ciascuna scatola di sostituzione può essere vista come una tabella con 4 righe e 16 colonne, riempite con valori nell'intervallo $0, \ldots, 15$. Dato un input a 6 bit i due bit estremi (cioè il bit meno significativo ed il bit più significativo) individuano la riga ($2^2 = 4$ possibili valori) ed i restanti 4 bit intermedi individuano la colonna ($2^4 = 16$ possibili valori). Il valore nella tabelle è infine convertito

Tab. 5.1. La prima delle 8 S-box usate nel DES

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

in binario (4 bit). Un esempio è mostrato nella Tab. 5.1: dato l'input 110001 i bit estremi (11) individuano la riga numero 3; i restanti bit (1000) individuano la colonna numero 8. Quindi l'output è 5, ovvero la stringa 0101.

Passiamo alla gestione della chiave. La dimensione reale della chiave nel DES è 64 bit, ma 8 bit sono solo di ridondanza. Quindi, a partire dalla chiave viene scartato un bit ogni 8, ottenendo così la chiave a 56 bit di cui abbiamo parlato finora. Dopo aver subito una prima permutazione $\pi_1(\cdot)$, la stringa risultante è separata in due metà da 28 bit ciascuna. In ciascuno dei 16 round, la chiave k_i è derivata come mostra la Fig. 5.4. I 56 bit in ingresso subiscono uno spostamento circolare a sinistra; la stringa risultante sarà l'input per il prossimo round. Nel round corrente, d'altra parte, si selezionano 24 bit in ogni metà che formano la chiave k_i dopo aver applicato un'altra permutazione $\pi_2(\cdot)$.

La debolezza principale del DES è che la dimensione della chiave è troppo piccola, così che oggi un attacco a forza bruta non è più così impensabile. Inoltre i blocchi sono troppo corti. A parte queste caratteristiche la costruzione è molto solida, tanto che dopo circa tre decadi, l'unico attacco (pratico) al DES è ancora l'attacco a forza bruta. Nel 1977, Diffie ed Hellman hanno congetturato una macchina dal costo di 20 000 000 $ in grado di violare il DES in un solo giorno. Nel 1993, Wiener ha proposto una macchina dal costo di 1 000 000 $ in grado di violare lo schema in 7 ore. Nel 1997, il progetto DESCHALL (DES Challenge) ha violato il DES in 96 giorni, usando i cicli "vuoti" di migliaia di computer connessi ad internet. Nel 1998, l'EFF (Electronic Frontier Foundation) ha costruito un sistema dal valore di 250 000 $ in grado di eseguire l'attacco a forza bruta in 2 giorni. COPACOBANA (Cost-Optimized Parallel COde Breaker), dal costo di 10 000 $, ha violato il DES in una settimana. Nel 1980, Biham e Shamir hanno mostrato come sia possibile violare il DES attraverso la crittanalisi differenziale usando un attacco a messaggio scelto che richiede 2^{47} richieste all'oracolo di cifratura (quindi l'attacco non è considerato pratico).

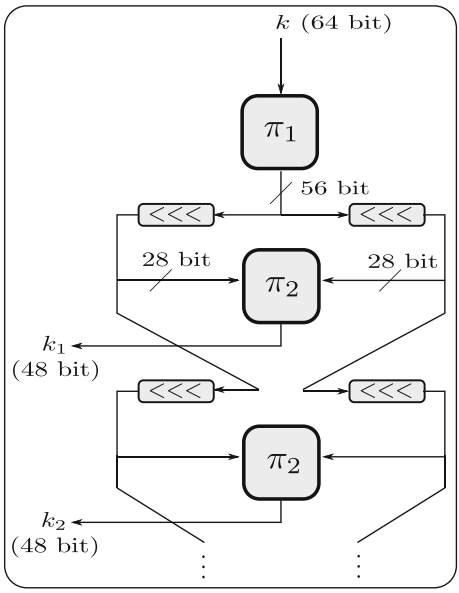


Fig. 5.4. La gestione della chiave nel DES

Un modo possibile per aumentare la lunghezza della chiave è usare il DES *in cascata*. L'idea più naturale è quella della doppia cifratura, in cui si pone:

$$c = \mathsf{Enc}_{k_1,k_2}(m) = \mathsf{Enc}_{k_2}(\mathsf{Enc}_{k_1}(m)). \tag{5.1}$$

Purtroppo questo schema non è sicuro a causa dell'attacco "incontro-nel-mezzo" (*meet-in-the-middle*). Mostriamo come sia possibile violare lo schema attraverso un attacco che richiede solamente 3 coppie testo in chiaro/testo cifrato (m, c). L'attacco procede come segue. Data la coppia (m, c) vogliamo trovare k_1 e k_2 che soddisfano l'Eq. (5.1). Allora:

1. Per ogni possibile chiave k_1 calcoliamo $z = \mathsf{Enc}_{k_1}(m)$ e memorizziamo le coppie (z, k_1) così ottenute in una lista $\mathcal{L}_1$.

2. Per ogni possibile chiave k_2 calcoliamo $z = \mathsf{Dec}_{k_2}(c)$ e memorizziamo le coppie (z, k_2) così ottenute in una lista $\mathcal{L}_2$.

3. Ordiniamo le liste $\mathcal{L}_1$ ed $\mathcal{L}_2$ rispetto alla prima componente.

4. L'output è costituito dall'insieme $\mathcal{S}$ di tutte le coppie (k_1, k_2) tali che, per un qualche z, risulta:

$$(z, k_1) \in \mathcal{L}_1 \quad \text{e} \quad (z, k_2) \in \mathcal{L}_2.$$

Supponiamo che la chiave abbia la stessa lunghezza del blocco, ovvero n bit. Siccome la mappa $\mathsf{Enc}_{k_2}(\mathsf{Enc}_{k_1}(\cdot))$ ha 2^n valori possibili, la probabilità che una coppia (k_1, k_2) a caso sia tale che $c = \mathsf{Enc}_{k_2}(\mathsf{Enc}_{k_1}(m))$, è $\approx 2^{-n}$. Poiché ci sono 2^{2n} possibili coppie, abbiamo

$$\mathbb{E}\left[\#\mathcal{S}\right] \approx 2^{2n} \cdot 2^{-n} = 2^n,$$

così che abbiamo 2^n candidati per la coppia corretta (k_1, k_2). Per eliminare i falsi positivi, ripetiamo il procedimento con una nuova coppia (m', c'). Non è difficile accorgersi che la probabilità che (k_1, k_2) sia un falso positivo per entrambe le coppie (m, c) ed (m', c') è $\approx 2^{-n} \cdot 2^{-n} = 2^{-2n}$; ne segue che il valore atteso del numero di falsi positivi è $\approx 2^{2n} \cdot 2^{-2n} = 1$. Una terza coppia (m'', c'') elimina anche questo falso positivo.

Le cose cambiano se decidiamo di cifrare 3 volte, come nel *Triplo DES* (3-DES). Invece di usare 3 chiavi, si continuano ad usare 2 chiavi per mantenere la compatibilità all'indietro con gli utenti che utilizzano il classico DES. Il testo cifrato è calcolato come in

$$c = \mathsf{Enc}_{k_1}(\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m))).$$

Quando $k_1 = k_2$ in effetti il sistema è del tutto equivalente al DES.

Lo Standard AES. La comparsa dei primi attacchi teorici al DES e la scarsa efficienza del 3-DES nel cifrare blocchi di piccole dimensioni ha indotto il NIST, nei primi anni 90, ad indire una competizione per la progettazione di un nuovo cifrario. Il cifrario vincitore della competizione sarebbe stato chiamato *Advanced Encryption Standard (AES)*. Correva l'anno 1997: 15 candidati furono accettati nel Giugno del 1998 e tra questi 5 arrivarono alla fase finale nell'Agosto 1999. La proposta vincente fu quella di Vincent Rijmen e Joan Daemen dal Belgio (rinominata cifrario di *Rijndael* unendo[34] il nome dei suoi inventori), selezionata nell'Ottobre 2000. L'AES è diventato quindi uno standard nel Novembre 2001.

[34]Rijndael, in Fiammingo, si pronuncia approssimativamente "rèin-daal".

I criteri di scelta iniziale per il nuovo cifrario furono la sicurezza (resistenza all'attacco a forza bruta ed agli attacchi basati sulle tecniche di crittanalisi differenziale e lineare[35]), il costo, gli aspetti implementativi e la semplicità. Gli algoritmi che raggiunsero la fase finale erano tutti molto buoni, alcuni basati su pochi round complessi, altri su un numero maggiore di round più semplici. La soluzione vincente era semplice, veloce e compatta; in particolare essa può essere implementata in modo molto efficiente sulle architetture a 32 bit.

La lunghezza della chiave non è univoca; infatti il cifrario può lavorare con chiavi di 3 diverse dimensioni: 128, 192 o 256 bit. Il blocco base che viene processato ha dimensione 128 bit (16 byte). Il numero di round può essere 10, 12 o 14 a seconda della dimensione della chiave. In ogni round si utilizza una sotto-chiave derivata da quella principale (come già per il DES); tutte le sotto-chiavi hanno dimensione 128 bit. Il blocco principale di 16 byte è processato in 4 gruppi da 4 byte (ovvero 4 parole da 32 bit) che costituiscono lo *stato* del cifrario.

In ciascun round lo stato subisce 4 operazioni invertibili:

1. $\mathsf{SubBytes}(\cdot)$: sostituzione di byte che compongono lo stato attraverso una S-box.

2. $\mathsf{ShiftRows}(\cdot)$: permutazione dei byte tra i diversi gruppi che compongono lo stato.

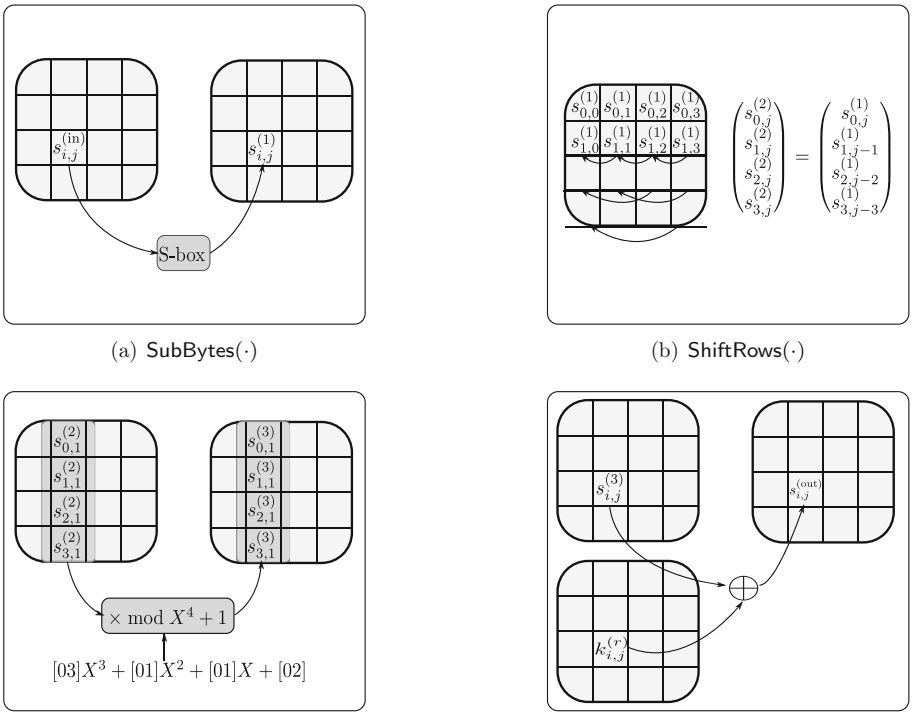
3. $\mathsf{MixColumns}(\cdot)$: sostituzione dei byte che compongono lo stato attraverso un'opportuna moltiplicazione matriciale.

4. $\mathsf{AddRoundKey}(\cdot)$: somma modulo 2 con la sotto-chiave.

Infine, come nel DES, è prevista una manipolazione della chiave originale per generare le sotto-chiavi nel processo di gestione della chiave. Rappresentiamo lo stato del cifrario con la matrice $\mathbf{S}$ a 16 elementi, dove ciascun elemento costituisce 1 byte dello stato. All'inizio lo stato coincide con un blocco del testo in chiaro da cifrare; al termine dell'ultimo round lo stato rappresenta il corrispondente blocco di testo cifrato.

Descriviamo l'algoritmo nel caso in cui la chiave sia lunga 128 bit e quindi quando ci sono $r = 10$ round. Le sotto-chiavi sono derivate come segue. Le 4 parole da 32 bit iniziali sono espanse in 44 parole da 32 bit. Le prime 4

[35] La crittanalisi lineare è stata sviluppata da Matsui [Mat94]. L'idea di base è considerare alcune relazioni di tipo lineare tra input ed output: tanto più il cifrario è lontano da implementare una PRP, quanto più facile è recuperare la chiave segreta attraverso queste relazioni.



(a) SubBytes(·) (b) ShiftRows(·)

(c) MixColumns(·) (d) AddRoundKey(·)

Fig. 5.5. Le 4 operazioni fondamentali in AES

parole così ottenute (128 bit) vengono usate in un primo AddRoundKey(·) (cf. anche Fig. 5.5(d)) con il testo in chiaro (sempre 128 bit); le restanti 40 parole sono usate 4 alla volta in ciascuno dei 10 round (quindi ecco che la sotto-chiave usata in ogni round è lunga $32 \cdot 4 = 128$ bit). La funzione di espansione copia la chiave originale nelle prime 4 parole da 32 bit: $w_0, \ldots, w_3$; in generale w_j è funzione di w_{j-1} e w_{j-4} attraverso alcuni spostamenti a sinistra e somme modulo 2 (dettagli omessi).

Restano da descrivere le 4 operazioni fondamentali. Lo faremo con particolare attenzione ad un'implementazione efficiente su architetture a 32 bit. AES è basato sull'aritmetica nell'anello di polinomi $\mathbb{Z}_2[X]/(h(X))$, essendo

$h(X)$ il polinomio irriducibile $h(X) = X^8 + X^4 + X^3 + X + 1$.[36] L'operazione $\mathsf{SubBytes}(\cdot)$ definisce una semplice sostituzione dello stato rappresentata da una tabella 16×16 contenente una permutazione di tutte le possibili coppie di valori esadecimali $00, \ldots, \mathrm{FF}$. Ciascun byte dello stato è trasformato in due cifre esadecimali di cui la prima cifra indica la riga e la seconda cifra indica la colonna corrispondente al nuovo byte da inserire nello stato (cf. Fig. 5.5(a)).[37] Siano $s_{i,j}^{(\mathrm{in})}$ gli elementi costituenti lo stato $\mathbf{S}$ all'inizio di ogni round; l'operazione descritta trasforma gli elementi dello stato come in $s_{i,j}^{(1)} = \mathsf{SubBytes}(s_{i,j}^{(\mathrm{in})})$.

L'operazione $\mathsf{ShiftRows}(\cdot)$ effettua un semplice spostamento a sinistra delle righe dello stato: per ogni $j = 0, 1, 2, 3$ la j-sima riga dello stato subisce uno spostamento a sinistra di j posizioni (cf. Fig. 5.5(b)). Pertanto:

$$\begin{pmatrix} s_{0,j}^{(2)} \\ s_{1,j}^{(2)} \\ s_{2,j}^{(2)} \\ s_{3,j}^{(2)} \end{pmatrix} = \begin{pmatrix} s_{0,j}^{(1)} \\ s_{1,j-1}^{(1)} \\ s_{2,j-2}^{(1)} \\ s_{3,j-3}^{(1)} \end{pmatrix} \qquad \forall\ j = 0, \ldots, 3,$$

dove i pedici vanno intesi modulo 4 ove necessario.

L'operazione $\mathsf{MixColumns}(\cdot)$ (cf. Fig. 5.5(c)) sostituisce un byte dello stato in funzione di tutti e 4 i byte della sua stessa colonna. Possiamo vedere questa trasformazione come una mappa lineare: ogni colonna dello stato viene associata ad un polinomio di grado 3 con coefficienti pari al valore dei 4 byte nella colonna; tale polinomio è quindi moltiplicato per il polinomio $[03]X^3 + [01]X^2 + [01]X + [02]$ modulo $X^4 + 1$. Un semplice calcolo mostra che

[36] Un polinomio è detto irriducibile se non ha divisori a parte 1 e se stesso. Un elemento in $\mathbb{Z}_2[X]/(X^8 + X^4 + X^3 + X + 1)$ è un polinomio di grado al più 7 con coefficienti in $\mathbb{Z}_2$. La somma di due polinomi è definita addizionando (modulo 2) i coefficienti dei termini con uguale grado. La moltiplicazione è definita calcolando il polinomio prodotto e prendendo il resto della divisione per $h(X)$. Per calcolare l'inverso di un polinomio in $\mathbb{Z}_2[X]/(h(X))$, si può utilizzare l'algoritmo di Euclide esteso (cf. Lemma B.7). In particolare si può mostrare che ogni elemento è invertibile e che tutte le proprietà della Definizione B.4 sono soddisfatte, così che $\mathbb{Z}_2[X]/(h(X))$ è in realtà un campo.

[37] La permutazione è derivata attraverso la seguente trasformazione algebrica. All'inizio la S-box è riempita in modo sequenziale con tutti i valori esadecimali $00, \ldots, \mathrm{FF}$. Ad ogni valore esadecimale (8 bit) può essere associato un polinomio di grado 7 con coefficienti in $\mathbb{Z}_2$. Quindi, per ognuno di questi polinomi, si calcola l'inverso modulo il polinomio irriducibile $h(X)$. Infine, il polinomio così ottenuto subisce una trasformazione lineare.

ciò equivale alla trasformazione:

$$\begin{pmatrix} s_{0,j}^{(3)} \\ s_{1,j}^{(3)} \\ s_{2,j}^{(3)} \\ s_{3,j}^{(3)} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} s_{0,j}^{(2)} \\ s_{1,j}^{(2)} \\ s_{2,j}^{(2)} \\ s_{3,j}^{(2)} \end{pmatrix} \qquad \forall\ j = 0, \ldots, 3.$$

Infine l'operazione $\mathsf{AddRoundKey}(\cdot)$ calcola la somma modulo 2 tra i 128 bit di stato e la sotto-chiave utilizzata nel round $1 \leq r \leq 10$. Se indichiamo con $(k_{i,j}^{(r)})$ i byte che compongono tale chiave, possiamo calcolare lo stato in uscita dal round r-simo come:

$$\begin{aligned}
\begin{pmatrix} s_{0,j}^{(\text{out})} \\ s_{1,j}^{(\text{out})} \\ s_{2,j}^{(\text{out})} \\ s_{3,j}^{(\text{out})} \end{pmatrix} &= \begin{pmatrix} s_{0,j}^{(3)} \\ s_{1,j}^{(3)} \\ s_{2,j}^{(3)} \\ s_{3,j}^{(3)} \end{pmatrix} \oplus \begin{pmatrix} k_{0,j}^{(r)} \\ k_{1,j}^{(r)} \\ k_{2,j}^{(r)} \\ k_{3,j}^{(r)} \end{pmatrix} \\
&= \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} \mathsf{SubBytes}\left(s_{0,j}^{(\text{in})}\right) \\ \mathsf{SubBytes}\left(s_{1,j-1}^{(\text{in})}\right) \\ \mathsf{SubBytes}\left(s_{2,j-2}^{(\text{in})}\right) \\ \mathsf{SubBytes}\left(s_{3,j-3}^{(\text{in})}\right) \end{pmatrix} \oplus \begin{pmatrix} k_{0,j}^{(r)} \\ k_{1,j}^{(r)} \\ k_{2,j}^{(r)} \\ k_{3,j}^{(r)} \end{pmatrix} \\
&= \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \cdot \mathsf{SubBytes}\left(s_{0,j}^{(\text{in})}\right) \oplus \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \cdot \mathsf{SubBytes}\left(s_{1,j-1}^{(\text{in})}\right) \\
&\oplus \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \cdot \mathsf{SubBytes}\left(s_{2,j-2}^{(\text{in})}\right) \oplus \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \cdot \mathsf{SubBytes}\left(s_{3,j-3}^{(\text{in})}\right) \oplus \begin{pmatrix} k_{0,j}^{(r)} \\ k_{1,j}^{(r)} \\ k_{2,j}^{(r)} \\ k_{3,j}^{(r)} \end{pmatrix} \\
&= T_0\left[s_{0,j}^{(\text{in})}\right] \oplus T_1\left[s_{1,j-1}^{(\text{in})}\right] \oplus T_2\left[s_{2,j-2}^{(\text{in})}\right] \oplus T_3\left[s_{3,j-3}^{(\text{in})}\right] \oplus \begin{pmatrix} k_{0,j}^{(r)} \\ k_{1,j}^{(r)} \\ k_{2,j}^{(r)} \\ k_{3,j}^{(r)} \end{pmatrix},
\end{aligned}$$

avendo definito le seguenti 4 tabelle da 256 byte

$$T_0[x] = \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} \cdot \mathsf{SubBytes}(x) \qquad T_1[x] = \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} \cdot \mathsf{SubBytes}(x)$$

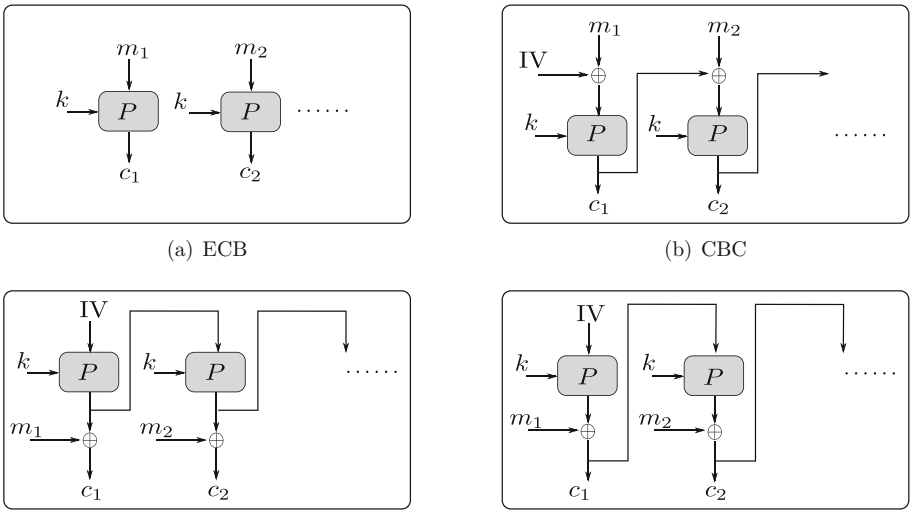
$$T_2[x] = \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} \cdot \mathsf{SubBytes}(x) \qquad T_3[x] = \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} \cdot \mathsf{SubBytes}(x).$$

Siccome queste 4 tabelle possono essere pre-calcolate, l'aggiornamento dello stato in un singolo round richiede solamente 4 operazioni di ricerca su tabella e 4 somme modulo 2 per ciascuna colonna, più 4 kB di memoria per memorizzare le tabelle. Si ritiene che un'implementazione così efficiente sia stato uno dei fattori determinanti per la scelta dei cifrario di Rijndael.

5.4 Modi operativi

Un cifrario a blocco protegge blocchi di testo a lunghezza fissa (e.g., 64 bit per il DES e 128 bit per AES). Siccome in generale un messaggio da cifrare può avere lunghezza arbitraria, è necessario specificare un *modo operativo*: si divide il testo in chiaro in blocchi e si individua una modalità per proteggere i diversi blocchi così ottenuti attraverso il cifrario a blocco sottostante. In particolare è necessario definire un'operazione preliminare che renda la lunghezza del messaggio un multiplo intero della dimensione di un singolo blocco. Tale operazione prende il nome di riempimento (*padding* in inglese). Ogni forma di riempimento è valida, purché essa sia invertibile in modo non ambiguo. Ad esempio, è possibile aggiungere un bit 1 seguito dal numero di 0 necessari a rendere la lunghezza del messaggio un multiplo intero della dimensione del blocco. Nel seguito supporremo che il messaggio m sia costituito da B blocchi, ovvero $m = m_1 \,||\, \ldots \,||\, m_B$; assumeremo inoltre che la dimensione di ciascun blocco sia compatibile con la dimensione dell'input del cifrario a blocco sottostante, che indicheremo con la funzione $P(\cdot)$ (modellata come una PRP). Si veda [Bel+97] per una trattazione più formale dei modi operativi.

Modalità ECB. Il modo operativo più naturale è detto ECB (dall'inglese *Electronic Code Book*). La cifratura del messaggio m è ottenuta cifrando ciascun blocco *indipendentemente* (cf. Fig. 5.6(a)), ottenendo così $c =$



(a) ECB

(b) CBC

(c) OFB

(d) CFB

Fig. 5.6. Modi operativi

$F_k(m_1) \,||\, \ldots \,||\, F_k(m_B)$. La decifratura è eseguita in modo ovvio, sfruttando il fatto che $F_k^{-1}(\cdot)$ è calcolabile in modo efficiente.

Questo modo operativo è deterministico e quindi esso non può mai essere CPA-sicuro (cf. discussione dopo la Definizione 5.2). In effetti tale modo operativo non è nemmeno semanticamente sicuro, in quanto due blocchi identici di testo in chiaro sono mappati in due blocchi identici nel testo cifrato: è banale dunque distinguere la cifratura di un messaggio m_0 costituito da due blocchi identici, dalla cifratura di un messaggio m_1 costituito da due blocchi distinti.

Modalità CBC. Nel modo operativo CBC (dall'inglese *Cipher Block Chaining*), si sceglie un vettore iniziale a caso IV della dimensione di un singolo blocco. Quindi ciascun blocco del testo cifrato è ottenuto applicando la funzione $F_k(\cdot)$ alla somma modulo due tra il blocco corrente del testo in chiaro ed il blocco precedente del testo cifrato (cf. Fig. 5.6(b)). In simboli

$$c_i = F_k(c_{i-1} \oplus m_i) \qquad \forall\, i = 1, \ldots, B,$$

dove $c_0 = \text{IV}$. Come vedremo questo modo operativo è CPA-sicuro (cf. Teorema 5.7). Lo svantaggio principale è che la cifratura deve essere eseguita in modo sequenziale, ovvero è necessario attendere che c_{i-1} sia generato prima di cifrare m_i. Osserviamo inoltre che la propagazione di un errore in un blocco di testo cifrato è limitata a due blocchi contigui in decifratura.

Modalità OFB. Nel modo operativo OFB (dall'Inglese *Output Feedback*) si utilizza il cifrario a blocco $F_k(\cdot)$ per produrre un flusso pseudocasuale da sommare modulo due al messaggio originale. (Per questo motivo tale modo operativo è detto anche *a flusso.*) Si veda la Fig. 5.6(c) per una rappresentazione grafica. In simboli:

$$c_i = m_i \oplus \omega_i \qquad \forall\ i = 1, \ldots, B,$$

dove $\omega_i = F_k(\omega_{i-1})$ ed $\omega_1 = F_k(\text{IV})$. Come vedremo anche questo modo operativo è CPA-sicuro. Sia la cifratura che la decifratura devono essere eseguite in modo sequenziale. Siccome il flusso pseudocasuale è indipendente dal messaggio m, questo può essere pre-calcolato; fatto ciò la cifratura è molto efficiente. Gli errori non si propagano (cioè un errore presente in un blocco affligge solo tale blocco).

Modalità CFB. Il modo operativo CFB (dall'inglese *Cipher FeedBack*) è molto simile ad OFB, come mostra la Fig. 5.6(d). La differenza è che il prossimo valore ω_i è calcolato a partire da c_{i-1} e non da m_{i-1}. In simboli:

$$c_i = m_i \oplus \omega_i \qquad \forall\ i = 1, \ldots, B,$$

dove $\omega_i = F_k(c_{i-1})$ ed $\omega_1 = F_k(\text{IV})$. Anche questo modo operativo è CPA-sicuro (cf. Teorema 5.7). Come per il modo operativo CBC, la cifratura non è eseguibile in parallelo ed un errore in un blocco di testo cifrato affligge due blocchi in decifratura.

Teorema 5.7 (Sicurezza CPA dei modi operativi CBC, OFB e CFB). *Supponiamo di utilizzare uno dei modi operativi CBC, OFB oppure CFB per cifrare un messaggio m composto da B blocchi la cui lunghezza è compatibile con la funzione $F_k(\cdot)$ sottostante. Se la funzione $F_k(\cdot)$ è una PRP (t, ϵ)-sicura, tali modi operativi sono CPA-$(t, Q/B, \epsilon + 3 \cdot Q^2/2^{n+1})$-sicuri, dove $Q = \text{poly}(n)$ indica il numero di richieste eseguibili in tempo t.*

Dimostrazione. Consideriamo un avversario $\mathscr{A}$ che esegue un attacco CPA contro uno dei modi operativi CBC, OFB o CFB istanziati con una PRP. Non è complesso mostrare che è impossibile distinguere la configurazione originale dalla stessa configurazione, in cui la funzione $F_k(\cdot)$ è rimpiazzata da una funzione casuale $\$(\cdot)$, con probabilità migliore di $\epsilon + Q^2/2^{n+1}$. (Infatti, per definizione, una PRP è indistinguibile da una permutazione casuale $\pi(\cdot)$ ed abbiamo visto nella dimostrazione del Teorema 5.5 che quest'ultima è indistinguibile da $\$(\cdot)$.)

Invece di fissare la funzione $\$(\cdot)$ all'inizio dell'esperimento, la costruiremo "al volo" come segue. Memorizziamo una tabella con le coppie input-output che abbiamo già utilizzato. Ogni volta che bisogna valutare $\$(\cdot)$ in corrispondenza di un input x, si controlla se il valore x è già presente nella tabella: se questo è il caso si ritorna il valore y già scelto in precedenza, altrimenti si sceglie y a caso e si aggiunge la coppia (x, y) alla tabella. È facile vedere (dalla descrizione dei modi operativi CBC, OFB e CFB) che, fino a quando non dobbiamo consultare la tabella,[38] il vantaggio di $\mathscr{A}$ in un attacco a messaggio scelto è nullo, in quanto l'avversario non vede altro che stringhe completamente casuali.

Pertanto, è sufficiente mostrare che la probabilità che si verifichi una collisione *negli input* di $\$(\cdot)$ è trascurabile. Sia $\mathcal{E}_j$ l'evento che una collisione avvenga nelle prime j interrogazioni della funzione casuale $\$(\cdot)$. Siccome $\mathscr{A}$ effettua Q/B richieste al suo oracolo, $\$(\cdot)$ è invocata Q volte (infatti ogni messaggio m è composto da B blocchi). Dobbiamo mostrare che $\mathbb{P}[\mathcal{E}_Q]$ è trascurabile in n. Mostreremo per induzione su j che

$$\mathbb{P}[\mathcal{E}_j] \leq \frac{(j-1)Q}{2^n},$$

assumendo che $\$(\cdot)$ sia una funzione casuale su stringhe di n bit. Ovviamente $\mathbb{P}[\mathcal{E}_1] = 0$. Supponiamo che l'asserto sia verificato per tutti i valori fino a $j-1$. La probabilità che ci sia una collisione entro le prime j richieste (ovvero $\mathbb{P}[\mathcal{E}_j]$) è la probabilità che la collisione si verifichi in una delle $j-1$ chiamate precedenti (ovvero $\mathbb{P}[\mathcal{E}_{j-1}]$) oppure che il j-simo input di $\$(\cdot)$ collida con uno dei precedenti. Pertanto

$$\mathbb{P}[\mathcal{E}_j] \leq \mathbb{P}[\mathcal{E}_{j-1}] + \mathbb{P}\left[\mathcal{E}_j \mid \overline{\mathcal{E}_{j-1}}\right].$$

Usando l'ipotesi induttiva, il primo termine è limitato da $(j-2)Q/2^n$. Limitiamo il secondo termine. Osserviamo innanzitutto che condizionare sull'evento

[38] Ovvero fintanto che non si verifica una collisione negli input della funzione $F_k(\cdot)$. Ad esempio nel modo operativo OFB una collisione negli input di $F_k(\cdot)$ provoca una ripetizione del flusso pseudocasuale generato (a partire dal momento in cui avviene la collisione in poi). Allo stesso modo in CBC se $c_{i-1} \oplus m_i = c_{j-1} \oplus m_j$ (per qualche $i \neq j$) si ha $c_i = c_j$.

$\overline{\mathcal{E}_{j-1}}$ significa assumere che nelle prime $j-1$ chiamate non si verificano collisioni negli input di $\$(\cdot)$. Siccome la funzione $\$(\cdot)$ è casuale, ciò implica che l'output $j-1$ è esso stesso casuale. Dalla descrizione dei modi operativi CBC, OFB e CFB, questo porta a concludere che l'input j-simo per $\$(\cdot)$ è esso stesso casuale e quindi la probabilità che esso collida con uno dei $j-1$ input precedenti è $(j-1)/2^n \leq Q/2^n$. Mettendo tutto insieme abbiamo mostrato

$$\mathbb{P}[\mathcal{E}_j] \leq \frac{(j-2)Q}{2^n} + \frac{Q}{2^n} = \frac{(j-1)Q}{2^n},$$

come desiderato.

Quindi, $\mathbb{P}[\mathcal{E}_Q] \leq Q^2/2^n$ che è trascurabile quando Q è polinomiale. Sommando tutti i termini il vantaggio di $\mathscr{A}$ in un attacco CPA è limitato da

$$\begin{aligned}\mathbb{P}[\mathscr{A} \text{ vince}] &= \mathbb{P}[\mathscr{A} \text{ vince} \wedge \mathcal{E}_Q] + \mathbb{P}\left[\mathscr{A} \text{ vince} \wedge \overline{\mathcal{E}_Q}\right] \\ &\leq \mathbb{P}\left[\mathscr{A} \text{ vince} \mid \overline{\mathcal{E}_Q}\right] + \mathbb{P}[\mathcal{E}_Q] \\ &\leq \epsilon + \frac{Q^2}{2^{n+1}} + \frac{Q^2}{2^n} = \epsilon + \frac{3 \cdot Q^2}{2^{n+1}}.\end{aligned}$$

Questo conclude la dimostrazione. □

Modalità contatore. Nel modo operativo CTR (dall'inglese *counter*, contatore) si utilizza un contatore per generare un flusso pseudocasuale, similmente a quanto avviene nel modo operativo OFB. Inizialmente, si estrae a caso un vettore iniziale IV e si pone $ctr = \text{IV}$. Successivamente, si calcola il flusso pseudocasuale $\omega_i = F_k(ctr + 1)$, dove la somma va intesa modulo 2^n se il dominio della funzione $F_k(\cdot)$ è $\{0,1\}^n$. Quindi, ciascun blocco di testo in chiaro è cifrato come in $c_i = m_i \oplus \omega_i$. Anche questo modo operativo è CPA-sicuro. Sia la cifratura che la decifratura possono essere eseguite in parallelo ed il flusso pseudocasuale può essere pre-calcolato, dato che è indipendente dal messaggio da cifrare. Osserviamo inoltre che è possibile decifrare un singolo blocco di testo cifrato *indipendentemente* da tutti gli altri.

Sulla scelta dell' IV. Abbiamo visto che alcuni modi operativi utilizzano una stringa iniziale IV. È naturale chiedersi che caratteristiche deve avere tale stringa. Una prima possibilità potrebbe essere quella di usare un valore fisso per IV. Questa strategia ha lo svantaggio che, se due messaggi hanno il primo blocco in comune, la cifratura di tale blocco è identica. Si potrebbe scegliere un valore a caso per IV ed inviarlo insieme al testo cifrato; questo ha lo svantaggio

di creare una ridondanza non trascurabile, sopratutto per messaggi costituiti da pochi blocchi. Inoltre, è richiesta una sorgente di randomicità.

Una buona compromesso è quello di istanziare la stringa IV attraverso un Nonce (dall'inglese *Number used once*). Il Nonce è un identificativo univoco associato ad ogni messaggio (unico per una data chiave di cifratura e per un dato sistema sicuro). Non è necessario tenere il valore del Nonce segreto, ma esso può essere usato una sola volta. Un Nonce fresco è associato a ciascun messaggio, dopo essere stato tradotto in un unico blocco da cifrare in modalità ECB, ottenendo così la stringa IV. Non è necessario trasmettere IV, ma solo il Nonce, il che in generale richiede una minore ridondanza. Per una trattazione formale dei modi operativi su base Nonce si rimanda a [Rog04].

5.5 Cifrari a flusso

Un cifrario a flusso è uno schema di cifratura in cui si genera prima un flusso pseudocasuale di bit, da sommare al testo in chiaro per produrre il testo cifrato. Osserviamo che un cifrario a flusso non richiede riempimento, in quanto può cifrare direttamente a livello di bit (o byte). Un esempio di cifrario a flusso è il cifrario Π_{PRG} del Teorema 5.2. Abbiamo anche visto che alcuni modi operativi (ad esempio OFB oppure la modalità contatore) trasformano un cifrario a blocco in uno a flusso.

Siccome un PRG espande un piccolo seme di vera randomicità in un flusso pseudocasuale, come vedremo possiamo costruire un cifrario a flusso a partire da un qualsiasi PRG. Ovviamente, seppure il PRG è crittograficamente robusto, un uso improprio può rendere il cifrario a flusso risultante completamente insicuro. Ad esempio abbiamo visto che il cifrario Π_{PRG} del Teorema 5.2 è completamente insicuro se si usa la stessa chiave per cifrare più di un messaggio.

Esistono tipicamente due modalità per usare un cifrario a flusso: una *sincrona* ed una *asincrona*. Nella modalità sincrona, Alice ed il Bianconiglio condividono un flusso pseudocasuale che utilizzano per cifrare ciascun messaggio. In altri termini possiamo immaginare che il flusso pseudocasuale sia sezionato in diversi blocchi: Alice cifra il suo messaggio usando il primo blocco, il Bianconiglio riceve il crittotesto, lo decifra e risponde usando il secondo blocco del flusso pseudocasuale condiviso, e così via. La sicurezza dello schema deriva dal fatto che ogni messaggio è cifrato con una parte diversa del flusso pseudocasuale, quindi possiamo interpretare la cifratura di tutti i messaggi inviati da Alice come la cifratura di un unico (lungo) messaggio. Osserviamo che le parti

comunicanti devono mantenere uno stato (cioè devono sapere quale parte del flusso usare per cifrare il messaggio successivo).

Nella modalità asincrona non c'è bisogno di mantenere alcuno stato e la cifratura è effettuata indipendentemente da ciascuna parte comunicante. Come vedremo, il prezzo da pagare è che sarà necessaria una nozione più robusta di PRG. La chiave k è usata come seme di un PRG, insieme ad una stringa iniziale IV di lunghezza n bit. Quindi, la cifratura del messaggio m è:

$$c = \mathsf{Enc}_k(m) = (\mathrm{IV}, G(k \mid\mid \mathrm{IV}) \oplus m),$$

dove $\mathrm{IV} \xleftarrow{\$} \{0,1\}^n$. Data la coppia (IV, c) il ricevitore può recuperare $m = c \oplus G(k||\mathrm{IV})$. Affinché tale costruzione sia sicura, è necessario che $G(k||\mathrm{IV})$ sia pseudocasuale anche quando IV è noto (ma k è segreto). Inoltre per due stringhe uniformemente casuali $\mathrm{IV}_1, \mathrm{IV}_2 \in \{0,1\}^n$ i flussi $G(k||\mathrm{IV}_1)$ e $G(k||\mathrm{IV}_2)$ devono essere pseudocasuali anche quando sono entrambi noti (con il rispettivo IV). Tali requisiti non sono in generale soddisfatti da un PRG crittografico. In effetti un tale PRG è quasi una PRF, anche detta PRF debole (*Weak PseudoRandom Function*), cf. Esercizio 5.11.

Il cifrario RC4. Esistono diverse costruzioni pratiche di cifrari a flusso, tutte estremamente veloci. Sfortunatamente la loro sicurezza non è ben compresa come nel caso dei cifrari a blocco; il motivo principale è che non c'è un cifrario a flusso standard che è stato utilizzato (e quindi attaccato) per anni (come invece è successo per il DES o per AES). I registri a scorrimento a retroazione lineare (*Linear Feedback Shift Register*, LFSR) sono stati molto popolari come cifrari a flusso, ma sono stati dimostrati essere intrinsecamente insicuri (si può recuperare l'intera chiave dati abbastanza bit dell'output). Per queste ragioni, ove possibile, l'uso di un cifrario a blocco è preferibile.

L'esempio più noto di cifrario a flusso è senz'altro RC4, progettato da Rivest. Il cifrario utilizza una chiave $\mathbf{k}$ di lunghezza n variabile fino a 256 byte (quindi ogni elemento $\mathbf{k}[i]$ è rappresentabile con 8 bit), usata per generare un flusso pseudocasuale. A tale scopo si inizializza un vettore di stato $\boldsymbol{\sigma}$ come segue:

- Per $i = 0, \ldots, 255$

$$\boldsymbol{\sigma}[i] = i \qquad\qquad \boldsymbol{\tau}[i] = \mathbf{k}[i \bmod n].$$

- Si pone $j = 0$. Quindi per $i = 0, \ldots, 255$

$$j \leftarrow j + \boldsymbol{\sigma}[i] + \boldsymbol{\tau}[i] \bmod 256 \qquad\qquad \mathsf{Swap}(\boldsymbol{\sigma}[i], \boldsymbol{\sigma}[j]).$$

La funzione $\mathsf{Swap}(\boldsymbol{\sigma}[i], \boldsymbol{\sigma}[j])$ inverte semplicemente gli elementi in posizione i e j nel vettore $\boldsymbol{\sigma}$.

La cifratura del messaggio $\mathbf{m} = (\mathbf{m}[1], \ldots, \mathbf{m}[n])$ si effettua byte per byte addizionando il flusso pseudocasuale $\boldsymbol{\sigma}$ generato e mischiando continuamente i valori di stato.

- Poni $i, j = 0$. Per ogni byte del messaggio da cifrare:

$$\begin{aligned} i &\leftarrow i + 1 \bmod 256 & j &\leftarrow j + \boldsymbol{\sigma}[i] \bmod 256 \\ &\mathsf{Swap}(\boldsymbol{\sigma}[i], \boldsymbol{\sigma}[j]) & i^* &= \boldsymbol{\sigma}[i] + \boldsymbol{\sigma}[j] \bmod 256. \end{aligned}$$

- Quindi cifra il byte corrente $\mathbf{m}[i]$ come $\mathbf{c}[i] = \mathbf{m}[i] \oplus \boldsymbol{\sigma}[i^*]$.

È stato mostrato che i primi byte del flusso generato da RC4 sono polarizzati (cioè non sono propriamente pseudocasuali). In realtà il sistema presenta anche altri problemi, tanto che la chiave k può essere completamente ricostruita a partire da una sequenza di byte cifrati non troppo lunga.[39] Se si scarta un prefisso iniziale di una certa lunghezza non esiste alcun attacco noto. Si raccomanda, pertanto, di usare RC4 scartando i primi 4096 bit (la scelta è conservativa).

[39] Il cifrario RC4 è stato utilizzato nello standard WEP per la sicurezza delle reti senza fili, ad oggi considerato insicuro.

Esercizi

Esercizio 5.1. Assumendo l'esistenza delle funzioni pseudocasuali, mostrare un crittosistema che è IND-sicuro, ma non è mai CPA-sicuro.

Esercizio 5.2. Si dispone di due cifrari Π_1 e Π_2. Sapendo che uno solo dei due schemi è CPA-sicuro, mostrare come combinarli in un cifrario Π che è CPA-sicuro quando almeno uno tra Π_1 e Π_2 è CPA-sicuro.

Esercizio 5.3. Sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ definita come $F_k(x) = k \oplus x$. Mostrare che F non è pseudocasuale. È possibile definire un attacco che utilizza una sola interrogazione?

Esercizio 5.4. Sia $F : \{0,1\}^n \times \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$ una funzione pseudocasuale (t, ϵ)-sicura. Mostrare che

$$\frac{t}{\epsilon} \leq 2^n \cdot O(\ell).$$

Esercizio 5.5. Sia F una permutazione pseudocasuale. Considerare il cifrario $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ che, per una chiave casuale $k \leftarrow \mathsf{Gen}(1^n)$, cifra m estraendo ω a caso e calcolando $c = (F_k(\omega), \omega \oplus m)$. Dire se Π è CPA-sicuro.

Esercizio 5.6. Sia $P : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$ una PRP (t, ϵ)-sicura. Consideriamo il seguente cifrario con chiave condivisa $k \in \{0,1\}^n$. Dato come input un messaggio $m \in \{0,1\}^{\ell/2}$, si sceglie un riempimento casuale $\omega \xleftarrow{\$} \{0,1\}^{\ell/2}$ e si restituisce $c = P_k(\omega || m)$.

1. Spiegare come si decifra c.
2. Dimostrare che il cifrario è CPA-sicuro.
3. Mostrare un attacco CCA contro il cifrario.

Esercizio 5.7. Considerare la permutazione di Feistel di Fig. 5.2 istanziata con una funzione pseudocasuale F.

1. Mostrare che $\Psi_F^1(L, R)$ e $\Psi_F^2(L, R)$ non sono PRP.
2. Mostrare che $\Psi_F^3(L, R)$ non è una PRP forte.

Esercizio 5.8. Mostrare che il modo operativo CBC è insicuro contro attacchi CCA.

Esercizio 5.9. Supponiamo di voler usare il DES come funzione pseudocasuale. Quanti bit sono necessari per memorizzare una funzione veramente casuale che mappa stringhe di 64 bit in stringhe di 64 bit? Supponiamo di voler memorizzare una funzione casuale che mappa stringhe di n bit in stringhe di n bit su un disco da 10GB. Qual è il massimo valore di n che ci possiamo permettere?

Esercizio 5.10. Consideriamo il cifrario AES con chiave a 128 bit. Vogliamo provare un attacco a forza bruta. Supponiamo di avere a disposizione un computer a 500 Mhz in grado di testare una chiave AES per ogni ciclo. Quante chiavi al secondo siamo in grado di testare? Quanto tempo ci vuole a provare tutte le chiavi?

Esercizio 5.11. Dare una definizione formale di PRF debole. Mostrare che una PRF è anche una PRF debole. Sia F' una funzione pseudocasuale. Mostrare che la funzione

$$F(x) = \begin{cases} F'_k(x) & \text{se } x \text{ è pari} \\ F'_k(x+1) & \text{se } x \text{ è dispari,} \end{cases}$$

è una PRF debole ma non una PRF.

Letture consigliate

[Bel+97] Mihir Bellare, Anand Desai, E. Jokipii e Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: *FOCS*. 1997, pp. 394–403.

[GGM84] Oded Goldreich, Shafi Goldwasser e Silvio Micali. "How to Construct Random Functions (Extended Abstract)". In: *FOCS*. 1984, pp. 464–479.

[GM84] Shafi Goldwasser e Silvio Micali. "Probabilistic Encryption". In: *J. Comput. Syst. Sci.* 28.2 (1984), pp. 270–299.

[HR10] Viet Tung Hoang e Phillip Rogaway. "On Generalized Feistel Networks". In: *CRYPTO*. 2010, pp. 613–630.

[LR85] Michael Luby e Charles Rackoff. "How to Construct Pseudo-Random Permutations from Pseudo-Random Functions (Abstract)". In: *CRYPTO*. 1985, p. 447.

[Luc96] Stefan Lucks. "Faster Luby-Rackoff Ciphers". In: *FSE*. 1996, pp. 189–203.

[Mat94] Mitsuru Matsui. "The First Experimental Cryptanalysis of the Data Encryption Standard". In: *CRYPTO*. 1994, pp. 1–11.

[Mau+06] Ueli M. Maurer, Yvonne Anne Oswald, Krzysztof Pietrzak e Johan Sjödin. "Luby-Rackoff Ciphers from Weak Round Functions?" In: *EUROCRYPT*. 2006, pp. 391–408.

[MP03] Ueli M. Maurer e Krzysztof Pietrzak. "The Security of Many-Round Luby-Rackoff Pseudo-Random Permutations". In: *EUROCRYPT*. 2003, pp. 544–561.

[MRS88] Silvio Micali, Charles Rackoff e Bob Sloan. "The Notion of Security for Probabilistic Cryptosystems". In: *SIAM J. Comput.* 17.2 (1988), pp. 412–426.

[NR99] Moni Naor e Omer Reingold. "On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited". In: *J. Cryptology* 12.1 (1999), pp. 29–66.

[Pie90] Josef Pieprzyk. "How to Construct Pseudorandom Permutations from Single Pseudorandom Functions". In: *EUROCRYPT*. 1990, pp. 140–150.

[Rog04] Phillip Rogaway. "Nonce-Based Symmetric Encryption". In: *FSE*. 2004, pp. 348–359.

[RR00] Zulfikar Ramzan e Leonid Reyzin. "On the Round Security of Symmetric-Key Cryptographic Primitives". In: *CRYPTO*. 2000, pp. 376–393.

[RS91] Charles Rackoff e Daniel R. Simon. "Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack". In: *CRYPTO*. 1991, pp. 433–444.

[Sha49] Claude Elwood Shannon. “Communication Theory of Secrecy Systems”. In: *Bell Sys. Tech. J.* 28 (1949), pp. 657–715.

[Sho98] Victor Shoup. *Why chosen ciphertext security matters*. Rapp. tecn. RZ 3076. IBM Zurich, 1998. URL: `http://www.shoup.net/papers/expo.pdf`.

[Yao82] Andrew Chi-Chih Yao. “Theory and Applications of Trapdoor Functions (Extended Abstract)”. In: *FOCS*. 1982, pp. 80–91.

Capitolo 6

Cifrari asimmetrici

Nel capitolo precedente abbiamo studiato le tecniche simmetriche per soddisfare il requisito di confidenzialità in una comunicazione sicura. Come visto, l'ipotesi di base è che Alice ed il Bianconiglio *condividano una chiave segreta* k. Resta il problema di come distribuire le chiavi in modo sicuro. Osserviamo inoltre che volendo utilizzare le tecniche simmetriche in uno scenario con molti utenti, è necessario che ciascun utente condivida una chiave segreta con ogni altro utente: quest'approccio non è quindi molto scalabile. (Cf. anche il Paragrafo 11.1 per una discussione più approfondita su questo punto, nonché per vedere alcuni approcci atti a mitigare il problema.) In questo capitolo vogliamo affrontare il problema della confidenzialità da un punto di vista diverso:

> *Possono Alice ed il Bianconiglio scambiare messaggi su un canale insicuro, celando il contenuto dei messaggi stessi alla Regina Rossa che controlla il canale, senza condividere a priori un segreto?*

La risposta a questa domanda è dovuta a Diffie ed Hellman [DH76], gli inventori della *crittografia a chiave pubblica*. L'idea di base è quella di richiedere che ogni utente del sistema possegga una *coppia* di chiavi: una chiave segreta sk ed una chiave pubblica pk nota a tutti gli utenti del sistema. In questo modo se Alice vuole inviare un messaggio al Bianconiglio preservando la confidenzialità, può cifrarlo *usando la chiave pubblica del Bianconiglio*. Quest'ultimo potrà quindi recuperare il messaggio originale invertendo il processo di cifratura attraverso la sua personale chiave chiave segreta, corrispondente alla chiave pubblica usata da Alice per cifrare il contenuto del messaggio.

Come vedremo sarà necessario mantenere un registro contenente le chiavi pubbliche di tutti gli utenti (in modo che se, ad esempio, Alice vuole comunicare con il Cappellaio Matto, può recuperare la chiave pubblica di quest'ultimo). In effetti, non è difficile accorgersi che abbiamo solamente spostato il problema: dobbiamo ora preoccuparci che le chiavi pubbliche siano *autentiche*, altrimenti la Regina potrebbe semplicemente inserire la sua chiave pubblica nel

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_6, © Springer-Verlag Italia 2012

registro "spacciandola" come la chiave pubblica — diciamo — del Bianconiglio, con il risultato che sarà poi in grado di decifrare tutti i messaggi destinati a quest'ultimo.

Guida per il lettore. Ciò premesso, la struttura del capitolo è la seguente. Nel Paragrafo 6.1 daremo la definizione formale di cifrario a chiave pubblica e studieremo diverse nozioni di sicurezza per cifrari asimmetrici. Nello stesso paragrafo vedremo anche una costruzione generale di cifrario asimmetrico CPA-sicuro. Quindi, nei paragrafi successivi, studieremo alcuni cifrari la cui sicurezza si basa sulla difficoltà a risolvere problemi computazionali legati alla teoria dei numeri: la fattorizzazione di interi (nel Paragrafo 6.2), il logaritmo discreto (nel Paragrafo 6.3) e la residuosità quadratica (nel Paragrafo 6.4). (La comprensione di queste costruzioni richiede diversi concetti di teoria dei numeri, pertanto si rimanda il lettore che non ha familiarità con tali concetti all'Appendice B.) Infine, nel Paragrafo 6.5, introdurremo il cifrario di Cramer-Shoup, noto per essere il primo cifrario asimmetrico CCA-sicuro nel modello standard.

6.1 Nozioni di sicurezza per cifrari asimmetrici

La definizione formale di cifrario asimmetrico ricalca esattamente quella data nel contesto simmetrico (cf. Definizione 2.1). La differenza fondamentale è che lo spazio delle chiavi è costituito ora da coppie di chiavi pubbliche/chiavi segrete; inoltre, il processo di cifratura dipende solo dalla chiave pubblica del destinatario e la decifratura dipende solo dalla chiave segreta del destinatario.

Definizione 6.1 (Cifrario asimmetrico). Indichiamo con $\mathcal{M}$ lo spazio dei possibili testi in chiaro, con $\mathcal{C}$ lo spazio dei testi cifrati e con $\mathcal{K}$ lo spazio delle chiavi. Un cifrario asimmetrico è una tripletta di algoritmi $\Pi_{\mathsf{PKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ definita come segue:

- `Generazioni delle chiavi.` L'algoritmo Gen è l'algoritmo di generazione delle chiavi pubbliche/private. Dato il parametro di sicurezza n restituisce la coppia di chiavi $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ in $\mathcal{K}$.

- `Cifratura.` L'algoritmo $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ è l'algoritmo di cifratura. Dato un messaggio da cifrare $m \in \mathcal{M}$ e la chiave pubblica pk del destinatario calcola il crittotesto $c = \mathsf{Enc}_{pk}(m)$.

- `Decifratura.` L'algoritmo $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ è l'algoritmo di decifratura. Dato un crittotesto c e la chiave segreta sk corrispondente a pk restituisce il messaggio $m \in \mathcal{M}$ tale che $m = \mathsf{Dec}_{sk}(c)$. ■

Richiederemo che lo schema sia *completo*, nel senso che per ogni n, per ogni coppia di chiavi $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ e per ogni messaggio $m \in \mathcal{M}$, si abbia:

$$m = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)).$$

Al solito, quando l'algoritmo di cifratura è randomizzato scriveremo $c \leftarrow \mathsf{Enc}_{pk}(m)$.

Osserviamo che, poiché dato un crittotesto c esiste un unico messaggio m tale che $c = \mathsf{Enc}_{pk}(m)$ e poiché la chiave pk è pubblica, *un cifrario asimmetrico non può mai fornire sicurezza incondizionata*. Come avremo ampiamente modo di vedere, infatti, i cifrari asimmetrici sono tutti basati su problemi computazionali *ritenuti difficili*: mostrare se tali problemi sono veramente non risolvibili in tempo polinomiale è tanto difficile quanto provare che $\mathbf{P} \neq \mathbf{NP}$. Le nozioni di sicurezza per i cifrari asimmetrici sono identiche a quelle nel contesto simmetrico (cf. Paragrafo 5.1) e sono richiamate di seguito.

Sicurezza CPA. La prima definizione da considerare sarebbe una traduzione della Definizione 5.1 nel contesto asimmetrico. Dato un cifrario asimmetrico $\Pi_{\mathsf{PKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, dovremmo richiedere che la Regina non sia in grado di distinguere la cifratura di due messaggi $m_0, m_1 \in \mathcal{M}$ a sua scelta, *data la chiave pubblica* pk. Osserviamo, però, che quando diciamo che la Regina conosce la chiave pubblica pk, stiamo praticamente dandole accesso ad un oracolo di cifratura: data pk, la Regina può cifrare da sola qualsiasi messaggio (rispetto alla chiave pk). Per questo motivo, nel contesto asimmetrico, la nozione di sicurezza IND è del tutto equivalente a quella di sicurezza CPA e possiamo concentrarci solo sulla seconda. Consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n)$:
1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot)}(1^n, pk)$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_{pk}(m_b)$;
4. $b' \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot)}(c_b)$;
5. restituisci 1 se e solo se $b' = b$ e $|m_0| = |m_1|$.

Definizione 6.2 (Sicurezza CPA per cifrari asimmetrici). Diremo che il cifrario Π_{PKE} è CPA-(t, Q, ϵ)-sicuro se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t che effettua Q richieste all'oracolo di cifratura, risulta:

$$\mathsf{IP}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

Dal punto di vista astratto, l'oggetto matematico che permette di realizzare un cifrario asimmetrico CPA-sicuro è quello di funzione unidirezionale con botola (*One-Way Trapdoor Function*, TDF).

Definizione 6.3 (Funzione unidirezionale con botola). Una funzione $f : \mathcal{X} \to \mathcal{Y}$ è una funzione unidirezionale con botola (t, ϵ)-sicura se: (i) $f(\cdot)$ è una funzione unidirezionale (cf. Definizione 3.4), (ii) la funzione inversa $f^{-1}(\cdot)$ può essere calcolata efficientemente attraverso un parametro aggiuntivo tk (detto la "botola"). ■

Quando la funzione $f(\cdot)$ è una permutazione si parla di permutazione con botola (Trapdoor Permutation, TDP). Data una TDP, possiamo usare il fatto che ogni funzione unidirezionale possiede un bit estremo (cf. Teorema 3.6) per definire il seguente cifrario $\Pi_{\mathsf{TDF}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ che cifra un singolo bit (ovvero $\mathcal{M} = \{0, 1\}$). La chiave pubblica pk consiste nella descrizione della funzione $f(\cdot)$ e dell'insieme $\mathcal{X}$, la chiave segreta è $sk = tk$. Per cifrare $m \in \{0, 1\}$ è sufficiente estrarre un elemento $x \xleftarrow{\$} \mathcal{X}$, calcolare $f(x) = y$ e calcolare

$$c \leftarrow \mathsf{Enc}_{pk}(m) = (y, \chi(x) \oplus m),$$

dove $\chi(x)$ è un bit estremo per $f(\cdot)$. Quindi, dato $c = (y, z)$ è possibile recuperare m invertendo y e recuperando il bit estremo usato in cifratura:

$$m = \mathsf{Dec}_{sk}(c) = z \oplus \chi(f^{-1}(y)).$$

È un semplice esercizio dimostrare che questo schema è CPA-sicuro quando la TDP è sicura (cf. Esercizio 6.2).

Sicurezza CCA. Possiamo tradurre anche la Definizione 5.3 nel contesto asimmetrico. In pratica è sufficiente estendere la definizione precedente al caso in cui la Regina abbia anche accesso all'oracolo di decifratura $\mathsf{Dec}_{sk}(\cdot)$. (Ovviamente dobbiamo richiedere che $\mathscr{A}$ non possa interrogare l'oracolo di decifrare in corrispondenza del crittotesto sfida.) Consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n)$:

1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot),\mathsf{Dec}_{sk}(\cdot)}(1^n, pk)$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_{pk}(m_b)$;
4. $b' \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot),\mathsf{Dec}_{sk}(\cdot)}(c_b)$;
5. restituisci 1 se e solo se:
   (i) il crittotesto c_b non è stato mai inviato a $\mathsf{Dec}_{sk}(\cdot)$;
   (ii) $|m_0| = |m_1|$;
   (iii) $b' = b$.

Definizione 6.4 (Sicurezza CCA per cifrari asimmetrici). Diremo che il cifrario Π_{PKE} è CCA-(t, Q, ϵ)-sicuro se per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t che effettua Q richieste agli oracoli di cifratura e decifratura risulta

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

Sulla cifratura di messaggi multipli. Come già discusso nel Capitolo 5, le nozioni di sicurezza CPA e CCA non implicano immediatamente che un crittosistema sia sicuro quando si usa la *stessa* chiave per cifrare un numero arbitrario di messaggi. In effetti, questo è uno dei motivi per cui abbiamo scelto di passare dalla sicurezza incondizionata a quella computazionale: poter usare la stessa chiave condivisa (o la stessa coppia di chiavi pubblica/privata) per cifrare un numero arbitrario di messaggi. Come accennato nel capitolo precedente, fortunatamente, la nozione di sicurezza CPA si estende facilmente al caso di *cifratura di messaggi multipli*. (Un discorso del tutto identico vale per il caso di sicurezza CCA.)

Dobbiamo quindi estendere la Definizione 6.2 a questo caso più generale. Per fare ciò modificheremo l'esperimento in modo che la Regina (data la chiave pubblica pk) scelga due *vettori di messaggi*:

$$\mathbf{m}_0 = \left(m_0^1, \ldots, m_0^L\right)$$
$$\mathbf{m}_1 = \left(m_1^1, \ldots, m_1^L\right),$$

dove $m_0^i, m_1^i \in \mathcal{M}$ per ogni $i = 1, \ldots, L$. Quindi, l'attaccante riceve il crittotesto sfida (ora un vettore):

$$\mathbf{c}_b = \left(\mathsf{Enc}_{pk}(m_b^1), \ldots, \mathsf{Enc}_{pk}(m_b^L)\right),$$

per un bit casuale $b \xleftarrow{\$} \{0,1\}$, e deve stabilire se questo corrisponde alla cifratura di $\mathbf{m}_0$ oppure di $\mathbf{m}_1$ (ovvero deve indovinare il valore di b). Analogamente a quanto già fatto, diremo che un crittosistema Π_{PKE} è CPA-(t, Q, ϵ)-sicuro per cifrare messaggi multipli se nessun attaccante PPT eseguibile in tempo t e che effettua Q richieste all'oracolo di cifratura può indovinare il valore di b con probabilità migliore di $1/2 + \epsilon$. Possiamo allora enunciare il seguente importante teorema:

Teorema 6.1. *Se Π_{PKE} è CPA-$(O(t), O(Q), \epsilon/L)$-sicuro, allora esso è anche CPA-(t, Q, ϵ)-sicuro per cifrare messaggi multipli.*

Dimostrazione. In realtà, l'asserto è un semplice corollario del Teorema 3.2 che abbiamo dimostrato nel Capitolo 3. Tale teorema, infatti, assicura che se due insiemi di distribuzioni sono ϵ-indistinguibili , allora gli insiemi composti da un numero polinomiale $L = \mathrm{poly}(n)$ di campioni prelevati (separatamente) dai due insiemi sono $(\epsilon \cdot L)$-indistinguibili. Presentiamo quindi solo una bozza di dimostrazione.

Useremo l'argomento ibrido (cf. Teorema 3.1). Per ogni $0 \leq i \leq L$, definiamo le seguenti distribuzioni ibride:

$$\mathcal{H}_n^i = (\underbrace{\mathsf{Enc}_{pk}(m_0^1), \ldots, \mathsf{Enc}_{pk}(m_0^i)}_{i \text{ elementi}}, \underbrace{\mathsf{Enc}_{pk}(m_1^{i+1}), \ldots, \mathsf{Enc}_{pk}(m_1^L)}_{L-i \text{ elementi}}).$$

Notare che il primo ibrido $\mathcal{H}_n^0 \stackrel{d}{\equiv} \mathbf{c}_1$ è distribuito come il vettore contenente la cifratura di tutti gli elementi in $\mathbf{m}_1$, mentre l'ultimo ibrido $\mathcal{H}_n^L \stackrel{d}{\equiv} \mathbf{c}_0$ è distribuito come il vettore contenente la cifratura di tutti gli elementi in $\mathbf{m}_0$. In altri termini, gli ibridi estremi hanno la stessa distribuzione di due possibili crittotesti sfida nell'esperimento che definisce la sicurezza di Π_{PKE} quando esso è usato per cifrare messaggi multipli.

Supponiamo che esista un attaccante PPT $\mathscr{A}$ (eseguibile in tempo t e che effettua Q richieste all'oracolo di cifratura) in grado di violare la sicurezza CPA del sistema quando esso è usato per cifrare messaggi multipli con vantaggio $> \epsilon$. Applicando il Teorema 3.1, ciò implica che $\mathscr{A}$ è in grado di distinguere due ibridi consecutivi $\mathcal{H}_n^i$ e $\mathcal{H}_n^{i+1}$ (per qualche $0 \leq i \leq L$) con vantaggio $> \epsilon/L$. Mostreremo come costruire un attaccante $\mathscr{B}$ in grado di violare la sicurezza CPA di Π_{PKE} (ovvero nel caso di cifratura di messaggi singoli, cf. Definizione 6.2) con lo stesso vantaggio. L'avversario $\mathscr{B}$ simula l'ambiente per $\mathscr{A}$ come segue:

1. Data la chiave pk lancia $\mathscr{A}(pk)$ per ottenere $\mathbf{m}_0$ ed $\mathbf{m}_1$.

2. Estrai a caso un indice $i \xleftarrow{\$} \{1, 2, \ldots, L\}$ ed invia al tuo oracolo i messaggi $m_0^i, m_1^i \in \mathcal{M}$. Ricevi in cambio il crittotesto sfida $c_b^i \in \mathcal{C}$ corrispondente alla cifratura di uno dei due messaggi.

3. Calcola

$$c_0^j \leftarrow \mathsf{Enc}_{pk}(m_0^j) \qquad \forall j = 1, \ldots, i-1$$
$$c_1^j \leftarrow \mathsf{Enc}_{pk}(m_1^j) \qquad \forall j = i+1, \ldots, L,$$

   e poni $\mathbf{c} = (c_0^1, \ldots, c_b^i, \ldots, c_1^L)$. Lancia quindi $\mathscr{A}(\mathbf{c})$ e ritorna lo stesso bit b' che decide $\mathscr{A}$.

Non è difficile vedere che la simulazione di $\mathscr{B}$ è perfetta; in particolare quando $b = 0$ il crittotesto sfida $\mathbf{c}$ per $\mathscr{A}$ è distribuito esattamente come l'ibrido $\mathcal{H}_n^i$, mentre quando $b = 1$ il crittotesto sfida $\mathbf{c}$ è distribuito esattamente come l'ibrido $\mathcal{H}_n^{i+1}$. Siccome (per un qualche indice j) l'avversario $\mathscr{A}$ è in grado di distinguere $\mathcal{H}_n^j$ da $\mathcal{H}_n^{j+1}$ con vantaggio $> \epsilon/L$, un'analisi identica a quella fatta nella parte finale della dimostrazione del Teorema 3.2 mostra che $\mathscr{B}$ può distinguere una cifratura di m_0^i da una di $m_1^i \in \mathcal{M}$ e quindi violare la sicurezza CPA di Π_{PKE}. Questo conclude la dimostrazione. □

(La dimostrazione è simile nel caso di cifrari CPA-sicuri (e CCA-sicuri) a chiave segreta; il motivo per cui abbiamo scelto di presentarla nel contesto asimmetrico è che qui sicurezza CPA e sicurezza IND sono equivalenti, quindi abbiamo potuto lavorare con la seconda semplificando l'esposizione.) La conseguenza principale del Teorema 6.1 è che possiamo usare un qualsiasi crittosistema che prende come input messaggi a lunghezza fissa, per costruirne uno che cifra messaggi a lunghezza arbitraria (con la stessa identica sicurezza). Mettiamoci nel caso estremo in cui $\Pi^1 = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ cifra messaggi lunghi un solo bit, i.e. $\mathcal{M} = \{0, 1\}$. Possiamo trasformare Π^1 in un cifrario che cifra messaggi in $\mathcal{M} = \{0, 1\}^*$ semplicemente cifrando i singoli bit di un messaggio $m \in \{0, 1\}^*$ usando Π^1. Il teorema precedente implica che, se Π^1 è CPA-sicuro (risp. CCA-sicuro), anche Π^* lo è.

Più in generale, dato un crittosistema che cifra messaggi lunghi ℓ bit, possiamo cifrare un messaggio lungo $\ell' \geq \ell$ bit attraverso $L = \lceil \ell'/\ell \rceil$ invocazioni dell'algoritmo di cifratura. In certi casi, ciò può essere ancora non pratico. È possibile ridurre la complessità computazionale e di comunicazione usando contemporaneamente cifratura simmetrica ed asimmetrica in un meccanismo

di incapsulamento della chiave (*Key Encapsulation Mechanism*, KEM o anche *cifratura ibrida*). L'idea è che Alice usi il cifrario asimmetrico per trasmettere un messaggio al Bianconiglio che sarà poi utilizzato come chiave nel cifrario simmetrico, per cifrare i messaggi successivi. (Notare che in questo modo Alice ed il Bianconiglio hanno praticamente condiviso una chiave segreta.)

Non introduciamo qui un modello formale, ma è semplice dimostrare che se i due cifrari sono CPA-sicuri, lo schema risultante è anch'esso CPA-sicuro (cf. Esercizio 6.5).

6.2 La fattorizzazione di interi ed RSA

Il primo crittosistema a chiave pubblica è comparso solo 10 anni dopo l'idea di Diffie-Hellman ed è dovuto a Rivest, Shamir ed Adleman [RSA78].[40] Lo schema è basato su alcuni concetti elementari di teoria dei numeri, pertanto si raccomanda una lettura dell'Appendice B per la comprensione di quanto segue. La sicurezza del sistema è in qualche modo collegata alla difficoltà di fattorizzare in modo efficiente numeri interi di un certo tipo, in particolare della forma $N = p \cdot q$ dove p, q sono primi a 1024 bit (cf. Appendice C.2 per una panoramica sui migliori algoritmi noti).

La versione base del cifrario RSA, indicata con Π_{RSA}, è mostrata nel Crittosistema 6.1. Sia i testi in chiaro che i testi cifrati sono elementi di $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$; lo spazio delle chiavi è

$$\mathcal{K} = \{(N, e, d) : \ N = p \cdot q, e \cdot d \equiv 1 \bmod \varphi(N)\},$$

dove $\varphi(N) = (p-1)(q-1)$ è la funzione toziente di Eulero (cf. Definizione B.5). Essenzialmente, RSA usa l'esponenziale modulare (nell'anello $\mathbb{Z}_N^*$) come permutazione con botola (cf. Definizione 6.3). La correttezza del sistema è basata sul piccolo Teorema di Fermat (cf. Teorema B.11) e sul fatto che $e \cdot d \equiv 1 \pmod{\varphi(N)}$, ovvero $e \cdot d = 1 + r \cdot \varphi(N)$ per qualche $r \in \mathbb{N}$. Infatti, applicando il piccolo Teorema di Fermat, possiamo verificare che la decifratura inverte correttamente la cifratura:

$$\mathsf{RSA}_{sk}^{-1}(c) \equiv (m^e)^d \equiv m^{1+r\cdot\varphi(N)} \equiv m \cdot m^{r\cdot\varphi(N)} \equiv m \pmod{N}.$$

[40] Clifford Cocks, un matematico britannico che lavorava per un dipartimento di spionaggio (il GCHQ), descrisse un cifrario molto simile in un documento interno nel 1973. I documenti furono mantenuti segreti e, visto il costo relativamente alto delle macchine necessario a quel tempo ad implementare lo schema, non ci furono ulteriori indagini né prove pratiche (la cosa fu considerata più che altro come una curiosità). La scoperta di Cocks fu resa pubblica solo nel 1997.

Crittosistema 6.1. Il cifrario RSA (versione base)

Sia $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$ (per un qualche intero N definito di seguito) e $\mathcal{K} = \{(N, e, d) : N = p \cdot q, e \cdot d \equiv 1 \bmod \varphi(N)\}$. Il cifrario $\Pi_{\mathsf{RSA}} = (\mathsf{Gen}, \mathsf{RSA}, \mathsf{RSA}^{-1})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo di generazione delle chiavi restituisce $(N, e, d) \leftarrow \mathsf{Gen}(1^n)$, dove $N = p \cdot q$ è prodotto di due primi di n bit, mentre d ed e sono interi tali che
$$d \cdot e \equiv 1 \pmod{\varphi(N)},$$
essendo $\varphi(\cdot)$ la funzione toziente di Eulero. La chiave pubblica è $pk = (N, e)$, la chiave segreta $sk = (d, p, q)$.
- `Cifratura.` Dato un messaggio $m \in \mathbb{Z}_N^*$ e la chiave pubblica pk, calcola
$$c = \mathsf{RSA}_{pk}(m) = m^e \bmod N.$$
- `Decifratura.` Dato il testo cifrato $c \in \mathbb{Z}_N^*$ e la chiave segreta sk, calcola
$$m = \mathsf{RSA}_{sk}^{-1}(c) = c^d \bmod N.$$

Cominciamo a discutere alcuni aspetti implementativi legati alla sicurezza del cifrario.

Generazione dei parametri. La generazione delle chiavi richiede di generare due primi p, q di dimensione opportuna. (Ciò non è affatto banale e tipicamente richiede l'uso di un *test di primalità*, cf. Appendice C.1.) Si calcola quindi $N = p \cdot q$ e $\varphi(N) = (p-1)(q-1)$. Si sceglie poi un valore casuale per e, con $1 < e < \varphi(N)$ e $\gcd(e, \varphi(N)) = 1$. Infine si calcola $d \equiv e^{-1} \pmod{\varphi(N)}$. (Un modo efficiente per fare ciò è attraverso l'algoritmo di Euclide esteso, cf. Appendice B.1.) Infine si cancellano tutti i valori intermedi, si memorizza in modo sicuro $sk = d$ e si pubblica $pk = (N, e)$.

Codifica binaria. Siccome $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$, abbiamo bisogno di definire un modo per mappare stringhe binarie in elementi di $\mathbb{Z}_N^*$. Sia $\ell = |N|$. Possiamo interpretare una stringa m lunga $\ell - 1$ bit come elemento di $\mathbb{Z}_N$. Potrebbe però accadere che l'elemento così generato non sia un elemento di $\mathbb{Z}_N^*$, se

$\gcd(m, N) \neq 1$. Osserviamo però che il processo di decifratura continua a funzionare correttamente; inoltre se m è scelto a caso, la probabilità di tale evento è molto bassa.[41]

Efficienza. Il costo computazionale del cifrario dipende dalla velocità nell'eseguire le operazioni modulari in $\mathbb{Z}_N^*$, che è polinomiale in N (cf. Appendice B.1).[42] L'esponenziale modulare è calcolabile efficientemente attraverso l'algoritmo *quadra e moltiplica* (detto anche *raddoppia e somma* nei gruppi additivi).

L'idea è quella di scrivere l'espansione binaria dell'esponente e, così che

$$x^e \equiv x^{\sum_{i=0}^{s-1} e_i 2^i} \equiv \prod_{i=0}^{s-1} \left(x^{2^i}\right)^{e_i} \equiv \left(\left(x^{e_{s-1}}\right)^2 x^{e_{s-2}}\right)^2 \dots x^{e_0} \pmod{N},$$

dove $e = (e_{s-1}, e_{s-2}, \dots, e_0)_2$ è la rappresentazione binaria di e. Pertanto sono necessarie (al più) $\log(e)$ moltiplicazioni modulari ed il costo è $O(\log(e) \log^\mu(N))$.

Si può velocizzare la decifratura usando il teorema del resto cinese (cf. Teorema B.12). In particolare siccome $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ si può calcolare

$$\begin{aligned} m_p &= c^d \equiv c^{d \bmod (p-1)} \pmod{p} \\ m_q &= c^d \equiv c^{d \bmod (q-1)} \pmod{q}. \end{aligned}$$

Il messaggio m è quindi ricostruibile attraverso la *formula di Garner*:

$$m \equiv \left(\left((m_p - m_q)(q^{-1} \bmod p)\right) \bmod p\right) \cdot q + m_q \pmod{N},$$

com'è facile verificare. Osserviamo che i valori $d \bmod (p-1)$ e $d \bmod (q-1)$ possono essere pre-calcolati e memorizzati.

Esistono accorgimenti per ridurre la complessità fino ad un fattore 27, si veda ad esempio [Pai03].

Scelta di e. Ogni valore di $1 < e < \varphi(N)$ è utilizzabile in teoria, a patto che esso sia invertibile modulo $\varphi(N)$, i.e. $\gcd(e, \varphi(N)) = 1$. Questo implica che e

[41] Supponiamo di aver trovato un elemento $x \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$. Allora, $x \bmod q = 0$ (per $x \neq 0$) ed abbiamo trovato un fattore di N, i.e. $\gcd(x, N) = q$.

[42] Normalmente la complessità associata alla moltiplicazione ed alla divisione in $\mathbb{Z}_N^*$ è $O(\log^\mu N)$, dove $\mu = 2$ oppure $\mu = 1 + \varepsilon$ (per $\varepsilon \in (0, 1]$) se si usano tecniche speciali come quelle definite in [Ber08].

deve essere dispari ed $e > 3$. È possibile, ad esempio, usare un valore di e fisso e generare i primi p, q in modo che $\gcd(e, (p-1)(q-1)) = 1$.

Usare $e = 3$ ha il vantaggio di rendere la cifratura e la distribuzione delle chiavi pubbliche più efficienti. D'altra parte, affinché $(p-1)(q-1)$ sia primo con $e = 3$, il valore $p-1$ non può essere multiplo di 3, il che porta a concludere $p \equiv 2 \pmod 3$. (Un discorso identico vale per q.) Siccome p deve essere dispari, deve aversi $p = 6k + 5$ il che rende meno veloce la generazione di p attraverso un test di primalità (perché p non solo deve essere primo, ma deve anche avere una forma particolare). Inoltre bisogna sempre tenere a mente il fatto che qualora si avesse $|m| < |N|/3$, tutti possono decifrare semplicemente estraendo la radice cubica.

Un altro problema è che esiste un attacco generale per violare RSA quando si usa un valore piccolo di e condiviso con più valori N. Sia ad esempio $e = 3$ e supponiamo che lo stesso messaggio m sia cifrato con tre differenti chiavi pubbliche $pk_1 = (N_1, 3)$, $pk_2 = (N_2, 3)$ e $pk_3 = (N_3, 3)$. Un attaccante vede i crittotesti:

$$c_1 \equiv m^3 \pmod{N_1} \qquad c_2 \equiv m^3 \pmod{N_2} \qquad c_3 \equiv m^3 \pmod{N_3}.$$

Supponiamo che $\gcd(N_i, N_j) \neq 1$ per $i \neq j$ (se ciò non accade uno dei moduli RSA può essere immediatamente fattorizzato, recuperando così m). Per il Teorema del resto cinese (cf. Teorema B.12) esiste un unico valore $\widehat{c}$ modulo $N = N_1 \cdot N_2 \cdot N_3$, tale che

$$\widehat{c} \equiv c_1 \pmod{N_1} \qquad \widehat{c} \equiv c_2 \pmod{N_2} \qquad \widehat{c} \equiv c_3 \pmod{N_3},$$

con $\widehat{c} \equiv m^3 \pmod N$. Siccome però $m < \min\{N_1, N_2, N_3\}$ possiamo concludere $m^3 < N$ e quindi, non essendoci riduzione modulare, possiamo calcolare m semplicemente estraendo la radice cubica di $\widehat{c}$.

Sulla fattorizzazione di N. Osserviamo che, data la chiave pubblica $pk = (N, e)$, la conoscenza di uno solo tra $p, q, \varphi(N)$ e d è sufficiente a violare RSA. Infatti:

- Dato p (risp. q), si può calcolare $q = N/p$ (risp. $p = N/q$) e quindi prima $\varphi(N)$, e poi d come l'inverso moltiplicativo di e modulo $\varphi(N)$ (attraverso l'algoritmo di Euclide esteso, cf. Appendice B.1).

- Dato $\varphi(N)$ è immediato verificare che $r := N - \varphi(N) + 1 = p + q$. Allora siccome $N = p \cdot q$, i fattori p e q possono essere calcolati cercando le soluzioni

dell'equazione di secondo grado

$$X^2 - rX + N = 0.$$

Ciò mostra che calcolare $\varphi(N)$ è tanto difficile quanto fattorizzare N.

- Noto d esiste un algoritmo probabilistico (cf. [Sti95, Capitolo 5]) per fattorizzare N.[43] L'algoritmo ha probabilità di successo media almeno $1/2$. Ciò implica che calcolare d è essenzialmente tanto difficile quanto fattorizzare N.

Wiener [Wie90] ha trovato un algoritmo polinomiale per fattorizzare N se $d < N^{1/4}/3$ e $q < p < 2q$.

La sicurezza del cifrario è legata direttamente al problema della fattorizzazione d'interi (cf. Appendice C.2): fattorizzare N è supposto computazionalmente infattibile per una scelta opportuna dei primi p e q.[44] Tuttavia non esiste alcuna dimostrazione del fatto che violare RSA sia equivalente a fattorizzare N (anche se ci sono risultati che vanno in questa direzione [AM09]), quindi a priori potrebbe esistere una strategia migliore. In particolare per dimostrare la sicurezza di RSA è necessario introdurre un'altra ipotesi computazionale.

Definizione 6.5 (Ipotesi RSA). Dati (N, e) come nel cifrario RSA, diremo che il problema RSA è (t, ϵ)-difficile se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathsf{IP}\left[\mathscr{A}(1^n, N, e, y) = x : \ y \xleftarrow{\$} \mathbb{Z}_N^*, x^e \equiv y \bmod N\right] \leq \epsilon.$$

■

Ciò è chiaramente equivalente ad affermare che è difficile calcolare radici e-sime in $\mathbb{Z}_N^*$.

Malleabilità e simbolo di Jacobi. La versione che abbiamo definito di RSA è *malleabile*: se $c_1, c_2 \in \mathcal{C}$ sono le cifrature dei messaggi $m_1, m_2 \in \mathcal{M}$, allora $c = c_1 \cdot c_2 \bmod N$ è la cifratura di $m = m_1 \cdot m_2 \bmod N$, come è banale verificare.

Mostriamo come l'attaccante può calcolare il simbolo di Jacobi (cf. Appendice B.3) di m. Ci occorre il seguente:

[43] Se quindi d è compromesso *non* è sicuro generarne uno nuovo mantenendo gli stessi valori di p e q.

[44] Per testare la sicurezza del cifrario, i laboratori RSA lanciano sfide relative alla fattorizzazione di alcuni valori N, con un numero crescente di cifre. L'ultima sfida, vinta nel 2010 [Kle+10], riguardava primi da 768 bit. Ad oggi, l'uso di RSA è consigliato con primi di almeno 1024 bit.

Lemma 6.2. *Per ogni primo p e per ogni intero e dispari risulta*

$$m^e \bmod p \in \mathbb{QR}_p \quad \Leftrightarrow \quad m \in \mathbb{QR}_p.$$

Dimostrazione. Sia g un generatore di $\mathbb{Z}_p^*$; allora $m^e = g^y$ per qualche intero y. Ovviamente $x \in \mathbb{QR}_p$ se e solo se x è una potenza pari di g. Ma, poiché e è dispari,

$$g^y e = m^e \equiv g^{ye \bmod p-1} \pmod{p}$$

è una potenza pari di g se e solo se g^y lo è. Quindi l'asserto. □

Sia $c \equiv m^e \pmod{N}$ una cifratura RSA. Il Lemma 6.2 ha le seguenti conseguenze:

- Calcolando $J_p(c)$ (risp. $J_q(c)$) l'avversario può imparare $J_p(m)$ (risp. $J_q(m)$). Tuttavia questo non è molto significativo, in quanto l'avversario non conosce p e q.

- Se l'avversario può verificare che $c \in \mathbb{QR}_N$, allora può concludere che anche $m \in \mathbb{QR}_N$. Tuttavia vedremo che è impossibile stabilire se $c \in \mathbb{QR}_N$ senza conoscere p e q. (In effetti, decidere se c è un residuo quadratico è equivalente a fattorizzare N, cf. Teorema 6.4.)

- L'avversario può calcolare il simbolo di Jacobi $J_N(m)$ relativo al messaggio m usando il solo testo cifrato c. Infatti

  $$J_N(c) = J_N(m^e) = J_p(m^e) \cdot J_q(m^e) = J_p(m) \cdot J_q(m) = J_N(m).$$

  (Sorprendentemente il simbolo di Jacobi $J_N(x)$ può essere calcolato efficientemente senza conoscere i fattori di N, si veda [ES98; BZ10].) Se questo è un problema oppure no, in generale, dipende dall'applicazione. Osserviamo comunque che questa "perdita" non contraddice l'ipotesi RSA, in quanto l'avversario può solo calcolare $J_N(m)$ e non m.

Versione randomizzata. Osserviamo che la versione di RSA descritta nel Crittosistema 6.1 *non* può essere CPA-sicura, in quanto l'algoritmo di cifratura è deterministico. Per ottenere sicurezza CPA, è necessario in qualche modo randomizzare RSA, ad esempio usando uno schema di riempimento (tale schema consente anche di cifrare messaggi di lunghezza inferiore a $|N|$). La versione randomizzata di RSA è mostrata nel Crittosistema 6.2.

Crittosistema 6.2. Il cifrario RSA (versione con riempimento)

Sia $\mathcal{M} = \mathcal{C} = \mathbb{Z}_N^*$ (per un qualche intero N definito di seguito), $\mathcal{K}$ come nel Crittosistema 6.1 ed $\ell(n) \leq 2n - 2$. Il cifrario $\Pi_{\widehat{\mathsf{RSA}}} = (\mathsf{Gen}, \widehat{\mathsf{RSA}}, \widehat{\mathsf{RSA}}^{-1})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n l'algoritmo di generazione delle chiavi restituisce $(N, e, d) \leftarrow \mathsf{Gen}(1^n)$, come nel Crittosistema 6.1.
- `Cifratura.` Dato un messaggio da cifrare $m \in \{0,1\}^\ell$ poni $\mathsf{pad}(m) = \omega$ dove $\omega \xleftarrow{\$} \{0,1\}^{|N|-\ell-1}$ ed interpreta la stringa $\widehat{m} = \mathsf{pad}(m)||m$ come un valore in $\mathbb{Z}_N^*$. Data la chiave pubblica pk, calcola:

$$c \leftarrow \widehat{\mathsf{RSA}}_{pk}(m) = (\widehat{m})^e \bmod N.$$

- `Decifratura.` Dato il testo cifrato $c \in \mathbb{Z}_N^*$ e la chiave segreta sk, calcola:

$$\widehat{m} = \widehat{\mathsf{RSA}}_{sk}^{-1}(c) = c^d \bmod N.$$

Il messaggio m è costituito dagli ℓ bit meno significativi di $\widehat{m}$.

Un esempio di schema di riempimento è fornito dallo standard PKCS # 1 [Lab98]. Si può mostrare che la versione randomizzata di RSA è CPA-sicura se vale l'ipotesi RSA. Si veda [Ale+88; HN04] per una dimostrazione.

Esiste un attacco a crittotesto scelto in grado di violare $\Pi_{\widehat{\mathsf{RSA}}}$ dovuto a Bleichenbacher [Ble98]. Tale attacco sfrutta alcune debolezze dello standard PKCS # 1, che consentono di recuperare il testo in chiaro cifrato con RSA inviando diversi milioni di crittotesti "correlati" al dispositivo di decifratura. Vista la sua natura, l'attacco ha evidenziato l'importanza della nozione di sicurezza CCA, non solo dal punto di vista teorico, ma anche dal punto di vista pratico. Per ottenere sicurezza CCA, nel 1994, Bellare e Rogaway [BR94] hanno definito il "cifrario asimmetrico ottimo con riempimento" (*Optimal Asymmetric Encryption Padding*, OAEP). Lo schema Π_{OAEP} è rappresentato nel Crittosistema 6.3. Bellare e Rogaway hanno dimostrato che OAEP è CCA-sicuro nel modello dell'oracolo casuale (cf. Paragrafo 4.4). Astraendo, il cifrario OAEP può essere come un "cifrario con riempimento" in cui prima si applica un riempimento $\widehat{m} = \pi(m, \omega)$ (con randomicità ω) del messaggio m, quindi si cifra m calco-

Crittosistema 6.3. Il cifrario Π_{OAEP}

Sia $\mathcal{M} = \{0,1\}^{n/2}$, $\mathcal{C} = \mathbb{Z}_N^*$ (per un qualche intero N definito di seguito) e $\mathcal{K}$ come nel Crittosistema 6.1. Assumiamo $|N| > 2n$. Siano $G(\cdot), H(\cdot)$ due mappe da $\{0,1\}^n$ a $\{0,1\}^n$. Il cifrario $\Pi_{\mathsf{OAEP}} = (\mathsf{Gen}, \mathsf{OAEP}, \mathsf{OAEP}^{-1})$ è definito come segue.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n l'algoritmo di generazione delle chiavi restituisce $(N, e, d) \leftarrow \mathsf{Gen}(1^n)$, come nel Crittosistema 6.1.
- `Cifratura.` Dato un messaggio da cifrare $m \in \{0,1\}^{n/2}$ estrai $\omega \xleftarrow{\$} \{0,1\}^n$. Posto $m' = m||0^{n/2}$ calcola $\widehat{m}_1 = G(\omega) \oplus m'$ e $\widehat{m}_2 = \omega \oplus H(\widehat{m}_1)$. Quindi poni $\widehat{m} = \widehat{m}_1||\widehat{m}_2$. Calcola il crittotesto:

$$c \leftarrow \mathsf{OAEP}_{pk}(m) \equiv (\widehat{m})^e \bmod N.$$

- `Decifratura.` Dato il testo cifrato $c \in \mathbb{Z}_N^*$ e la chiave segreta sk, recupera innanzitutto:

$$\widehat{m} = \mathsf{OAEP}_{sk}^{-1}(c) \equiv c^d \bmod N.$$

Converti in binario il risultato e dividilo come in $\widehat{m} = \widehat{m}_1||\widehat{m}_2$, con $|\widehat{m}_1| = |\widehat{m}_2| = n$. Recupera quindi $\omega = H(\widehat{m}_1) \oplus \widehat{m}_2$ ed $m' = \widehat{m}_1 \oplus G(\omega)$. Se gli ultimi $n/2$ bit di m' non sono zero ritorna $\perp$. Altrimenti il messaggio m è costituito dai primi $n/2$ bit di m'.

lando $f(\pi(m, \omega))$, dove f è una permutazione con botola. Recentemente Kiltz e Pietrzak [KP09] hanno dimostrato che, nel modello standard, è impossibile ridurre la sicurezza di un tale schema alle proprietà della permutazione con botola sottostante, anche quando tale permutazione è una permutazione "ideale". Questo risultato negativo, spiega perché è difficile ottenere una dimostrazione di sicurezza per OAEP nel modello standard.

6.3 Il logaritmo discreto ed ElGamal

Il crittosistema di ElGamal [Gam85] è basato sul problema del logaritmo discreto, ovvero sulla difficoltà a calcolare il logaritmo in alcuni gruppi algebrici

(cf. Appendice C.3). Sia $\mathbb{G}$ un gruppo ciclico di ordine q. Esiste un elemento $g \in \mathbb{G}$, detto generatore, tale che ciascun elemento $\alpha \in \mathbb{G}$ è esprimibile come $\alpha = g^a$ per qualche intero $a \in \mathbb{Z}_q$. L'elemento a è detto logaritmo discreto di α rispetto alla base g. L'idea di base è quella di usare il logaritmo discreto come funzione unidirezionale nel processo di cifratura e di creare una "botola" che consenta al ricevitore di decifrare il crittotesto usando la chiave segreta.

Il cifrario di ElGamal Π_{EG} è mostrato nel Crittosistema 6.4. Sia i testi in chiaro che i testi cifrati sono elementi di $\mathbb{G}$. Notare che tutte le operazioni sono eseguite in $\mathbb{G}$. In particolare, l'insieme dei testi in chiaro è $\mathcal{M} = \mathbb{G}$, l'insieme dei testi cifrati è $\mathcal{C} = \mathbb{G} \times \mathbb{G}$ e l'insieme delle chiavi è:

$$\mathcal{K} = \{(\mathbb{G}, q, a, g, \alpha) : \ \alpha = g^a \in \mathbb{G}\}.$$

Verifichiamo innanzitutto che la decifratura restituisce effettivamente il testo in chiaro m. In effetti:

$$\frac{c_2}{c_1^a} = \frac{m \cdot \alpha^b}{c_1^a} = \frac{m \cdot (g^a)^b}{(g^b)^a} = m.$$

Crittosistema 6.4. Il cifrario di ElGamal

Siano $\mathcal{M} = \mathbb{G}$, $\mathcal{C} = \mathbb{G} \times \mathbb{G}$ e $\mathcal{K} = \{(\mathbb{G}, q, a, g, \alpha) : \alpha = g^a \in \mathbb{G}\}$. Il cifrario $\Pi_{\mathsf{EG}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n l'algoritmo di generazione delle chiavi restituisce $(q, \mathbb{G}, g, a, \alpha) \leftarrow \mathsf{Gen}(1^n)$. Il gruppo $\mathbb{G}$ è ciclico di ordine q, l'elemento g è un generatore di $\mathbb{G}$ ed $a \xleftarrow{\$} \mathbb{Z}_q$. Poniamo $\alpha = g^a$. La chiave segreta è $sk = a$, la chiave pubblica $pk = (\mathbb{G}, g, q, \alpha)$.
- `Cifratura.` Dato un messaggio da cifrare $m \in \mathcal{M}$ e la chiave pubblica pk, estrai $b \xleftarrow{\$} \mathbb{Z}_q$; quindi calcola:

$$c \to \mathsf{Enc}_{pk}(m) = (c_1, c_2) = (g^b, \alpha^b \cdot m).$$

- `Decifratura.` Dato il testo cifrato $c \in \mathcal{C}$, con $c = (c_1, c_2)$, e la chiave segreta sk, calcola

$$m = \frac{c_2}{c_1^a}.$$

Logaritmi discreti ed ipotesi di Diffie-Hellman. Osserviamo innanzitutto che il crittosistema di ElGamal è randomizzato, pertanto possiamo sperare che sia CPA sicuro. La sicurezza è ovviamente connessa al problema del logaritmo discreto in $\mathbb{G}$: dato un elemento $\alpha \xleftarrow{\$} \mathbb{G}$ (con $\alpha = g^a$ per un qualche $a \in \mathbb{Z}_q$), non esistono algoritmi efficienti in grado di calcolare a, per una scelta opportuna del gruppo $\mathbb{G}$ e di q (cf. Appendice C.3 per una panoramica sulle tecniche note).

Definizione 6.6 (Ipotesi del logaritmo discreto). Il problema del logaritmo discreto è (t, ϵ)-difficile se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathbb{P}\left[\mathscr{A}(1^n, \mathbb{G}, q, g, \alpha) = a : \ \alpha \xleftarrow{\$} \mathbb{G}, \alpha = g^a\right] \leq \epsilon.$$

■

In realtà, mostreremo che il cifrario di ElGamal è CPA-sicuro sulla base di un'altra ipotesi computazionale, detta *ipotesi di Diffie-Hellman*. Il problema *computazionale* di Diffie-Hellman (*Computational Diffie-Hellman problem*, CDH) richiede di calcolare $g^{ab} \in \mathbb{G}$ noti $g^a, g^b \xleftarrow{\$} \mathbb{G}$.

Definizione 6.7 (Ipotesi computazionale di Diffie-Hellman). Il problema computazionale di Diffie-Hellman è (t, ϵ)-difficile se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathbb{P}\left[\mathscr{A}(1^n, \mathbb{G}, q, g^a, g^b) = g^{ab} : \ g^a, g^b \xleftarrow{\$} \mathbb{G}\right] \leq \epsilon.$$

■

Ovviamente se il problema del logaritmo discreto in $\mathbb{G}$ è facile, anche il problema CDH lo è: dati $\alpha = g^a$ e $\beta = g^b$ è sufficiente calcolare $a = \log_g(\alpha)$ e la soluzione al problema CDH è semplicemente $\beta^a = g^{ab}$. D'altra parte, non è noto se la difficoltà del problema del logaritmo discreto implica quella del problema CDH.

Esiste anche una versione decisionale del problema di Diffie-Hellman (*Decisional Diffie-Hellman problem*, DDH). Dati g, g^a, g^b ed un elemento $\zeta \in \mathbb{G}$, si richiede di stabilire se $\zeta = g^{ab}$, oppure se $\zeta \xleftarrow{\$} \mathbb{G}$ è un elemento casuale in $\mathbb{G}$.

Definizione 6.8 (Ipotesi decisionale di Diffie-Hellman). Il problema decisionale di Diffie-Hellman è (t, ϵ)-difficile in $\mathbb{G}$ se, per ogni avversario PPT $\mathscr{D}$

eseguibile in tempo t, risulta:

$$\left| \mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta = g^{ab}\right] - \mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{G}\right] \right| \leq \epsilon.$$

■

Come per il problema CDH, se il problema del logaritmo discreto è facile in $\mathbb{G}$, anche il problema DDH lo è. Basta infatti calcolare $a = \log_g(g^a)$, $b = \log_g(g^b)$ per poter distinguere $\zeta = g^{ab}$ da $\zeta \xleftarrow{\$} \mathbb{G}$. Non è complesso mostrare che la difficoltà del problema DDH implica la difficoltà del problema del logaritmo discreto in $\mathbb{G}$ (cf. Esercizio 6.9). D'altro canto, esistono esempi di gruppi in cui il problema del logaritmo discreto ed il problema CDH sono ritenuti difficili, ma il problema DDH non lo è.

Possiamo mostrare che il cifrario di ElGamal è CPA-sicuro se assumiamo che il problema DDH è difficile in $\mathbb{G}$.

Teorema 6.3 (Π_{EG} è CPA-sicuro). *Se il problema DDH è (t, ϵ)-difficile, Π_{EG} è CPA-(t, Q, ϵ)-sicuro, dove $Q = \mathrm{poly}(n)$ è il numero di richieste effettuate all'oracolo di cifratura in tempo t.*

Dimostrazione. Poiché siamo nel contesto asimmetrico, basta mostrare che lo schema è IND-sicuro (ovvero non dobbiamo preoccuparci dell'oracolo di cifratura) per ottenere sicurezza CPA con un numero arbitrario $Q = \mathrm{poly}(n)$ di richieste all'oracolo di cifratura in un attacco a messaggio scelto.

Supponiamo che il cifrario non sia CPA-sicuro: esiste un attaccante PPT $\mathscr{A}$ eseguibile in tempo t, che, effettuando Q richieste all'oracolo di cifratura, può violare la sicurezza CPA di Π_{EG} con vantaggio ϵ. Dato $\mathscr{A}$, mostriamo come costruire un attaccante PPT $\mathscr{D}$ in grado di risolvere il problema DDH in tempo polinomiale. L'attaccante $\mathscr{D}$ riceve come input gli elementi $(\mathbb{G}, q, g, g^a, g^b, \zeta)$ e deve decidere se $\zeta = g^{ab}$ oppure se $\zeta \xleftarrow{\$} \mathbb{G}$. Per fare ciò $\mathscr{D}$ usa $\mathscr{A}$ come segue:

1. Poni $pk = (\mathbb{G}, g, q, \alpha)$, con $\alpha = g^a$.
2. Lancia $\mathscr{A}(1^n, pk)$ ottenendo i messaggi $m_0, m_1 \in \mathcal{M}$ scelti da $\mathscr{A}$. Estrai $j \xleftarrow{\$} \{0, 1\}$ e ritorna ad $\mathscr{A}$ il crittotesto sfida $c_j = (g^b, \zeta \cdot m_j)$.
3. Ritorna lo stesso valore deciso da $\mathscr{A}$.

Dobbiamo distinguere due casi, a seconda della natura dell'oracolo di $\mathscr{D}$.

Il caso $\zeta \xleftarrow{\$} \mathbb{G}$. In questo caso l'elemento ζ è un elemento casuale in $\mathbb{G}$. Di conseguenza il crittotesto sfida è a distribuzione uniforme in $\mathbb{G} \times \mathbb{G}$, il che implica che, in particolare, $\mathscr{A}$ non può predire j con probabilità migliore di $1/2$. Pertanto:

$$\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{G}\right] \leq \frac{1}{2}.$$

Il caso $\zeta = g^{ab}$. È facile vedere che, in questo caso, $\mathscr{D}$ simula perfettamente l'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{EG}}}(\mathscr{A}, n)$ per $\mathscr{A}$ (nel senso che sia la chiave pubblica che il crittotesto sfida sono distribuiti esattamente come se $\mathscr{A}$ stesse attaccando il Crittosistema 6.4). Siccome per ipotesi il vantaggio di $\mathscr{A}$ nell'esperimento è ϵ, risulta:

$$\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta = g^{ab}\right] \geq \frac{1}{2} + \epsilon.$$

Mettendo tutto insieme, abbiamo trovato che $\mathscr{D}$ può risolvere il problema DDH con vantaggio:

$$\Big|\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta = g^{ab}\right] - \mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g, g^a, g^b, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{G}\right]\Big| \geq \epsilon,$$

contro l'ipoteso che il problema DDH sia (t, ϵ)-difficile. □

Concludiamo il paragrafo con una serie di considerazioni pratiche sull'utilizzo del cifrario.

Freschezza di b. Osserviamo che è importante usare un valore *diverso* di b per ogni messaggio da cifrare. In caso contrario infatti la cifratura di due messaggi distinti $m, m' \in \mathcal{M}$ è:

$$c = (g^b, \alpha^b \cdot m) = (c_1, c_2) \quad c' = (g^b, \alpha^b \cdot m') = (c_1, c_2'),$$

e quindi

$$c_2^{-1} \cdot c_2' = (\alpha^b \cdot m)^{-1} \cdot (\alpha^b \cdot m') = m^{-1} \cdot m',$$

ovvero $m' = m \cdot c_2^{-1} c_2'$.

Malleabilità. Data una cifratura $c = (c_1, c_2) = (g^b, \alpha^b \cdot m)$ di un messaggio $m \in \mathcal{M}$, possiamo scegliere un messaggio $m' \in \mathcal{M}$ e calcolare $c' = (g^b, m' \cdot c_2)$, corrisponde alla cifratura di $m'' = m' \cdot m$. Pertanto, il testo in chiaro può essere modificato lavorando sulla corrispondente cifratura.[45]

Scelta dei parametri. Come anticipato, è necessario scegliere il gruppo $\mathbb{G}$ in modo che il problema del logaritmo discreto sia difficile in $\mathbb{G}$. Esistono diverse possibilità:

- *Gruppi $\mathbb{G}$ con q primo.* Un primo esempio è dato da gruppi il cui ordine è un numero primo. In effetti, sono noti algoritmi in grado di ridurre un'istanza del logaritmo discreto in gruppi di ordine $q = q_1 \cdot q_2$ in due istanze separate in gruppi di ordine q_1 e q_2.[46] Notare che questo non significa che il problema del logaritmo discreto è facile in gruppi il cui ordine non è primo, significa solo che è *più facile* che in altri gruppi.

  Inoltre se q è primo ogni elemento di $\mathbb{G}$ è generatore, il che facilita la generazione delle chiavi. (Esistono comunque algoritmi probabilistici efficienti per trovare generatori di gruppi il cui ordine non è primo.) Un altro motivo per usare questi gruppi è che si può dimostrare che in una terna (g, g^a, g^b, g^{ab}), l'elemento $\zeta = g^{ab}$ è pseudocasuale, condizione necessaria (ma non sufficiente) affinché il problema DDH sia difficile in $\mathbb{G}$.

- *$\mathbb{Z}_p^*$ e i suoi sottogruppi.* Un'altra possibilità è quella di usare $\mathbb{Z}_p^*$, che è un gruppo ciclico quando p è primo. Anche in questo caso è possibile calcolare un generatore g in modo efficiente (cf. Appendice C.3).

  Il problema del logaritmo discreto è ritenuto difficile in questi gruppi. Tuttavia ci sono due problemi. Il primo è che l'ordine del gruppo, ovvero $q = p-1$, non è primo. Inoltre si può dimostrare che il problema DDH è sempre facile in $\mathbb{Z}_p^*$. Questi due problemi possono essere evitati passando ad un sottogruppo di $\mathbb{Z}_p^*$, ad esempio il sottogruppo $\mathbb{QR}_p$ dei residui quadratici in $\mathbb{Z}_p^*$ (cf. Appendice B.3). Sia p un primo della forma $p = 2q + 1$, con q primo (tali primi sono detti *primi forti*). Allora $\mathbb{QR}_p$ ha esattamente $(p-1)/2 = q$

[45] Notare che questo attacco non è previsto nelle nozioni di sicurezza che abbiamo definito finora. In effetti tale attacco minaccia l'*integrità* del messaggio e, come vedremo più avanti, può essere prevenuto attraverso l'uso di codici autenticatori di messaggio (cf. Capitolo 7) e di firme digitali (cf. Capitolo 8).

[46] Ciò è praticabile solo se la fattorizzazione di q è nota, il che è ragionevole se q ha divisori non troppo grandi.

elementi, è ciclico e ciascun elemento è un generatore del gruppo. In tale gruppo il problema DDH è ritenuto difficile.

- *Curve ellittiche.* Tutti i gruppo visti finora utilizzano l'aritmetica modulare. Un'altra classe di gruppi è quella dei gruppi dei punti di una curva ellittica su un campo finito (si veda l'Appendice B.4 per una breve introduzione alle curve ellittiche). Descriviamo qui una versione semplificata del crittosistema di ElGamal sulle curve ellittiche (detto *Elliptic Curve Integrated Encryption Scheme*, ECIES). Consideriamo la curva $E : Y^2 \equiv X^3 + AX + B \pmod{p}$ con p un primo ed $A, B \in \mathbb{Z}_p$. Si può dimostrare che l'insieme:

$$E(\mathbb{Z}_p) = \left\{(x, y) \in \mathbb{Z}_p : \ y^2 \equiv x^3 + Ax + B\right\} \cup \infty,$$

è un gruppo con un'opportuna operazione di somma. (Il punto ∞ è detto *punto all'infinito della curva* ed assume il ruolo di elemento neutro di tale somma.)

Per poter definire un crittosistema usando le curve ellittiche, dobbiamo definire un modo per mappare un testo in chiaro in un punto della curva. Tipicamente, si è soliti *comprimere* i punti della curva. Un punto $P = (x_P, y_P) \in E(\mathbb{Z}_p)$ è tale che le sue coordinate soddisfano $y_P^2 \equiv x_P^3 + Ax_P + B \pmod{p}$. Dato un valore $x_P \in \mathbb{Z}_p$, ci sono due valori possibili per y_P, l'uno l'opposto dell'altro modulo p. Siccome p è dispari, $y_P \bmod p$ è pari e $-y_P \bmod p$ è dispari. Poiché esistono algoritmi probabilistici polinomiali per calcolare radici quadrate modulo un primo p, possiamo rappresentare il punto $P \in E(\mathbb{Z}_p)$ comprimendolo in:

$$\Psi(P) = (x_P, y_P \bmod 2),$$

riducendo così l'occupazione di memoria di circa il 50%. Osserviamo che la mappa Ψ è del tipo $\Psi : E(\mathbb{Z}_p) \to \mathbb{Z}_p \times \mathbb{Z}_2$. L'algoritmo di decompressione $\Psi^{-1}(\cdot)$ restituisce il valore di P a partire dalla coppia $(x_P, y_P \bmod 2) \in \mathbb{Z}_p \times \mathbb{Z}_2$.

Sia $\mathbb{H} \subset E(\mathbb{Z}_p)$ un sottogruppo ciclico di ordine q, con q primo. Per una scelta opportuna di p, q, il problema del logaritmo discreto è ritenuto difficile in $\mathbb{H}$. Poniamo $\mathcal{M} = \mathbb{Z}_p^*$, $\mathcal{C} = (\mathbb{Z}_p \times \mathbb{Z}_2) \times \mathbb{Z}_p^*$ e

$$\mathcal{K} = \{E, q, P, R, a : \ R = [a]P\},$$

dove E, P, R, q sono i parametri pubblici ed $a \in \mathbb{Z}_q^*$ è la chiave segreta. La cifratura di un messaggio $m \in \mathbb{Z}_p^*$ avviene scegliendo a caso $b \in \mathbb{Z}_q^*$ e calcolando:

$$c = (c_1, c_2) = (\Psi([b]P), m \cdot \Upsilon_{[b]R} \bmod p),$$

dove $\Upsilon_{[b]R}$ è l'ascissa del punto $[b]R$. La decifratura di $c = (c_1, c_2)$ avviene calcolando $m = c_2 \cdot (\Upsilon_{[b]R})^{-1} \bmod p$; il valore $\Upsilon_{[b]R}$ può essere calcolato come ascissa del punto:

$$[a]\Psi^{-1}(\beta) = [a \cdot b]P = [b]R.$$

La sicurezza del cifrario dipende, in ultima istanza, anche dalla scelta dei parametri p e q. Quando $\mathbb{G} = \mathbb{Z}_p^*$ si raccomandano primi p con ≈ 300 cifre (più o meno 900 bit) per prevenire l'attacco basato sull'algoritmo del calcolo dell'indice (cf. Appendice C.3). Inoltre $p - 1$ deve essere non fattorizzabile in pratica. Quando invece $\mathbb{G}$ è un gruppo ciclico di ordine q (con q primo) sono sufficienti valori q a circa 160 bit, in quanto l'algoritmo del calcolo dell'indice non è efficace in tali gruppi.

6.4 Residuosità quadratica e Goldwasser-Micali

Un altro problema computazionale con svariate applicazioni in crittografia è il problema della *residuosità quadratica* (si veda l'Appendice B.3). Sia $N = p \cdot q$, con p e q primi distinti. Dato un elemento $x \in \mathbb{Z}_N^*$ la versione computazionale del problema consiste nel calcolare (se esiste) una radice quadrata di x.

Definizione 6.9 (Ipotesi Computazionale della Residuosità Quadratica). Diremo che il problema computazionale della residuosità quadratica è (t, ϵ)-difficile in $\mathbb{Z}_N^*$ (dove $N = p \cdot q$ con $|p| = |q| \approx n$ bit) se, per ogni avversario PPT $\mathscr{A}$ eseguibile in tempo t, risulta:

$$\mathbb{P}\left[\mathscr{A}(1^n, N, x) = y : \ x \xleftarrow{\$} \mathbb{QR}_N, x \equiv y^2 \bmod N\right] \leq \epsilon(n).$$

■

Non è complesso dimostrare che risolvere il problema computazionale della residusità quadratica è del tutto equivalente a fattorizzare N. Infatti:

Teorema 6.4 (Equivalenza tra fattorizzazione e problema QR computazionale). *Se il problema QR computazionale è $(t, 2\epsilon)$-difficile, non è possibile fattorizzare N in tempo $O(t)$ e con probabilità migliore di ϵ.*

Dimostrazione. Si tratta di mostrare che se è possibile fattorizzare N allora si può anche risolvere il problema della residuosità quadratica e viceversa. Un verso è banale, in quanto, noti p e q, è sufficiente calcolare la radice di x modulo

p e modulo q (il che è fattibile in tempo polinomiale, cf. Appendice B.3) e quindi modulo N attraverso il Teorema del resto cinese (cf. Teorema B.12).

D'altra parte, supponiamo che esista un'attaccante PPT $\mathscr{A}$ in grado di calcolare radici quadrate modulo N. Possiamo allora costruire un attaccante $\mathscr{B}$ in grado di fattorizzare N. L'attaccante $\mathscr{B}$ usa $\mathscr{A}$ come scatola nera nel modo seguente:

1. Scegli $x \xleftarrow{\$} \{0, 1, \ldots, N-1\}$ e calcola $\gcd(N, x)$. Se il risultato non è 1 abbiamo trovato un fattore di N.

2. Calcola $s \equiv x^2 \pmod{N}$ ed usa $\mathscr{A}$ per calcolare la radice quadrata x' di s modulo N. Se $x' = x$ oppure $x' = -x$ torna al passo (1).

3. Calcola $\gcd(N, x - x')$ e restituisci il risultato.

Rimane da calcolare la probabilità che tale algoritmo restituisca un fattore di N. Siccome ogni elemento di $\mathbb{QR}_N$ ha 4 radici quadrate modulo N (cf. Lemma B.18), la probabilità che $x \equiv \pm x' \pmod{N}$ è $1/2$. Quando ciò *non* accade:

$$x' \equiv x \pmod{p} \qquad \text{e} \qquad x' \equiv -x \pmod{q}.$$

(Oppure si verifica il caso simmetrico, scambiando p e q). Di conseguenza $x \not\equiv x' \pmod{N}$ e $x - x' \neq 0$. Siccome però $x' \equiv x \pmod{p}$, possiamo concludere:

$$\gcd(N, x - x') = p,$$

così che $\mathscr{B}$ trova un fattore di N con probabilità $2\epsilon/2 = \epsilon$. □

Un esempio di cifrario asimmetrico CPA-sicuro basato sul problema QR computazionale è il cifrario di Rabin [Rab79].

Il crittosistema di Goldwasser e Micali. Esiste anche una versione decisionale del problema della residuosità quadratica. In questo caso, l'attaccante vede un elemento $\zeta \in \mathbb{Z}_N^*$ e deve stabilire se questo è un residuo quadratico modulo N oppure no.

Definizione 6.10 (Ipotesi QR decisionale). Diremo che il problema decisionale della residuosità quadratica è (t, ϵ)-difficile in $\mathbb{Z}_N^*$ (dove $N = p \cdot q$ con

$|p| = |q| \approx n$ bit) se, per ogni avversario PPT $\mathscr{D}$ eseguibile in tempo t, risulta:

$$\left| \mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QR}_N\right] - \mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QNR}_N^{+1}\right] \right| \leq \epsilon(n),$$

avendo indicato con $\mathbb{QNR}_N^{+1}$ l'insieme dei non residui quadratici con simbolo di Jacobi $+1$. ■

È cruciale che nel secondo termine ζ sia estratto da $\mathbb{QNR}_N^{+1}$ e non semplicemente da $\mathbb{QNR}_N$. Infatti, è possibile mostrare che quando $\zeta \in \mathbb{QR}_N$ si ha $J_N(\zeta) = +1$, mentre quando $\zeta \in \mathbb{QNR}_N$ si ha $J_N(\zeta) = -1$ con probabilità $2/3$ (cf. Lemma B.18). In altri termini, è sempre possibile distinguere il caso $\zeta \xleftarrow{\$} \mathbb{QR}_N$ dal caso $\zeta \xleftarrow{\$} \mathbb{QNR}_N$ con vantaggio non trascurabile.

Goldwasser e Micali [GM82] hanno proposto il seguente semplice schema per cifrare un singolo bit: la cifratura del bit 0 è data da un elemento a caso in $\mathbb{QR}_N$, mentre la cifratura del bit 1 è data da un elemento a caso di $\mathbb{QNR}_N^{+}$. (La decifratura avviene decidendo se il crittotesto è un residuo quadratico oppure no, attraverso l'uso dei fattori di N.) È immediato dimostrare che tale crittosistema è CPA-sicuro se il problema decisionale della residuosità quadratica è difficile. Resta da capire come sia possibile selezionare un elemento casuale di $\mathbb{QR}_N$ e di $\mathbb{QNR}_N^{+1}$. Scegliere $x \xleftarrow{\$} \mathbb{QR}_N$ è semplice: si estrae $y \xleftarrow{\$} \mathbb{Z}_N^*$ e si pone $x = y^2 \bmod N$. Ovviamente, in questo modo, $x \in \mathbb{QR}_N$; inoltre x è casuale in $\mathbb{QR}_N$ perché elevare al quadrato modulo N è una mappa 4-a-1 (cf. Lemma B.18), e poiché y è esso stesso un elemento casuale di $\mathbb{Z}_N^*$.

Purtroppo, in generale, non è noto come scegliere un elemento casuale di $\mathbb{QNR}_N^{+1}$ quando la fattorizzazione di N non è nota (mentre ciò è sempre possibile quando sono noti i fattori di N). Questo problema suggerisce la seguente modifica: si usano p e q per scegliere un elemento casuale z in $\mathbb{QNR}_N^{+1}$ e si include z nella chiave pubblica del crittosistema. Se ora vogliamo estrarre $x \xleftarrow{\$} \mathbb{QNR}_N^{+1}$ senza conoscere la chiave segreta, è sufficiente estrarre $y \xleftarrow{\$} \mathbb{Z}_N^*$ e porre $x = z \cdot y^2 \bmod N$. Si può dimostrare che in questo modo (cf. Lemma B.21) l'elemento x è un elemento casuale in $\mathbb{QNR}_N^{+1}$, come desiderato. Il cifrario di Goldwasser e Micali Π_{GM} è presentato nel Crittosistema 6.5.

Teorema 6.5 (Π_{GM} è CPA-sicuro). *Se il problema decisionale della residuosità quadratica è (t, ϵ)-difficile rispetto al modulo $N = pq$, il crittosistema di Goldwasser e Micali è CPA-(t, Q, ϵ)-sicuro per ogni $Q = \mathrm{poly}(n)$.*

Crittosistema 6.5. Il cifrario di Goldwasser e Micali

Siano $\mathcal{M} = \{0,1\}$ e $\mathcal{C} = \mathbb{Z}_N^*$. Il cifrario $\Pi_{\mathsf{GM}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo di generazione delle chiavi restituisce $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$. La chiave pubblica è $pk = (z, N)$, con z un elemento casuale in $\mathbb{QINR}_N^{+1}$ ed $N = p \cdot q$ (dove p e q sono primi con $|p| = |q| \approx n$ bit). La chiave segreta è $sk = (p, q)$.
- `Cifratura.` Data la chiave pubblica $pk = (z, N)$ ed un bit $m \in \{0,1\}$, estrai $y \xleftarrow{\$} \mathbb{Z}_N^*$ e calcola
$$c \leftarrow \mathsf{Enc}_{pk}(m) = z^m \cdot y^2 \bmod N.$$
- `Decifratura.` Data la chiave segreta $sk = (p, q)$ ed un crittotesto $c \in \mathcal{C}$, usa i fattori di N per decidere se c è un residuo quadratico modulo N. In caso affermativo ritorna $m = 1$, altrimenti $m = 0$.

Dimostrazione. Come di consueto, è sufficiente mostrare che Π_{GM} è IND-sicuro, per assicurarsi che esso resista anche ad un attacco a messaggio scelto dove $\mathscr{A}$ effettua un numero polinomiale $Q = \mathrm{poly}(n)$ di richieste all'oracolo di cifratura.

Dato un attaccante PPT $\mathscr{A}$ in grado di violare la sicurezza del cifrario con vantaggio ϵ, mostriamo come costruire un attaccante PPT $\mathscr{D}$ in grado di distinguere un elemento casuale in $\mathbb{QR}_N$ da un elemento casuale in $\mathbb{QINR}_N^{+1}$, contro l'ipotesi che la versione decisionale del problema della residuosità quadratica sia difficile. Data la coppia (N, ζ), l'attaccante $\mathscr{D}$ deve decidere se $\zeta \xleftarrow{\$} \mathbb{QR}_N$ oppure se $\zeta \xleftarrow{\$} \mathbb{QINR}_N^{+1}$. Per fare ciò, $\mathscr{D}$ simula per $\mathscr{A}$ l'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{GM}}}(\mathscr{A}, n)$ come segue:

1. Poni $pk = (\zeta, N)$ e lancia $\mathscr{A}(1^n, pk)$ per ottenere i due bit $m_0, m_1 \in \{0,1\}$.
2. Scegli un bit casuale $b \xleftarrow{\$} \{0,1\}$ ed estrai $y \xleftarrow{\$} \mathbb{Z}_N^*$. Calcola il crittotesto sfida $c_b = \zeta^{m_b} \cdot y^2 \bmod N$.
3. Sia b' il bit restituito da $\mathscr{A}$. Se $b' = b$, ritorna 1, altrimenti ritorna 0.

Possiamo quindi distinguere due casi a seconda della natura dell'elemento ζ che $\mathscr{D}$ riceve come input.

Il caso $\zeta \xleftarrow{\$} \mathbb{QR}_N$. In questo caso mostreremo che ciò che $\mathscr{A}$ vede è indipendente da b. L'osservazione chiave è che, quando $\zeta \xleftarrow{\$} \mathbb{QR}_N$, il crittotesto sfida c_b è esso stesso un residuo quadratico casuale, *indipendentemente dal bit che si vuole cifrare*. Infatti, quando si vuole cifrare 0, l'elemento $c_b = y^2 \bmod N$ è ovviamente un residuo quadratico casuale. D'altra parte, quando si vuole cifrare 1, si ha $c_b = \zeta \cdot y^2 \bmod N$ con $y \xleftarrow{\$} \mathbb{Z}_N^*$. Siccome sia ζ che l'elemento $x = y^2 \bmod N$ sono residui quadratici casuali, anche il loro prodotto è casuale in $\mathbb{QR}_N$.

Ma allora ciò che $\mathscr{A}$ vede è indipendente da b e la probabilità che $b' = b$ è al più $1/2$. Quindi

$$\mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QR}_N\right] \leq \frac{1}{2}.$$

Il caso $\zeta \xleftarrow{\$} \mathbb{QNR}_N^{+1}$. In questo caso la chiave pubblica pk è distribuita esattamente come nel Crittosistema 6.5 e quindi la vista di $\mathscr{A}$ quando è lanciato da $\mathscr{D}$ è identica alla vista di $\mathscr{A}$ nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{GM}}}(\mathscr{A}, n)$. Siccome abbiamo supposto che il cifrario non sia CPA sicuro e poiché $\mathscr{D}$ restituisce 1 esattamente quando $\mathscr{A}$ restituisce 1, abbiamo

$$\mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QNR}_N^{+1}\right] = \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{GM}}}(\mathscr{A}, n) = 1\right] \geq \frac{1}{2} + \epsilon.$$

Mettendo tutto insieme, abbiamo mostrato

$$\left| \mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QR}_N\right] - \mathbb{P}\left[\mathscr{D}(1^n, N, \zeta) = 1 : \ \zeta \xleftarrow{\$} \mathbb{QNR}_N^{+1}\right] \right| \geq \epsilon,$$

contro l'ipotesi che la versione decisionale del problema della residuosità quadratica sia (t, ϵ)-difficile. □

L'efficienza del cifrario di Goldwasser-Micali è limitata dal fatto che il cifrario lavora su $\mathcal{M} = \{0, 1\}$ (quindi cifra singoli bit). Esiste una generalizzazione dovuta a Paillier [Pai99] (basata su una diversa ipotesi computazionale) che permette di cifrare messaggi più lunghi, migliorando l'efficienza.

6.5 Sicurezza CCA e Cramer-Shoup

Finora abbiamo visto diversi crittosistemi asimmetrici CPA-sicuri (nel modello standard). Il primo crittosistema (asimmetrico) CCA-sicuro ed efficiente allo stesso tempo è dovuto a Cramer e Shoup [CS98] ed è basato sul problema DDH. Il cifrario è mostrato nel Crittosistema 6.6. È facile verificare che, quando il crittotesto è costruito in modo legittimo

$$\begin{aligned} u^{a+\alpha a'} v^{b+\alpha b'} &= u^a v^b (u^{a'} v^{b'})^\alpha = (g_1^a g_2^b)^\omega (g_1^{a'} g_2^{b'})^{\omega\alpha} \\ &= \gamma^\omega \gamma'^{\omega\alpha} = (\gamma \cdot (\gamma')^\alpha)^\omega = r, \end{aligned}$$

così che il cifrario non ritorna mai $\perp$. Inoltre è immediato verificare che la decifratura restituisce effettivamente il testo in chiaro m.

Crittosistema 6.6. Il cifrario di Cramer e Shoup

Posto $\mathcal{M} = \mathbb{G}$ e $\mathcal{C} = \mathbb{G}^4$, sia $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}_H, H)$ una funzione hash (cf. Paragrafo 4.1). Il cifrario $\Pi_{\mathsf{CS}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo di generazione delle chiavi campiona $s \leftarrow \mathsf{Gen}_H(1^n)$ (ottenendo così la funzione hash H^s) e restituisce $(\mathbb{G}, q, g_1, g_2, H^s) \leftarrow \mathsf{Gen}(1^n)$. Il gruppo $\mathbb{G}$ è un gruppo finito di ordine un primo q con generatori g_1, g_2. La chiave pubblica è:

  $$pk = (g_1, g_2, \beta = g_1^x g_2^y, \gamma = g_1^a g_2^b, \gamma' = g_1^{a'} g_2^{b'}, H^s),$$

  dove $x, y, a, b, a', b' \in \mathbb{Z}_q$. La chiave segreta è $sk = (x, y, a, b, a', b')$.

- `Cifratura.` Dato il testo da cifrare $m \in \mathcal{M}$, scegli $\omega \xleftarrow{\$} \mathbb{Z}_q$ e calcola:

  $$c \leftarrow \mathsf{Enc}_{pk}(m) = (g_1^\omega, g_2^\omega, \beta^\omega \cdot m, (\gamma \cdot \gamma'^\alpha)^\omega),$$

  dove $\alpha = H^s(g_1^\omega, g_2^\omega, \beta^\omega \cdot m)$.

- `Decifratura.` Dato il crittotesto $c = (u, v, z, r)$, se $u^{a+\alpha a'} \cdot v^{b+\alpha b'} \neq r$ (per $\alpha = H^s(u, v, z)$) ritorna $\perp$. Altrimenti calcola:

  $$m = \mathsf{Dec}_{sk}(c) = \frac{z}{u^x v^y}.$$

Teorema 6.6 (Π_{CS} è CCA-sicuro). *Se il problema DDH è (t, ϵ)-difficile, e se Π_{HASH} è $(t_{\mathsf{HASH}}, \epsilon_{\mathsf{HASH}})$-sicura, il crittosistema di Cramer e Shoup è CCA-$(t_{\mathsf{CS}}, Q, \epsilon_{\mathsf{CS}})$-sicuro per ogni $Q = \mathrm{poly}(n)$, e con:*

$$t_{\mathsf{CS}} \approx t_{\mathsf{HASH}} \approx t \qquad \epsilon_{\mathsf{CS}} = \epsilon + \frac{Q}{q - Q + 1} + \epsilon_{\mathsf{HASH}}.$$

Dimostrazione. Dato un avversario PPT $\mathscr{A}$ che sia in grado di violare la sicurezza CCA del cifrario con vantaggio ϵ_{CS} come nell'enunciato del teorema, mostriamo come costruire un attaccante PPT $\mathscr{D}$ in grado di risolvere il problema DDH con vantaggio ϵ, contro l'ipotesi che quest'ultimo sia difficile. L'avversario $\mathscr{D}$ riceve in input gli elementi (g_1, g_2, g_3, g_4) e deve stabilire se questi sono del tipo DDH (ovvero se $g_3 = g_1^{\omega}$ e $g_4 = g_2^{\omega}$ per un qualche $\omega \xleftarrow{\$} \mathbb{Z}_q$) oppure no (ovvero se g_3 e g_4 sono elementi casuali).[47] Per fare ciò, $\mathscr{D}$ usa $\mathscr{A}$ come segue:

1. Estrai $x, y, a, b, a', b' \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\beta = g_1^x g_2^y$, $\gamma = g_1^a g_2^b$ e $\gamma' = g_1^{a'} g_2^{b'}$. Lancia $s \leftarrow \mathsf{Gen}_H(1^n)$ per ottenere H^s; poni $pk = (g_1, g_2, \beta, \gamma, \gamma', H^s)$ ed $sk = (x, y, a, b, a', b')$.

2. Lancia $\mathscr{A}(1^n, pk)$. Quando $\mathscr{A}$ interroga l'oracolo di decifratura, utilizza sk per rispondere alle richieste di $\mathscr{A}$ come definito nel Crittosistema 6.6.

3. Quando $\mathscr{A}$ sceglie i due messaggi $m_0, m_1 \in \mathcal{M}$ costruisci il crittotesto sfida come segue. Estrai $j \xleftarrow{\$} \{0, 1\}$ e ritorna:

$$c_j = (u, v, z, r) = (g_3, g_4, g_3^x g_4^y \cdot m_j, g_3^{a+\alpha a'} g_4^{b+\alpha b'}),$$

   dove $\alpha = H^s(u, v, z)$.

4. Continua ad usare sk per rispondere alle richieste di $\mathscr{A}$. Restituisci lo stesso valore che ritorna $\mathscr{A}$.

Dobbiamo distinguere due casi a seconda della natura dell'oracolo di $\mathscr{D}$.

Il caso $g_3 = g_1^{\omega}$ e $g_4 = g_2^{\omega}$. In questo caso è semplice verificare che la distribuzione delle chiavi (pk, sk) è identica a quella del Crittosistema 6.6 e che inoltre $\mathscr{D}$ simula perfettamente l'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{CS}}}(\mathscr{A}, n)$ per $\mathscr{A}$.

[47] Questa versione del problema DDH è del tutto equivalente a quella che abbiamo introdotto nel Paragrafo 6.3 se si pone $g_1 = g$, $g_2 = g^a$ ed $\omega = b$.

Siccome per ipotesi $\mathscr{A}$ è in grado di violare la sicurezza CCA del cifrario con vantaggio ϵ_{CS}, abbiamo:

$$\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g_1, g_2, g_3, g_4) = 1 : \ g_3 = g_1^{\omega}, g_4 = g_2^{\omega}\right] \geq \frac{1}{2} + \epsilon_{\mathsf{CS}}.$$

Il caso $g_3, g_4 \xleftarrow{\$} \mathbb{G}$. Mostreremo che in questo caso:

$$\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g_1, g_2, g_3, g_4) = 1 : \ g_3, g_4 \xleftarrow{\$} \mathbb{G}\right] \leq \frac{1}{2} + \frac{Q}{q - Q + 1} + \epsilon_{\mathsf{HASH}}, \quad (6.1)$$

il che conclude la dimostrazione, in quanto implica

$$\left| \mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g_1, g_2, g_3, g_4) = 1 : \ g_3 = g_1^{\omega}, g_4 = g_2^{\omega}\right] \right.$$
$$\left. - \mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g_1, g_2, g_3, g_4) = 1 : \ g_3, g_4 \xleftarrow{\$} \mathbb{G}\right] \right| \geq \epsilon_{\mathsf{CS}} - \frac{Q}{q - Q + 1} - \epsilon_{\mathsf{HASH}},$$

contro l'ipotesi che il problema DDH sia (t, ϵ)-sicuro.

In particolare, mostreremo che l'Eq. (6.1) è verificata anche se $\mathscr{A}$ è computazionalmente *illimitato*, ovvero anche quando $\mathscr{A}$ è in grado di calcolare logaritmi discreti in $\mathbb{G}$. Intuitivamente, vogliamo argomentare che ogni richiesta "pericolosa" di $\mathscr{A}$ all'oracolo di decifratura è rifiutata (cioè la decifratura ritorna $\perp$) e, condizionando su tale evento, concluderemo che $\mathscr{A}$ non ha alcuna informazione sul testo cifrato (perché non ha informazione su x ed y). Supponiamo che $\mathscr{A}$ sia in grado di calcolare logaritmi discreti, in particolare conosce $\xi = \log_{g_1}(g_2)$. Ispezionando la chiave pubblica, $\mathscr{A}$ può concludere

$$\log_{g_1}(\beta) = x + y \cdot \xi \quad (6.2)$$
$$\log_{g_1}(\gamma) = a + b \cdot \xi \quad (6.3)$$
$$\log_{g_1}(\gamma') = a' + b' \cdot \xi. \quad (6.4)$$

Siccome $g_3, g_4 \xleftarrow{\$} \mathbb{G}$, possiamo assumere $g_3 = g_1^{\omega_1}$ e $g_4 = g_2^{\omega_2}$, dove $\omega_1 \neq \omega_2$ con alta probabilità. Quando $\mathscr{A}$ riceve il crittotesto sfida

$$c_j = (u^*, v^*, z^*, r^*) = (g_3, g_4, g_3^x g_4^y \cdot m_j, g_3^{a+\alpha a'} g_4^{b+\alpha b'}),$$

impara anche che:

$$\log_{g_1}(r^*) = (a + \alpha a')\omega_1 + \xi(b + \alpha b')\omega_2. \quad (6.5)$$

Mostriamo ora che (con alta probabilità) ogni richiesta *invalida* che $\mathscr{A}$ sottopone all'oracolo di decifratura — ovvero una richiesta della forma (u, v, z, r) tale che $\log_{g_1}(u) \neq \log_{g_2}(v)$ — è rifiutata. Ovviamente $\mathscr{A}$ non può interrogare l'oracolo di decifratura usando $(u, v, z, r) = (u^*, v^*, z^*, r^*)$. Possiamo allora distinguere tre casi:

1. $(u, v, z) = (u^*, v^*, z^*)$, ma $r \neq r^*$. In questo caso è facile vedere che l'oracolo di decifratura ritorna sempre $\perp$.

2. $(u, v, z) \neq (u^*, v^*, z^*)$, ma $H(u, v, z) = H(u^*, v^*, z^*)$. Ciò significa che $\mathscr{A}$ ha trovato una collisione in $H(\cdot)$, il che può accadere solo con probabilità ϵ_{HASH}.

3. $(u, v, z) \neq (u^*, v^*, z^*)$, ma $H(u, v, z) \neq H(u^*, v^*, z^*)$. Indichiamo con $\hat{\alpha} = H(u, v, z)$, mentre il valore in Eq. (6.5) è $\alpha = H(u^*, v^*, z^*)$. Siano inoltre $\hat{\omega}_1 = \log_{g_1}(u)$ e $\hat{\omega}_2 = \log_{g_2}(v)$ e ricordiamo che stiamo considerando richieste invalide, cioè $\hat{\omega}_1 \neq \hat{\omega}_2$. Possiamo concludere che la prima richiesta invalida di $\mathscr{A}$ *non è* rifiutata dall'oracolo di decifratura se e solo se

$$\log_{g_1}(r) = (a + \hat{\alpha}a')\hat{\omega}_1 + \xi(b + \hat{\alpha}b')\hat{\omega}_2.$$

   Osserviamo che tale equazione è linearmente *indipendente* dalle Eq. (6.3)-(6.5), quindi consente di calcolare a, b, a', b'. Questo caso comunque si verifica solo quando $r^* = r$, il che accade con probabilità $1/q$.

Ne segue che la prima richiesta invalida di $\mathscr{A}$ è rifiutata con probabilità $1/q$. Ciò consente di escludere una tupla (a, b, a', b') — corrispondente ad un valore per r — ma ne restano ancora $q - 1$ possibili. Ripetendo lo stesso argomento possiamo concludere che la seconda richiesta invalida è rifiutata con probabilità $1/(q-1)$ ed in generale l'i-sima richiesta restituisce $\perp$ con probabilità $1/(q - i + 1)$. Quindi la probabilità che *una* di queste richieste *non* sia rifiutata è al più $Q/(q - Q + 1)$: siccome q è esponenziale in n mentre $Q = \text{poly}(n)$, tale probabilità è trascurabile.

Abbiamo dunque mostrato che con alta probabilità tutte le richieste invalide di $\mathscr{A}$ sono rifiutate. Per concludere la dimostrazione, resta da vedere cosa $\mathscr{A}$ ha imparato su x ed y (e quindi sul testo cifrato). Ovviamente, quando una richiesta invalida è rifiutata, $\mathscr{A}$ non impara niente su x ed y, in quanto tale richiesta è rifiutata solo sulla base dei valori a, b, a', b'. Quindi, dobbiamo considerare solo le richieste *valide*, ovvero quelle per cui $\hat{\omega} = \log_{g_1}(u) = \log_{g_2}(v)$.

Dalla risposta m a tale richiesta, $\mathscr{A}$ impara che $z/(u^x v^y) = m$, ovvero:

$$\log_{g_1}\left(\frac{z}{m}\right) = \hat{\omega} \cdot x + \xi \cdot \hat{\omega} \cdot y.$$

Tuttavia, tale equazione è linearmente *dipendente* dall'Eq. (6.2) e quindi nessuna informazione aggiuntiva su (x, y) è rivelata. Pertanto, con alta probabilità, quando $\mathscr{A}$ riceve il crittotesto sfida c_j, ci sono q valori equiprobabili per la coppia (x, y), il che implica che l'elemento $g_3^x g_4^y \cdot m_j$ è uniformemente distribuito ed indipendente da j, quindi:

$$\mathbb{P}\left[\mathscr{D}(1^n, \mathbb{G}, q, g_1, g_2, g_3, g_4) = 1 : \ g_3, g_4 \xleftarrow{\$} \mathbb{G}\right] \leq \frac{1}{2} + \frac{Q}{q - Q + 1} + \epsilon_{\mathsf{HASH}},$$

come richiesto. □

Esistono molti altri esempi di cifrari asimmetrici CCA-sicuri. Recentemente Hofheinz e Kiltz [HK09] ne hanno costruito uno molto efficiente basato sul problema della residuosità quadratica. Nel Paragrafo 13.4 studieremo una tecnica generale per rendere CCA-sicuro ogni crittosistema CPA-sicuro.

Esercizi

Esercizio 6.1. Sia $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un cifrario asimmetrico. Consideriamo la seguente variante dell'esperimento che definisce la sicurezza CPA.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n, b)$:
1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$;
2. $m_0, m_1 \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot)}(1^n, pk)$, con $|m_0| = |m_1|$;
3. $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{Enc}_{pk}(m_b)$;
4. restituisci $b' \leftarrow \mathscr{A}^{\mathsf{Enc}_{pk}(\cdot)}(c_b)$.

Supponiamo di dire che Π è CPA-(t, Q, ϵ)-sicuro se per ogni attaccante $\mathscr{A}$ eseguibile in tempo t si ha:

$$\left| \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n, 0) = 1\right] - \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{PKE}}}(\mathscr{A}, n, 1) = 1\right] \right| \leq \epsilon.$$

Mostrare che Π è CPA-(t, Q, ϵ)-sicuro nel senso di cui sopra se e solo se esso è anche CPA-$(t, Q, \epsilon/2)$-sicuro nel senso della Definizione 6.2.

Esercizio 6.2. Consideriamo il cifrario Π_{TDF} introdotto nel Paragrafo 6.1, basato sul bit estremo di un'arbitraria TDP. La chiave pubblica pk consiste nella descrizione della TDP $f : \mathcal{X} \to \mathcal{Y}$ e dell'insieme $\mathcal{X}$, la chiave segreta è $sk = tk$ (ovvero la botola). Per cifrare $m \in \{0,1\}$ si estrae $x \xleftarrow{\$} \mathcal{X}$, si calcolare $f(x) = y$ e si pone:

$$c \leftarrow \mathsf{Enc}_{pk}(m) = (y, \chi(x) \oplus m),$$

dove $\chi(x)$ è un bit estremo per $f(\cdot)$. Quindi, dato $c = (y, z)$ si recupera m invertendo y e calcolando:

$$m = \mathsf{Dec}_{sk}(c) = z \oplus \chi(f^{-1}(y)).$$

Mostrare che Π_{TDF} è CPA-sicuro.

Esercizio 6.3. Sia $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un cifrario a chiave pubblica. Consideriamo la seguente versione modificata di sicurezza CPA. L'avversario sceglie κ messaggi $(m_1, \ldots, m_\kappa)$ e riceve la cifratura corrispondente ad uno di essi (scelto a caso), ovvero $c_i \leftarrow \mathsf{Enc}_{pk}(m_i)$ per $i \xleftarrow{\$} [\kappa]$. Il cifrario è CPA-$(t, Q, \epsilon)$-sicuro se la probabilità che un attaccante eseguibile in tempo t ritorni $i' = i$ è al più $1/\kappa + \epsilon$.

1. Dare una definizione formale (introducendo un esperimento).

2. Mostrare che se Π è CPA-(t, Q, ϵ)-sicuro in senso classico, allora anche la definizione data al punto precedente è soddisfatta.

Esercizio 6.4. Sia $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un cifrario a chiave pubblica con spazio dei messaggi $\mathcal{M}^2$. Consideriamo il cifrario $\Pi^\pi = (\mathsf{Gen}, \mathsf{Enc}^\pi, \mathsf{Dec}^\pi)$ con spazio dei messaggi $\mathcal{M}$ derivato da Π come segue. Sia $\pi : \mathcal{M} \to \mathcal{M}$ una funzione di riempimento deterministica. L'algoritmo di generazione delle chiavi restituisce $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$. Per cifrare $m \in \mathcal{M}$ si calcola dapprima $\pi(m)$ e quindi si restituisce $c \leftarrow \mathsf{Enc}_{pk}(m, \pi(m)) = \mathsf{Enc}^\pi_{pk}(m)$. In decifratura si calcola $(m, m') = \mathsf{Dec}_{sk}(c) = \mathsf{Dec}^\pi_{sk}(c)$ e si restituisce m.

1. Mostrare che se Π è CCA-(t, Q, ϵ)-sicuro, allora Π^π è CCA-$(O(t/t_\pi), Q, \epsilon)$-sicuro, dove t_π è il tempo necessario a valutare π.

2. Vale lo stesso risultato se π è una funzione probabilistica?

Esercizio 6.5. Abbiamo accennato al concetto di cifratura ibrida. Rispondere alle seguenti domande:

1. Dare una definizione formale di sicurezza CPA per un cifrario ibrido.

2. Considerare il seguente schema di cifratura ibrida derivato da ElGamal con chiave pubblica $pk = (\mathbb{G}, q, g, \alpha)$. Sia $\Pi_{\mathsf{SKE}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un cifrario simmetrico con $\mathcal{K} = \mathbb{G}$. Per inviare un messaggio $m \in \mathcal{M}$, Alice sceglie la chiave segreta $k \xleftarrow{\$} \mathbb{G}$ ed un valore casuale $b \xleftarrow{\$} \mathbb{Z}_p$, quindi invia al Bianconiglio $c = (g^b, \alpha^b \cdot k, \mathsf{Enc}_k(m))$. Mostrare che il cifrario ibrido descritto è CPA-sicuro.

Esercizio 6.6. In un sistema RSA sono noti i valori $N = 18830129$ e $\varphi(N) = 18819060$. Ricavare p e q (senza fattorizzare N).

Esercizio 6.7. Consideriamo un sistema RSA con $pk = (N, e) = (143, 77)$.

1. Determinare la chiave segreta $sk = (p, q, d)$ e $\phi(N)$.

2. Cifrare il messaggio $m = 101$.

3. Decifrare il corrispondente crittotesto usando il Teorema cinese del Resto.

Esercizio 6.8. Consideriamo la seguente versione con riempimento di RSA. Dato un messaggio m, si estrae un valore r a caso e si pone $\widehat{m} = (0^n||r||0^8||m)$. Il crittotesto è quindi $c = \widehat{m}^e \bmod N$. Mostrare un attacco CCA al cifrario ed analizzarne la probabilità di successo.
(Suggerimento: sia $m_0 = 0 \dots 0$ ed $m_1 = 01 \dots 1$. Dato un crittotesto c_b che è la cifratura di m_0 oppure m_1, richiedere la cifratura di $c' = 2^e \cdot c_b \bmod N$. Se l'oracolo di cifratura restituisce $\perp$, ritornare un bit casuale. Altrimenti, se la risposta è $0 \dots 0$ ritornare $b = 0$, mentre se la risposta è $10 \dots 0$, restituiamo $b = 1$.)

Esercizio 6.9. Dare una dimostrazione formale del fatto che la difficoltà del problema DDH in un gruppo $\mathbb{G}$ implica la difficoltà del problema del logaritmo discreto in $\mathbb{G}$.

Esercizio 6.10. Consideriamo il seguente problema. Dati

$$(g, g^a \bmod p, g^b \bmod p, g^c \bmod p),$$

calcolare $g^{abc} \bmod p$. Assumendo che il problema CDH sia (t, ϵ)-difficile, mostrare che il problema di cui sopra è $(O(t), \epsilon')$-difficile (calcolando esplicitamente i parametri).
(Suggerimento: notare che è possibile estrarre radici x-sime modulo p, i.e. dato y è sempre possibile calcolare $y^{1/x} \bmod p$.)

Esercizio 6.11. Consideriamo il gruppo $\mathbb{Z}_p^*$ per $p = 2 \cdot q + 1$ e sia $g \in \mathbb{Z}_p^*$ un generatore.

1. Il problema del logaritmo discreto è ritenuto difficile in $\mathbb{Z}_p^*$. Ciò significa che la funzione $f : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ definita come $f(x) = g^x \bmod p$ è unidirezionale. Dimostrare che la funzione $\chi(x)$ che restituisce il bit meno significativo di x non è un predicato estremo per f.

2. Dimostrare che il problema DDH è facile in $\mathbb{Z}_p^*$.

Esercizio 6.12. Sia $\mathbb{G}$ un gruppo ciclico con ordine un primo q. Assumiamo che il problema del logaritmo discreto sia difficile in $\mathbb{G}$. Fissati g_1, g_2, g_3 in $\mathbb{G}$, diciamo che (x, y, z) è una rappresentazione di un elemento $h \in \mathbb{G}$ se e solo se $h = g_1^x \cdot g_2^y \cdot g_3^z$.

1. Fissiamo $h \in \mathbb{G}$. Quante rappresentazioni di h ci sono rispetto (g_1, g_2, g_3). Quante di queste soddisfano $x = x^*$ per un qualche $x^* \in \mathbb{Z}_q$? Quante soddisfano $x = x^*$, $z = z^*$ per fissati $x^*, y^* \in \mathbb{Z}_q$?

2. Supponendo che il problema del logaritmo discreto sia difficile in $\mathbb{G}$, mostrare che nessun attaccante PPT può prendere (g_1, g_2, g_3) come input e ritornare un elemento $h \in \mathbb{G}$ e due sue rappresentazioni distinte (sempre rispetto (g_1, g_2, g_3)).

3. Supponiamo di poter rappresentare gli elementi di $\mathbb{G}$ in stringhe lunghe $\log q + 1$. Sia $H^s : \mathbb{Z}_q^3 \to \mathbb{G}$ definita come $H^s(x_1, x_2, x_3) = g_1^{x_1} \cdot g_2^{x_2} \cdot g_3^{x_3}$ (per $s = (g_1, g_2, g_3)$ scelto a caso). Mostrare che H è resistente alle collisioni.

Esercizio 6.13. Considerare il seguente requisito di anonimato per un cifrario a chiave pubblica: fissate due chiavi pubbliche pk_0, pk_1, l'avversario non è in grado di distinguere se un crittotesto c è stato cifrato usando pk_0 oppure pk_1.

1. Dare una definizione formale di questa nozione, distinguendo il caso in cui l'avversario ha o non ha accesso agli oracoli di decifratura (i.e. CPA versus CCA) relativi alle chiavi segrete sk_0, sk_1. (Notare che il requisito di anonimato è del tutto ortogonale a quello di confidenzialità, quindi nella definizione si può considerare solamente il primo.)

2. Mostrare che il cifrario di ElGamal soddisfa la definizione data nel caso CPA.

Esercizio 6.14. Violare la sicurezza del cifrario di ElGamal con un attacco di tipo CCA.

Esercizio 6.15. Diciamo che un cifrario asimmetrico $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è omomorfo se per ogni n e per ogni $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$, esistono gruppi $\mathbb{M}, \mathbb{C}$ tali che

1. Lo spazio dei testi in chiaro è $\mathbb{M}$ e tutti i crittotesti ritornati da Enc_{pk} sono elementi di $\mathbb{C}$.

2. Per ogni $m_1, m_2 \in \mathbb{M}$ e $c_1, c_2 \in \mathbb{C}$ tali che $m_1 = \mathsf{Dec}_{sk}(c_1)$ ed $m_2 = \mathsf{Dec}_{sk}(c_2)$ si ha
$$\mathsf{Dec}_{sk}(c_1 \cdot c_2) = m_1 \cdot m_2,$$
dove le operazioni di gruppo sono effettuate in $\mathbb{C}$ ed $\mathbb{M}$ rispettivamente.

Mostrare che il cifrario di Goldwasser-Micali è omomorfo quando lo spazio dei messaggi $\mathcal{M} = \{0, 1\}$ è interpretato come il gruppo $\mathbb{Z}_2$.

Esercizio 6.16. Considerare la seguente versione semplificata del cifrario di Cramer-Shoup. La chiave pubblica è

$$pk = (g_1, g_2, \beta = g_1^x g_2^y, \gamma = g_1^a g_2^b),$$

dove $x, y, a, b, a', b' \in \mathbb{Z}_q$. La chiave segreta è $sk = (x, y, a, b)$. Dato il testo da cifrare $m \in \mathcal{M}$, scegli $\omega \xleftarrow{\$} \mathbb{Z}_q$ e calcola

$$c \leftarrow \mathsf{Enc}_{pk}(m) = (g_1^\omega, g_2^\omega, \beta^\omega \cdot m, \gamma^\omega) = (u, v, z).$$

In decifratura, se $r \neq u^a \cdot v^b$ restituisci $\perp$. Ritorna $m = z/(u^x \cdot v^y)$.
Mostrare che il sistema è CCA-1 sicuro ma non CCA-2 sicuro.

Letture consigliate

[Ale+88] Werner Alexi, Benny Chor, Oded Goldreich e Claus-Peter Schnorr. "RSA and Rabin Functions: Certain Parts are as Hard as the Whole". In: *SIAM J. Comput.* 17.2 (1988), pp. 194–209.

[AM09] Divesh Aggarwal e Ueli M. Maurer. "Breaking RSA Generically Is Equivalent to Factoring". In: *EUROCRYPT*. 2009, pp. 36–53.

[Ber08] Daniel J. Bernstein. *Fast Multiplication and its Applications, pp. 325–384 in Surveys in Algorithmic Number Theory.* J. P. Buhler and P. Stevenhagen, Math. Sci. Res. Inst. Publ. 44. Cambridge University Press, New York, 2008.

[Ble98] Daniel Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1". In: *CRYPTO*. 1998, pp. 1–12.

[BR94] Mihir Bellare e Phillip Rogaway. "Optimal Asymmetric Encryption". In: *EUROCRYPT*. 1994, pp. 92–111.

[BZ10] Richard P. Brent e Paul Zimmermann. "An $O(M(n) \log n)$ Algorithm for the Jacobi Symbol". In: *ANTS*. 2010, pp. 83–95.

[CS98] Ronald Cramer e Victor Shoup. "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack". In: *CRYPTO*. 1998, pp. 13–25.

[DH76] Whitfield Diffie e Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Trans. Inform. Theory* 22 (nov. 1976), pp. 644–654.

[ES98] Shawna Meyer Eikenberry e Jonathan Sorenson. "Efficient Algorithms for Computing the Jacobi Symbol". In: *J. Symb. Comput.* 26.4 (1998), pp. 509–523.

[Gam85] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[GM82] Shafi Goldwasser e Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *STOC*. 1982, pp. 365–377.

[HK09] Dennis Hofheinz e Eike Kiltz. "The Group of Signed Quadratic Residues and Applications". In: *CRYPTO*. 2009, pp. 637–653.

[HN04] Johan Håstad e Mats Näslund. "The security of all RSA and discrete log bits". In: *J. ACM* 51.2 (2004), pp. 187–230.

[Kle+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev e Paul Zimmermann. "Factorization of a 768-Bit RSA Modulus". In: *CRYPTO*. 2010, pp. 333–350.

[KP09] Eike Kiltz e Krzysztof Pietrzak. “On the Security of Padding-Based Encryption Schemes - or - Why We Cannot Prove OAEP Secure in the Standard Model”. In: *EUROCRYPT*. 2009, pp. 389–406.

[Lab98] RSA Laboratories. *RFC 2313: SPKI requirements*. See `http://tools.ietf.org/html/rfc2313`. The Internet Society. 1998.

[Pai03] Cesar A. M. Paixao. *An efficient variant of the RSA cryptosystem*. 2003. URL: `http://eprint.iacr.org/2003/159`.

[Pai99] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *EUROCRYPT*. 1999, pp. 223–238.

[Rab79] Michael Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*. Rapp. tecn. MIT/LCS/TR-212. Laboratory for Computer Science, Massachusetts Institute of Technology, gen. 1979.

[RSA78] Ronald L. Rivest, Adi Shamir e Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

[Wie90] Michael J. Wiener. “Cryptanalysis of short RSA secret exponents”. In: *IEEE Transactions on Information Theory* 36.3 (mag. 1990), pp. 553–558.

CAPITOLO 7

Tecniche simmetriche di integrità

Sii quel che sembri essere. Oppure, se vuoi metterla più semplicemente: Non immaginarti mai di non essere altrimenti da quello che potrebbe apparire agli altri che ciò che fosti o potresti essere stato non fosse altrimenti da quello che eri stato sarebbe apparso ad essi essere altrimenti.

Lewis Carroll, Le Avventure di Alice nel Paese delle Meraviglie [Car65]

Nei Capitoli 5 e 6 abbiamo introdotto le tecniche crittografiche che permettono di proteggere la confidenzialità dei messaggi scambiati in un arbitrario scenario di comunicazione. Come abbiamo già anticipato nel Paragrafo 1.1, questo non è l'unico requisito da soddisfare nel contesto di comunicazione sicura.

Mettiamoci nello scenario simmetrico, in cui esiste una chiave k condivisa tra Alice ed il Bianconiglio. Supponiamo che Alice voglia inviare al Bianconiglio un messaggio m e che ne cifri il contenuto usando un cifrario a segretezza perfetta. Qualora la Regina Rossa intercetti la corrispondente cifratura c, non ha alcun modo di recuperare il contenuto del messaggio m. Tuttavia nulla le vieta di sostituire il crittotesto c con un messaggio arbitrario $c^* \neq c$, così che il Bianconiglio decifrando c^* (attraverso la chiave k) otterrà un messaggio $m^* \neq m$. Il problema qui è che, nonostante il cifrario utilizzato celi perfettamente il contenuto del messaggio m, non ne protegge in alcun modo l'*integrità*. Per concentrarci solo sul requisito di integrità, supporremo che Alice invii il messaggio m in chiaro sul canale e che lo scopo della Regina sia quello di scambiare m con $m^* \neq m$ convincendo il Bianconiglio che Alice ha inviato m^*. (In altre parole, almeno inizialmente, non ci preoccuperemo di preservare la confidenzialità del messaggio m.)

Supponiamo che Alice ed il Bianconiglio condividano una chiave segreta k. Esistono primitive che consentano di conservare l'integrità di un messaggio m inviato (in chiaro) sul canale insicuro controllato dalla Regina?

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_7, © Springer-Verlag Italia 2012

La risposta a questa domanda risiede nei codici autenticatori di messaggio (*Message Authentication Code*, MAC). L'idea di fondo è la seguente: Alice usa la chiave k per creare un autenticatore ϕ relativo al messaggio m ed invia sul canale la coppia (m, ϕ). Il Bianconiglio può usare un *algoritmo di verifica* (efficiente e dipendente dalla chiave segreta k) per verificare che la coppia (m, ϕ) sia valida, ovvero che il messaggio m non sia stato alterato. Se il messaggio m fosse stato modificato dalla Regina, il sistema è progettato in modo che la verifica fallisca (con alta probabilità). Il requisito intuitivo di sicurezza per un MAC è l'*inforgiabilità*: la Regina non deve essere in grado di forgiare un MAC valido per un messaggio m^* a sua scelta.

Guida per il lettore. La struttura del capitolo è la seguente. Nel Paragrafo 7.1 daremo la definizione formale di MAC, insieme ad alcune nozioni relative alla sicurezza di questa primitiva. Vedremo quindi diversi paradigmi per costruire un codice autenticatore di messaggio: attraverso una funzione pseudocasuale (nel Paragrafo 7.2) ed attraverso una funzione hash (nel Paragrafo 7.3). Infine nel Paragrafo 7.4 vedremo come un MAC è utilizzabile insieme ad un cifrario simmetrico CPA-sicuro per realizzare un cifrario simmetrico CCA-sicuro ed analizzeremo alcuni approcci per ottenere simultaneamente i requisiti di confidenzialità ed integrità.

7.1 Nozioni di sicurezza per autenticatori simmetrici

Cominciamo con il definire formalmente un codice autenticatore di messaggio:

Definizione 7.1 (Codice autenticatore di messaggio). Indichiamo con $\mathcal{M}$ lo spazio dei possibili testi in chiaro, con Φ lo spazio degli autenticatori e con $\mathcal{K}$ lo spazio delle chiavi. Un codice autenticatore di messaggio (MAC) è una tripletta di algoritmi $\Pi_{\mathsf{MAC}} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ definita come segue:

- `Generazioni delle chiavi.` L'algoritmo Gen è l'algoritmo di generazione della chiave segreta. Dato il parametro di sicurezza n (dove n quantificherà la sicurezza del cifrario) restituisce la chiave $k \leftarrow \mathsf{Gen}(1^n)$ condivisa tra Alice ed il Bianconiglio.

- `Creazione degli autenticatori.` L'algoritmo $\mathsf{Tag} : \mathcal{K} \times \mathcal{M} \to \Phi$ è l'algoritmo di generazione degli autenticatori. Dato un messaggio $m \in \mathcal{M}$ da autenticare e la chiave condivisa k calcola l'autenticatore $\phi = \mathsf{Tag}_k(m)$.

- `Verifica.` L'algoritmo $\mathsf{Vrfy} : \mathcal{K} \times \mathcal{M} \times \Phi \to \{0,1\}$ è l'algoritmo di verifica degli autenticatori. Data una coppia messaggio-autenticatore (m, ϕ) e la chiave condivisa k abbiamo:

$$\mathsf{Vrfy}_k(m,\phi) = \begin{cases} 1 & \text{se } \phi = \mathsf{Tag}_k(m) \\ 0 & \text{altrimenti.} \end{cases}$$

■

Quando $\mathsf{Vrfy}_k(m,\phi) = 1$ diremo che l'autenticatore ϕ è un *autenticatore valido del messaggio* m. Quando l'algoritmo $\mathsf{Tag}(\cdot)$ è probabilistico scriveremo $\phi \leftarrow \mathsf{Tag}_k(m)$. Notare che se l'algoritmo $\mathsf{Tag}(\cdot)$ è deterministico, $\mathsf{Vrfy}(\cdot)$ calcola semplicemente $\phi = \mathsf{Tag}_k(m)$ e confronta il risultato con l'autenticatore ricevuto (pertanto non c'è bisogno di definire Vrfy esplicitamente).

Restano da definire i requisiti di sicurezza che vogliamo da un MAC. La definizione che segue è dovuta a Bellare, Kilian e Rogaway [BKR00]. Informalmente richiederemo che la Regina Rossa, pur conoscendo un numero arbitrario di autenticatori validi relativi a messaggi a sua scelta, non sia in grado di forgiare un MAC valido per un messaggio m^* "fresco" (cioè di cui non ha già visto il corrispondente autenticatore). Più formalmente dato un MAC $\Pi_{\mathsf{MAC}} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ come nella Definizione 7.1, consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_{\mathsf{MAC}}}(\mathscr{F}, n)$:
1. $k \leftarrow \mathsf{Gen}(1^n)$;
2. l'attaccante può richiedere $Q = \mathrm{poly}(n)$ autenticatori $\phi \in \Phi$ relativi a messaggi $m \in \mathcal{M}$ a sua scelta;
3. quindi $\mathscr{F}$ sceglie una coppia $(m^*, \phi^*) \in \mathcal{M} \times \Phi$;
4. restituisci 1 se e solo se:
   (i) $\mathsf{Vrfy}_k(m^*, \phi^*) = 1$ e
   (ii) il messaggio m^* è diverso dai messaggi al punto (2).

Definizione 7.2 (Inforgiabilità universale contro attacchi a messaggio scelto). Diremo che il MAC Π_{MAC} è (t, Q, ϵ)-ufcma (universalmente inforgiabile contro attacchi a messaggio scelto, *universally unforgeable against chosen message attacks* in inglese) se per ogni attaccante PPT $\mathscr{F}$ eseguibile in tempo t che effettua Q richieste nell'esperimento sopra definito, risulta

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_{\mathsf{MAC}}}(\mathscr{F}, n) = 1\right] \leq \epsilon.$$

■

Supponiamo di voler realizzare un MAC attraverso il blocco monouso (cf. Crittosistema 2.1), che ricordiamo essere incondizionatamente sicuro. Il MAC relativo ad un messaggio $m \in \mathcal{M}$ è calcolato come $\phi = m \oplus k$ e l'algoritmo di verifica $\mathsf{Vrfy}_k(m, \phi)$ ritorna 1 se e solo se $k = m \oplus \phi$. È semplice mostrare che questo schema non soddisfa la Definizione 7.2. Supponiamo infatti di voler forgiare un autenticatore valido per il messaggio $m^* \in \mathcal{M}$. Ci basta chiedere l'autenticatore $\phi = m \oplus k$ di un qualunque messaggio $m \neq m^*$, e ritornare (m^*, ϕ^*) dove $\phi^* = \phi \oplus m \oplus m^*$. In effetti, siccome

$$\phi^* = \phi \oplus m \oplus m^* = (m \oplus k) \oplus m \oplus m^* = m^* \oplus k,$$

si avrà $\mathsf{Vrfy}_k(m^*, \phi^*) = 1$, in quanto $m^* \oplus \phi^* = k$. (Sottolineiamo che esistono MAC incondizionatamente sicuri, anche se non ne parleremo; si veda [Sti95, Paragrafo 4.5] per una panoramica.)

Attacchi di tipo ri-uso. In un certo senso, la definizione di sicurezza data non tiene in considerazione gli *attacchi di tipo ri-uso*: la Regina può intercettare una coppia (m, ϕ) valida ed inviarla (completamente invariata), in un secondo momento, al Bianconiglio. Questo tipo di attacco ha successo perché l'algoritmo di verifica è *senza stato*. Esistono accorgimenti per evitare questo tipo di problema, ma non entreremo nel dettaglio. Per dare un'idea, se Alice ed il Bianconiglio condividono un "orologio", possono calcolare l'autenticatore del messaggio concatenato con il valore istantaneo dell'orologio; in questo modo ogni messaggio ha associato un "francobollo temporale" (*timestamp* in inglese) che ne rappresenta in qualche modo la freschezza. Ora un attacco di tipo ri-uso non ha più effetto, in quanto il valore di orologio non sarà più fresco quando la coppia (m, ϕ) viene replicata dalla Regina.[48]

Principio di Horton. In generale vale la regola di "autenticare ciò che si vuole dire, non ciò che viene detto"[49] Supponiamo che Alice voglia inviare un messaggio $m = A \mid\mid B \mid\mid C$ costituito dalla concatenazione di tre campi. Il Bianconiglio deve sapere come suddividere il messaggio nelle sue componenti originali: un MAC è semplicemente una stringa di bit, autentica il messaggio, ma non la sua struttura.

[48] Ovviamente stiamo omettendo diversi dettagli, ad esempio non è chiaro come si affronti il problema della sincronizzazione degli orologi. L'esempio riportato non vuole essere la soluzione al problema, ma solo dare un'idea di come esso può essere affrontato.

[49] Ovvero: "Authenticate what is being meant, not what is being said", citando [FS03, pagina 108]. La regola deve il nome al protagonista del film d'animazione *Horton Hears a Who.*

Crittosistema 7.1. Il MAC Π_{PRF} attraverso una PRF $F_k(\cdot)$

Posto $\mathcal{K} = \mathcal{M} = \Phi = \{0,1\}^n$, sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una PRF. Il MAC $\Pi_{\mathsf{PRF}} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ è definito come segue.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, genera la chiave condivisa $k \leftarrow \mathsf{Gen}(1^n)$, dove $k \xleftarrow{\$} \{0,1\}^n$.
- `Creazione dell'autenticatore.` Dato un messaggio $m \in \{0,1\}^n$ e la chiave $k \in \{0,1\}^n$ calcola l'autenticatore

$$\phi = \mathsf{Tag}_k(m) = F_k(m).$$

- `Verifica.` Data una coppia messaggio-autenticatore $(m, \phi) \in \{0,1\}^{2n}$ e la chiave $k \in \{0,1\}^n$, l'algoritmo $\mathsf{Vrfy}_k(m, \phi)$ restituisce 1 se e solo se $\phi = F_k(m)$.

7.2 MAC su base PRF

In questo paragrafo vediamo alcuni codici autenticatori di messaggio costruiti usando una funzione pseudocasuale (cf. Definizione 5.4). La costruzione più naturale è il MAC Π_{PRF}, mostrato nel Crittosistema 7.1 ed analizzato in [GGM84].

Teorema 7.1 (Π_{PRF} è ufcma, ovvero PRF $\Rightarrow$ MAC). *Se F è una PRF $(t_{\mathsf{PRF}}, \epsilon_{\mathsf{PRF}})$-sicura, il MAC Π_{PRF} è $(t_{\mathsf{MAC}}, Q, \epsilon_{\mathsf{MAC}})$-ufcma per ogni $Q = \mathrm{poly}(n)$, dove*

$$t_{\mathsf{MAC}} \approx t_{\mathsf{PRF}} \qquad \epsilon_{\mathsf{MAC}} \leq \epsilon_{\mathsf{PRF}} + 2^{-n}.$$

Dimostrazione. Mostreremo che se esiste un attaccante PPT $\mathscr{F}$ per il MAC Π_{PRF} (con vantaggio ϵ_{MAC} come nell'enunciato del teorema), allora possiamo costruire un attaccante PPT $\mathscr{D}$ in grado di distinguere $F_k(\cdot)$ (per una chiave k casuale) da una funzione veramente random $\$(\cdot)$ (scelta a caso tra le possibili mappe tra stringhe di n bit), con vantaggio maggiore di ϵ_{PRF}, contro l'ipotesi che $F_k(\cdot)$ sia una PRF sicura. L'attaccante $\mathscr{D}$ ha accesso ad un oracolo che riceve stringhe $m \in \{0,1\}^n$ e restituisce come risposta stringhe binarie $\zeta \in \{0,1\}^n$; l'avversario deve stabilire se $\zeta = F_k(m)$ (per una chiave casuale $k \xleftarrow{\$} \{0,1\}^n$) oppure se $\zeta = \$(m)$ (il che equivale a dire $\zeta \xleftarrow{\$} \{0,1\}^n$). Per fare ciò $\mathscr{D}$ usa $\mathscr{F}$ come segue:

1. Lancia $\mathscr{F}(1^n)$. Quando $\mathscr{F}$ richiede l'autenticatore di un messaggio $m \in \{0,1\}^n$, inoltra m all'oracolo. Sia $\zeta \in \{0,1\}^n$ la risposta dell'oracolo. Ritorna ad $\mathscr{F}$ la coppia (m, ζ).

2. Quando $\mathscr{F}$ restituisce la forgiatura (m^*, ϕ^*), se m^* è uguale ad uno dei Q messaggi richiesti da $\mathscr{F}$ al passo precedente ritorna 0. Altrimenti inoltra m^* all'oracolo e restituisci 1 se e solo se la risposta ζ^* dell'oracolo soddisfa $\zeta^* = \phi^*$.

Possiamo distinguere due casi, a seconda della natura dell'oracolo di $\mathscr{D}$.

L'oracolo di $\mathscr{D}$ è $F_k(\cdot)$. In questo caso la simulazione effettuata da $\mathscr{D}$ è perfetta, in quanto la distribuzione delle risposte dell'oracolo è la stessa degli autenticatori nel Crittosistema 7.1 (ovvero $\zeta = F_k(m)$ è esattamente l'autenticatore ϕ del messaggio m in Π_{PRF}). Pertanto

$$\mathbb{P}\left[\mathscr{D}^{F_k(\cdot)}(1^n) = 1\right] = \mathbb{P}\left[\mathbf{Exp}^{\text{ufcma}}_{\mathsf{MAC},\Pi_{\mathsf{PRF}}}(\mathscr{F}, n) = 1\right] \geq \epsilon_{\mathsf{MAC}}.$$

L'oracolo di $\mathscr{D}$ è $\$(\cdot)$. In questo caso tutti gli autenticatori richiesti da $\mathscr{F}$ sono stringhe uniformemente casuali in $\{0,1\}^n$. Pertanto, $\mathscr{F}$ non può forgiare un MAC con probabilità migliore di 2^{-n}. Quindi:

$$\mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1\right] \leq 2^{-n}.$$

Mettendo tutto insieme, abbiamo trovato:

$$\left|\mathbb{P}\left[\mathscr{D}^{F_k(\cdot)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{\$(\cdot)}(1^n) = 1\right]\right| \geq \epsilon_{\mathsf{MAC}} - 2^{-n},$$

così che (sostituendo l'espressione per ϵ_{MAC}) l'attaccante $\mathscr{D}$ può distinguere $F_k(\cdot)$ da $\$(\cdot)$ con probabilità migliore di ϵ_{PRF}. □

XOR-MAC. Il Crittosistema 7.1 può essere usato solo per autenticare messaggi a lunghezza fissa. Come abbiamo già fatto nel caso della cifratura, vorremmo ottenere uno schema in grado di autenticare messaggi a lunghezza *arbitraria*. Una soluzione a questo problema è stata fornita da Goldreich [Gol01]. Supponiamo di voler autenticare messaggi lunghi $2n$ bit usando Π_{PRF}, diciamo messaggi della forma $m = m_1||m_2$ (con $m_1, m_2 \in \{0,1\}^n$). L'autenticatore del messaggio m è allora $\phi = \phi_1||\phi_2$, dove $\phi_i = F_k(m_i)$ (per $i = 1, 2$). Questo

Crittosistema 7.2. L'autenticatore XOR-MAC

Posto $\mathcal{M} = \{0,1\}^*$, $\mathcal{K} = \{0,1\}^n$ e $\Phi = \{0,1\}^{2n-1}$, sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una PRF. L'autenticatore XOR-MAC è definito come segue.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, genera la chiave condivisa $k \leftarrow \mathsf{Gen}(1^n)$, dove $k \xleftarrow{\$} \{0,1\}^n$.
- `Creazione dell'autenticatore.` Sia $\ell < n$ ed indichiamo con $\langle i \rangle_{\ell-1}$ la rappresentazione dell'intero $i \in \mathbb{N}$ con $\ell - 1$ bit. Per autenticare un messaggio m lo dividiamo in B blocchi $m = m_1 || \ldots || m_B$ (ognuno lungo $(n - \ell)$ bit), scegliamo $\omega \xleftarrow{\$} \{0,1\}^{n-1}$ e calcoliamo:

$$\hat{\phi} = F_k(0 \mid\mid \omega) \oplus F_k(1 \mid\mid \langle 1 \rangle_{\ell-1} \mid\mid m_1) \\ \oplus F_k(1 \mid\mid \langle 2 \rangle_{\ell-1} \mid\mid m_2) \oplus \ldots \oplus F_k(1 \mid\mid \langle B \rangle_{\ell-1} \mid\mid m_B).$$

  Quindi $\phi \leftarrow \mathsf{Tag}_k(m) = (\hat{\phi}, \omega)$.
- `Verifica.` Data una coppia messaggio-autenticatore (m, ϕ) e la chiave $k \in \{0,1\}^n$, l'algoritmo di verifica restituisce 1 se e solo se ϕ è un autenticatore valido per m con randomicità ω.

schema non è sicuro, in quanto ad esempio $\phi^* = \phi_2 || \phi_1$ è un autenticatore valido di $m^* = m_2 || m_1 \neq m$.

Cerchiamo di risolvere il problema. L'idea di base è quella di usare un contatore. Sia $\langle i \rangle_n$ la rappresentazione dell'intero $i \in \mathbb{N}$ con n bit. Poniamo:

$$\phi = \mathsf{Tag}_k(m_1 \mid\mid m_2) = \phi_1 \mid\mid \phi_2 = F_k(\langle 1 \rangle_n \mid\mid m_1) \mid\mid F_k(\langle 2 \rangle_n \mid\mid m_2).$$

Sebbene è immediato verificare che questa modifica evita l'attacco precedente, lo schema non è affatto sicuro in quanto un avversario che vede *diversi* autenticatori validi, può ancora mischiarli per comporre l'autenticatore di un qualche altro messaggio. Per evitare questo secondo problema, possiamo pensare di definire:

$$\phi = \mathsf{Tag}_k(m_1 \mid\mid m_2) = \phi_1 \oplus \phi_2 = F_k(\langle 1 \rangle_n \mid\mid m_1) \oplus F_k(\langle 2 \rangle_n \mid\mid m_2).$$

Purtroppo neanche questa soluzione è sicura. Ecco un attacco. Supponiamo che l'attaccante abbia ottenuto l'autenticatore ϕ_1 per il messaggio m_1,

l'autenticatore ϕ_2 per $m_1||m_2$ e ϕ_3 per m_3. Allora:

$$\begin{aligned}\phi_1 &= F_k(\langle 1\rangle_n \,||\, m_1)\\ \phi_2 &= F_k(\langle 1\rangle_n \,||\, m_1) \oplus F_k(\langle 2\rangle_n \,||\, m_2)\\ \phi_3 &= F_k(\langle 3\rangle_n \,||\, m_3).\end{aligned}$$

Siccome $\phi_1 \oplus \phi_2 = F_k(\langle 2\rangle_n||m_2)$, la stringa $\phi^* = \phi_1 \oplus \phi_2 \oplus \phi_3$ è un autenticatore valido per $m^* = m_3||m_2$.

Questi tentativi portano a definire lo schema XOR-MAC, descritto nel Crittosistema 7.2. Osserviamo che il MAC è randomizzato e la randomicità ω è di fatto parte dell'autenticatore ϕ. Data una coppia (m, ϕ), l'algoritmo di verifica controlla che ϕ sia un MAC valido per il messaggio m, rispetto alla randomicità ω. Lo schema presuppone che il messaggio m abbia lunghezza multipla di $n - \ell$.[50] Inoltre la lunghezza massima consentita per i messaggi da autenticare è $(n - \ell) \cdot (2^{\ell-1} - 1)$ bit (affinché il contatore $\langle i\rangle_{\ell-1}$ non si ripeta).

Si può dimostrare che XOR-MAC è ufcma. Si rimanda a [BGR95] per i dettagli.

CBC-MAC. Uno dei codici autenticatori di messaggio più famosi è basato sul modo operativo CBC (cf. Paragrafo 5.4). Lo schema è mostrato nel Crittosistema 7.3. L'autenticatore è deterministico e genera MAC per messaggi costituiti da L blocchi lunghi n bit ciascuno (eventualmente usando uno schema di riempimento, come di consueto).

Lo schema è stato introdotto ed analizzato in [BKR00], dove si mostra che nessun attaccante PPT può forgiare un MAC con probabilità migliore di $\epsilon \leq 2B^2Q^2/2^n$. In [Mau02; PR00] il fattore moltiplicativo è ridotto da 2 ad 1. Bellare, Pietrzak e Rogaway [BPR05] hanno mostrato che $\epsilon \leq 20BQ^2/2^n$, per $B \leq 2^{n/3}$. CBC-MAC è standardizzato dall'ANSI [ANS81] e dall'ISO [ISO89]. Lo schema è sicuro solo se usato per autenticare messaggi a lunghezza fissa. Altrimenti esiste il seguente attacco. Supponiamo che $\mathscr{F}$ conosca l'autenticatore ϕ_1 di un messaggio $m_1 \in \{0,1\}^n$ e richieda l'autenticatore ϕ^* per ϕ_1. È facile vedere che ϕ^* è un autenticatore valido per $m^* = m_1||0^n$ (cf. Esercizio 7.7).

Possiamo istanziare CBC-MAC con una PRP P (ovvero un cifrario a blocco, cf. Definizione 5.5) anziché con una PRF. In questo caso bisogna prestare attenzione al seguente attacco, basato sul paradosso del compleanno. Sia $B > 2$ e siano $m_3, \ldots, m_B$ stringhe binarie in $\{0,1\}^n$. Consideriamo Q messaggi $m_1^1, \ldots, m_1^Q \in \{0,1\}^n$ ed altrettante stringhe casuali $m_2^1, \ldots, m_2^Q \in \{0,1\}^n$.

[50] È sempre possibile definire uno schema di riempimento non ambiguo che soddisfi tale vincolo.

Crittosistema 7.3. L'autenticatore CBC-MAC

Posto $\mathcal{M} = \{0,1\}^*$ e $\mathcal{K} = \Phi = \{0,1\}^n$, sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una PRF. L'autenticatore CBC-MAC è definito come segue.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, genera la chiave condivisa $k \leftarrow \mathsf{Gen}(1^n)$, dove $k \xleftarrow{\$} \{0,1\}^n$.
- `Creazione dell'autenticatore.` Dato in input un messaggio $m = m_1||\dots||m_B \in \{0,1\}^*$, si pone $\mathrm{IV} = \phi_0 = 0^n$ e si calcola:

$$\phi_i = F_k(m_i \oplus \phi_{i-1}) \qquad \forall\ i = 1, 2, \dots B.$$

  Quindi $\phi = \mathsf{Tag}_k(m) = \phi_B$.
- `Verifica.` Data una coppia messaggio-autenticatore (m, ϕ) e la chiave $k \in \{0,1\}^n$, l'algoritmo di verifica restituisce 1 se e solo se $\phi = \mathsf{Tag}_k(m)$.

Sia inoltre:

$$m(i) = m_1^i \ ||\ m_2^i \ ||\ m_3 \ ||\ \dots \ ||\ m_B \qquad \forall i = 1, 2, \dots, Q.$$

Osserviamo che $m(i) \neq m(j)$ se $i \neq j$. L'attaccante può richiedere gli autenticatori dei messaggi $m(1), \dots, m(Q)$. L'autenticatore CBC-MAC $\phi(i)$ del messaggio $m(i)$ è calcolato come nel Crittosistema 7.3: si sceglie $\mathrm{IV} = \phi_0^i = 0^n$, si calcola $\phi_j^i = P_k(m_j^i \oplus \phi_{j-1}^i)$ (dove $j = 1, 2, \dots, B$ e per $m_j^i = m_j$ quando $j \geq 3$) e si ottiene $\phi(i) = \mathsf{Tag}_k(m(i)) = \phi_B^i$. Il punto chiave è che $\phi(i) = \phi(j)$ se e solo se $\phi_2^i = \phi_2^j$, che a sua volta è vero quando e solo quando $m_2^i \oplus \phi_1^i = m_2^j \oplus \phi_1^j$ (perché ora $P_k(\cdot)$ è una biezione). Sia $z \xleftarrow{\$} \{0,1\}^n$ una stringa arbitraria. Definiamo i due messaggi:

$$u = m_1^i \ ||\ z \oplus m_2^i \ ||\ m_3 \ ||\ \dots \ ||\ m_B$$
$$v = m_1^j \ ||\ z \oplus m_2^j \ ||\ m_3 \ ||\ \dots \ ||\ m_B.$$

Dato l'autenticatore ϕ_u di u questo è un'autenticatore valido anche per v. Tale attacco ha successo solo se è possibile trovare una collisione, il che avviene con probabilità maggiore di $1/2$ quando $Q = O(2^{n/2})$ (per il paradosso del compleanno, cf. Teorema 4.1).

7.3 MAC su base hash

Un altro approccio per la costruzione di codici autenticatori di messaggio è attraverso l'uso di funzioni hash (cf. Capitolo 4). Data la funzione hash $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}, H)$, la prima idea che viene in mente per creare l'autenticatore di un messaggio $m \in \mathcal{M}$ è quella di calcolare l'hash della stringa ottenuta concatenando m con la chiave condivisa k, come in

$$\phi = H^s(m \mid\mid k) \qquad \text{oppure} \qquad \phi = H^s(k \mid\mid m). \tag{7.1}$$

Purtroppo queste soluzioni sono insicure a causa della natura iterativa delle funzioni hash stesse (cf. Paragrafo 4.2). Come abbiamo visto, dato un messaggio $m \in \{0,1\}^L$, l'idea è quella di suddividere il messaggio in B blocchi da ℓ bit (eventualmente usando un opportuno schema di riempimento) aggiungendo un ulteriore blocco m_{B+1} che codifica la lunghezza L con ℓ bit.[51] Si ottiene così una stringa è del tipo $m_1||m_2||\ldots||m_{B+1}$. Quindi, data una funzione di compressione $(\mathsf{Gen}, \mathsf{cmps})$ per messaggi a lunghezza fissa, diciamo $\mathsf{cmps} : \{0,1\}^{2\ell} \to \{0,1\}^{\ell}$, si sceglie $z_0 = \mathrm{IV} \in \{0,1\}^{\ell}$ e si calcola $z_i = \mathsf{cmps}^s(z_{i-1}||m_i)$. L'hash di m è infine $H^s(m) = z_{B+1}$. Se ora costruiamo un MAC usando gli schemi di Eq. (7.1), possiamo applicare i seguenti attacchi:

- *Estensione della lunghezza.* Supponiamo che venga inviato l'autenticatore (m, ϕ), dove $\phi = H^s(k||m)$. Sia m^* il messaggio costruito come

  $$m^* = m \mid\mid m_{B+2},$$

  per un qualche blocco arbitrario $m_{B+2} \xleftarrow{\$} \{0,1\}^{\ell}$. Allora esiste una scorciatoia per calcolare l'autenticatore ϕ^* del messaggio m^*, poiché

  $$\phi^* = H^s(k \mid\mid m \mid\mid m_{B+2}) = \mathsf{cmps}^s(\phi \mid\mid m_{B+2}).$$

- *Collisione parziale.* Supponiamo che venga inviato l'autenticatore (m, ϕ), dove $\phi = H^s(m||k)$. Se riusciamo a trovare una collisione in $\mathsf{cmps}^s(\cdot)$, vale a dire $\mathsf{cmps}^s(m) = \mathsf{cmps}^s(m^*)$ per $m \neq m^*$, la natura iterativa di $H^s(\cdot)$ implica che anche $H^s(m||k) = H^s(m^*||k)$, indipendentemente dal valore di k. L'autenticatore (m^*, ϕ) è quindi valido.

[51] Affinché ciò sia possibile i messaggi devono avere al più lunghezza $L \leq 2^{\ell}$.

HMAC. Un modo per evitare gli attacchi di cui sopra è quello di calcolare l'hash più di una volta (cf. Esercizio 4.8). Questo principio è sfruttato nello schema HMAC, come segue. Sia $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}, H)$ una funzione hash resistente alle collisioni, dato il messaggio $m \in \{0,1\}^*$ si pone

$$\phi = \mathsf{Tag}_k(m) = H^s\left(k^+ \oplus \mathsf{opad} \mid\mid H^s(k^+ \oplus \mathsf{ipad} \mid\mid m)\right),$$

dove k^+ è il riempimento della chiave aggiungendo un opportuno numero di bit 0 a destra (in modo da ottenere la lunghezza dell'input della funzione di compressione, ad esempio 512-bit per SHA-1) e per

$$\mathsf{opad} = 5\mathrm{C}5\mathrm{C}\ldots5\mathrm{C} \qquad \mathsf{ipad} = 3636\ldots36,$$

in esadecimale. Bellare, Canetti e Krawczyk [BCK96] hanno dimostrato che HMAC è ufcma nel modello dell'oracolo casuale (cf. Paragrafo 4.4). Lo schema è standardizzato dal NIST [NIS02].

Hash & MAC. Lo schema seguente definisce un paradigma generale per trasformare un autenticatore di messaggi a lunghezza fissa, in un autenticatore di messaggi a lunghezza arbitraria attraverso una funzione hash. Il paradigma, detto *"Hash & MAC"* è mostrato nel Crittosistema 7.4. L'idea di base è semplice: dato un MAC per messaggi lunghi n bit ed una funzione hash che mappa stringhe a lunghezza arbitraria in stringhe lunghe n bit, possiamo definire il MAC di un messaggio $m \in \{0,1\}^*$ calcolando prima $H(m)$ e quindi l'autenticatore ϕ ad esso relativo. Notare che la funzione $H(\cdot)$ è pubblica, solo la chiave k deve essere mantenuta segreta. Abbiamo dunque il seguente:

Teorema 7.2 (Il paradigma Hash & MAC è ufcma). *Se lo schema* Π_{MAC} *è* $(t_{\mathsf{MAC}}, Q, \epsilon_{\mathsf{MAC}})$*-ufcma e se* Π_{HASH} *è una funzione hash* $(t_{\mathsf{HASH}}, \epsilon_{\mathsf{HASH}})$*-sicura, l'autenticatore* Π^*_{MAC} *definito usando il paradigma "Hash & MAC" è* (t^*, Q, ϵ^*)*-ufcma, dove*

$$t_{\mathsf{HASH}} \approx t^* \approx t_{\mathsf{MAC}} \qquad \epsilon^* = \epsilon_{\mathsf{MAC}} + \epsilon_{\mathsf{HASH}}.$$

Dimostrazione. Sia $\mathscr{F}^*$ un attaccante PPT eseguibile in tempo t^* che effettua Q richieste nell'esperimento $\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*, n)$. Indichiamo con $\mathcal{Q}$ l'insieme dei messaggi che $\mathscr{F}^*$ invia all'oracolo e con (m^*, ϕ^*) la forgiatura creata da $\mathscr{F}^*$ al termine dell'esperimento. Sia $\mathcal{E}_C$ l'evento in cui $H(m^*) = H(m)$ per

Crittosistema 7.4. Il paradigma Hash & MAC

Sia $\Pi_{\mathsf{MAC}} = (\mathsf{Gen}_M, \mathsf{Tag}, \mathsf{Vrfy})$ un MAC per messaggi in $\mathcal{M} = \{0,1\}^n$. Sia inoltre $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}_H, H)$ una funzione hash. Consideriamo il MAC $\Pi^*_{\mathsf{MAC}} = (\mathsf{Gen}^*, \mathsf{Tag}^*, \mathsf{Vrfy}^*)$ definito come segue:

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n, l'algoritmo di generazione delle chiavi restituisce $(k,s) \leftarrow \mathsf{Gen}^*(1^n)$,dove $k \leftarrow \mathsf{Gen}_M(1^n)$ è la chiave condivisa ed $s \leftarrow \mathsf{Gen}_H(1^n)$ è l'indice della funzione hash.
- `Creazione dell'autenticatore.` Dato un messaggio $m \in \{0,1\}^*$, si calcola:
$$\phi = \mathsf{Tag}_k(H^s(m)).$$
- `Verifica.` Data una coppia messaggio-autenticatore (m,ϕ) e la chiave $k \in \{0,1\}^n$, l'algoritmo di verifica è definito come $\mathsf{Vrfy}^*_k(m,\phi) = \mathsf{Vrfy}_k(H(m),\phi)$.

qualcuno dei messaggi $m \in \mathcal{Q}$ scelti da $\mathscr{F}^*$. Abbiamo ovviamente

$$\begin{aligned}\mathsf{IP}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*,n)=1\right] &= \mathsf{IP}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*,n)=1 \wedge \mathcal{E}_C\right] \\ &\quad + \mathsf{IP}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*,n)=1 \wedge \overline{\mathcal{E}_C}\right] \\ &\leq \mathsf{IP}\left[\mathcal{E}_C\right] + \mathsf{IP}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*,n)=1 \wedge \overline{\mathcal{E}_C}\right].\end{aligned}$$

Mostreremo che entrambi i termini nell'equazione sopra sono trascurabili. Intuitivamente l'evento $\mathcal{E}_C$ è trascurabile perché Π_{HASH} è resistente alle collisioni. Infatti quando $\mathcal{E}_C$ si verifica possiamo costruire un attaccante PPT $\mathscr{A}$ eseguibile in tempo $t_{\mathsf{HASH}} \approx t^*$ in grado di trovare una collisione in Π_{HASH}. L'attaccante $\mathscr{A}$ riceve l'indice s come input. Quindi usa $\mathscr{F}^*$ come segue:

1. Calcola $k \leftarrow \mathsf{Gen}_M(1^n)$ e poni $k^* = (k,s)$.
2. Lancia $\mathscr{F}^*(1^n)$. Quando $\mathscr{F}^*$ richiede il MAC del messaggio $m_i \in \{0,1\}^*$ calcola $\phi_i = \mathsf{Tag}_k(H^s(m_i))$ e restituisci ϕ_i come risposta (per ogni $i = 1,\ldots,Q$).
3. Quando $\mathscr{F}^*$ restituisce la forgiatura (m^*,ϕ^*), cerca un indice $1 \leq j \leq Q$ per cui $H^s(m_j) = H^s(m^*)$. Ritorna (m_j, m^*).

Notare che la simulazione è perfetta, nel senso che la vista di $\mathscr{F}^*$ è distribuita esattamente come in un'esecuzione dell'esperimento $\mathbf{Exp}^{\text{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*, n)$. Inoltre, la probabilità che $\mathscr{A}$ trovi una collisione in Π_{HASH} è esattamente la probabilità dell'evento $\mathcal{E}_C$. Poiché Π_{HASH} è resistente alle collisioni per ipotesi, deve essere:

$$\mathbb{P}[\mathcal{E}_C] \leq \epsilon_{\mathsf{HASH}}.$$

Rimane da limitare il secondo termine. Per fare ciò mostreremo che quando $\mathscr{F}^*$ è in grado di forgiare un autenticatore valido (m^*, ϕ^*) (senza generare collisioni in Π_{HASH}), è possibile costruire un attaccante PPT $\mathscr{F}$ in grado di forgiare un autenticatore valido per Π_{MAC} in tempo $t_{\mathsf{MAC}} \approx t^*$, contro l'ipotesi che Π_{MAC} sia ufcma. L'attaccante $\mathscr{F}$ ha accesso all'oracolo $\mathsf{Tag}_k(\cdot)$ (per una data chiave $k \leftarrow \mathsf{Gen}_M(1^n)$). Quindi usa $\mathscr{F}^*$ come segue:

1. Calcola $s \leftarrow \mathsf{Gen}_H(1^n)$ e poni implicitamente $k^* = (k, s)$ (senza conoscere ovviamente k).

2. Lancia $\mathscr{F}^*(1^n)$. Quando $\mathscr{F}^*$ richiede l'autenticatore del messaggio $m_i \in \{0,1\}^*$, calcola $\hat{m}_i = H^s(m_i)$. Quindi inoltra $\hat{m}_i$ all'oracolo $\mathsf{Tag}_k(\cdot)$ e restituisci ad $\mathscr{F}^*$ l'autenticatore ϕ_i corrispondente (per ogni $i = 1, \ldots, Q$).

3. Quando $\mathscr{F}^*$ restituisce la forgiatura (m^*, ϕ^*), ritorna $(H^s(m^*), \phi^*)$.

Notare che la simulazione è perfetta, nel senso che la vista di $\mathscr{F}^*$ è distribuita esattamente come in un'esecuzione dell'esperimento $\mathbf{Exp}^{\text{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*, n)$. Inoltre quando $\mathscr{F}^*$ ritorna una forgiatura valida e contemporaneamente l'evento $\mathcal{E}_C$ *non* si verifica, la forgiatura restituita da $\mathscr{F}$ è anch'essa valida. Infatti: (i) se la coppia (m^*, ϕ^*) è una forgiatura valida si deve avere $\mathsf{Vrfy}^*_k(m^*, \phi^*) = \mathsf{Vrfy}_k(H^s(m), \phi^*) = 1$ e quindi la coppia $(H^s(m^*), \phi^*)$ è una forgiatura valida in Π_{MAC} e (ii) siccome $\mathcal{E}_C$ *non* si verifica $H^s(m^*) \neq H^s(m_i)$ (per ogni $i = 1, \ldots, Q$) e quindi il messaggio $H^s(m^*)$ per cui $\mathscr{F}$ forgia l'autenticatore è fresco. Siccome per ipotesi Π_{MAC} è ufcma, deve essere

$$\mathbb{P}\left[\mathbf{Exp}^{\text{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*, n) = 1 \wedge \overline{\mathcal{E}_C}\right] \leq \epsilon_{\mathsf{MAC}}.$$

Mettendo tutto insieme, abbiamo trovato:

$$\epsilon^* = \mathbb{P}\left[\mathbf{Exp}^{\text{ufcma}}_{\mathsf{MAC},\Pi^*_{\mathsf{MAC}}}(\mathscr{F}^*, n) = 1\right] \leq \epsilon_{\mathsf{HASH}} + \epsilon_{\mathsf{MAC}},$$

come desiderato. □

7.4 MAC e sicurezza CCA

Possiamo utilizzare un cifrario simmetrico CPA-sicuro ed un MAC per ottenere un cifrario simmetrico CCA-sicuro, come mostrato per la prima volta in [DDN00]. Sia $\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$ un cifrario simmetrico CPA-sicuro e sia $\Pi_M = (\mathsf{Gen}_M, \mathsf{Tag}, \mathsf{Vrfy})$ un MAC ufcma. Possiamo pensare di combinare Π_E e Π_M come segue per cifrare un messaggio m: si utilizza prima Π_E per ottenere il crittotesto c, successivamente si applica Π_M su c ottenendo l'autenticatore ϕ; si definisce quindi la cifratura di m come $c^* = (c, \phi)$. La decifratura richiede innanzitutto di verificare la validità dell'autenticatore, quindi di decifrare c. La costruzione è mostrata nel Crittosistema 7.5.

L'intuizione dietro la sicurezza CCA di Π^* è la seguente. Un crittotesto $c^* = (c, \phi)$ è *valido* (rispetto alle chiavi k_E, k_M) se $\mathsf{Vrfy}_{k_M}(c, \phi) = 1$. Le richieste che un avversario può inviare al suo oracolo di decifratura in un attacco CCA sono di due tipi: testi cifrati che $\mathscr{A}$ ha ricevuto dal suo oracolo di cifratura, oppure no. Il primo tipo di richieste non è molto utile, siccome l'avversario conosce già il corrispondente testo in chiaro m. Per quanto riguarda il secondo tipo di richieste, che chiameremo "fresche" nella dimostrazione, useremo il fatto che Π_M è ufcma per mostrare che tutte le richieste di questo tipo saranno invalide (tranne che per una probabilità trascurabile), così che $\mathscr{A}$ ottiene $\perp$ in

Crittosistema 7.5. Il cifrario Π^*, che combina un cifrario simmetrico CPA-sicuro ed un MAC ufcma

Siano $\Pi_E = (\mathsf{Gen}_E, \mathsf{Enc}, \mathsf{Dec})$ un cifrario simmetrico e $\Pi_M = (\mathsf{Gen}_M, \mathsf{Tag}, \mathsf{Vrfy})$ un MAC. Il cifrario Π^* è definito come segue:

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n, genera $k = (k_E, k_M) \leftarrow \mathsf{Gen}^*(1^n)$, dove $k_E \leftarrow \mathsf{Gen}_E(1^n)$ e $k_M \leftarrow \mathsf{Gen}_M(1^n)$ sono le chiavi condivise per Π_E e Π_M (rispettivamente).
- `Cifratura.` Data la chiave $k = (k_E, k_M)$ ed un messaggio m, calcola $c \leftarrow \mathsf{Enc}_{k_E}(m)$ e l'autenticatore $\phi \leftarrow \mathsf{Tag}_{k_M}(c)$. Il crittotesto è:
$$c^* \leftarrow \mathsf{Enc}_k(m) = (c, \phi).$$
- `Decifratura.` Data la chiave $k = (k_E, k_M)$ ed il crittotesto $c^* = (c, \phi)$, verifica innanzitutto che $\mathsf{Vrfy}_{k_M}(c, \phi) = 1$. Se questo è il caso calcola $m = \mathsf{Dec}_{k_E}(c)$, altrimenti ritorna $\perp$.

questo caso. Siccome l'oracolo di decifratura è "inutile", la sicurezza CCA di Π^* segue dalla sicurezza CPA di Π_E.

Formalmente abbiamo il seguente:

Teorema 7.3 (Π^* è CCA-sicuro). *Se Π_E è CPA-$(t_{\mathsf{CPA}}, Q_{\mathsf{CPA}}, \epsilon_{\mathsf{CPA}})$-sicuro e Π_M è un MAC $(t_{\mathsf{MAC}}, Q_{\mathsf{MAC}}, \epsilon_{\mathsf{MAC}})$-ufcma* con autenticatori univoci*, allora Π^* è CCA-$(t^*, Q_{\mathsf{CPA}} + Q_{\mathsf{MAC}}, \epsilon^*)$-sicuro, dove*

$$t_{\mathsf{CPA}} \approx t^* \approx t_{\mathsf{MAC}} \qquad\qquad \epsilon^* \leq \epsilon_{\mathsf{CPA}} + Q_{\mathsf{MAC}} \cdot \epsilon_{\mathsf{MAC}}.$$

Dimostrazione. Sia $\mathscr{A}$ un attaccante PPT che attacca Π^* nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n)$. Indichiamo con $\mathcal{E}_F$ l'evento in cui $\mathscr{A}$ invia all'oracolo di decifratura una richiesta (c, ϕ) "fresca", i.e. che non è stata ottenuta in precedenza usando l'oracolo di cifratura e tale che $\mathsf{Vrfy}_{k_M}(c, \phi) = 1$. Abbiamo

$$\begin{aligned} \frac{1}{2} + \epsilon^* = \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n) = 1\right] &\leq \mathbb{P}\left[\mathcal{E}_F\right] \\ &+ \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n) = 1 \wedge \overline{\mathcal{E}_F}\right]. \end{aligned} \tag{7.2}$$

Mostreremo che entrambi i termini in Eq. (7.2) sono trascurabili.

Intuitivamente la probabilità dell'evento $\mathcal{E}_F$ è trascurabile perché per inviare una richiesta valida all'oracolo di decifratura, $\mathscr{A}$ deve forgiare un autenticatore ϕ per un "nuovo" messaggio c (i.e., il cui corrispondente messaggio m non sia stato richiesto in precedenza all'oracolo di cifratura). Consideriamo allora un attaccante $\mathscr{F}$ che attacca Π_M (cioè che esegue l'esperimento $\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n)$). L'attaccante $\mathscr{F}$ conosce 1^n e ha accesso all'oracolo $\mathsf{Tag}_{k_M}(\cdot)$. Quindi $\mathscr{F}$ usa $\mathscr{A}$ come di seguito:

1. Estrai $k_E \leftarrow \mathsf{Gen}_E(1^n)$ uniformemente a caso e poni implicitamente $k = (k_E, k_M)$ (ovviamente senza conoscere k_M).

2. Scegli $i \xleftarrow{\$} \{1, \ldots, Q_{\mathsf{MAC}}\}$ uniformemente a caso.

3. Lancia $\mathscr{A}$ con input 1^n (che si aspetta di attaccare Π^* come nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n)$). Quando $\mathscr{A}$ invia un messaggio m all'oracolo di cifratura rispondi come segue:

   (i) calcola $c \leftarrow \mathsf{Enc}_{k_E}(m)$;

   (ii) invia c all'oracolo $\mathsf{Tag}_{k_M}(\cdot)$; sia ϕ la risposta dell'oracolo;

   (iii) rispondi alla richiesta di $\mathscr{A}$ con (c, ϕ).

4. Quando $\mathscr{A}$ invia una coppia (c, ϕ) all'oracolo di decifratura rispondi come segue:
   (i) se (c, ϕ) è una risposta relativa ad una precedente richiesta m effettuata da $\mathscr{A}$ all'oracolo di cifratura, ritorna m;
   (ii) se questa è l'i-sima richiesta all'oracolo di decifratura usando un valore di c *fresco* ritorna la coppia (c, ϕ) come forgiatura e fermati;
   (iii) altrimenti restituisci $\perp$.

5. Quando $\mathscr{A}$ sceglie i due messaggi m_0, m_1 estrai un bit $b \xleftarrow{\$} \{0,1\}$ per preparare il crittotesto sfida relativo ad m_b. Continua a rispondere alle richieste di $\mathscr{A}$ come nei punti (3) e (4).

In pratica $\mathscr{F}$ sta sperando che l'i-sima richiesta all'oracolo di decifratura da parte di $\mathscr{A}$ sia la prima di questo tipo ad essere "fresca", nel cui caso $\mathscr{F}$ ha forgiato un MAC valido per un messaggio c che non ha mai inviato all'oracolo $\mathsf{Tag}_{k_M}(\cdot)$.

Ovviamente $\mathscr{F}$ è eseguibile in tempo polinomiale. Inoltre la vista di $\mathscr{A}$ quando è lanciato da $\mathscr{F}$ è esattamente la stessa di quando $\mathscr{A}$ esegue l'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n)$, a patto che l'evento $\mathcal{E}_F$ *si verifichi* nell'i-sima richiesta che $\mathscr{A}$ invia al suo oracolo di decifratura. Ne segue che se $\mathscr{F}$ indovina l'indice i per cui $\mathscr{A}$ invia una richiesta (c, ϕ) "fresca" al suo oracolo di decifratura, allora $\mathscr{F}$ ha forgiato un MAC valido. La probabilità di indovinare i è ovviamente $1/Q_{\mathsf{MAC}}$, dove $Q_{\mathsf{MAC}} = \mathrm{poly}(n)$ è il numero di richieste che $\mathscr{A}$ può inviare al suo oracolo di decifratura; siccome Π_M è ufcma, abbiamo:

$$\epsilon_{\mathsf{MAC}} = \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n) = 1\right] \geq \frac{\mathbb{P}\left[\mathcal{E}_F\right]}{Q},$$

da cui $\mathbb{P}\left[\mathcal{E}_F\right] \leq Q \cdot \epsilon_{\mathsf{MAC}}$ e la probabilità dell'evento $\mathcal{E}_F$ è trascurabile.

Per limitare il secondo termine in Eq. (7.2) useremo il fatto che Π_E è CPA-sicuro. Consideriamo un avversario PPT $\mathscr{B}$, che attacca Π_E nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n)$. L'attaccante $\mathscr{B}$ conosce 1^n e ha accesso all'oracolo di cifratura $\mathsf{Enc}_{k_E}(\cdot)$. Quindi $\mathscr{B}$ usa $\mathscr{A}$ come di seguito:

1. Estrai $k_M \leftarrow \mathsf{Gen}_M(1^n)$ uniformemente a caso e poni implicitamente $k = (k_E, k_M)$ (ovviamente senza conoscere k_E).

2. Lancia $\mathscr{A}$ con input 1^n (che si aspetta di attaccare Π^* come nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n)$). Quando $\mathscr{A}$ invia un messaggio m al suo oracolo di cifratura rispondi come segue:

(i) inoltra m all'oracolo $\mathsf{Enc}_{k_E}(\cdot)$ e ricevi la risposta c;

(ii) calcola $\phi \leftarrow \mathsf{Tag}_{k_M}(c)$ e restituisci (c, ϕ) ad $\mathscr{A}$.

3. Quando $\mathscr{A}$ invia una richiesta (c, ϕ) al suo oracolo di decifratura, rispondi come segue:

   (i) se (c, ϕ) è una risposta relativa ad una precedente richiesta m all'oracolo di cifratura, ritorna m ad $\mathscr{A}$;

   (ii) altrimenti ritorna $\perp$.

4. Quando $\mathscr{A}$ sceglie i messaggi (m_0, m_1), restituisci gli stessi messaggi e ricevi il crittotesto sfida c_b. Calcola $\phi_b \leftarrow \mathsf{Tag}_{k_M}(c_b)$ ed invia (c_b, ϕ_b) come crittotesto sfida ad $\mathscr{A}$. Continua a rispondere alle richieste di $\mathscr{A}$ come nei punti (2) e (3).

5. Ritorna lo stesso bit b' che ritorna $\mathscr{A}$.

In pratica $\mathscr{B}$ simula l'oracolo di decifratura assumendo che $\mathscr{A}$ non inoltri mai una richiesta "fresca" che sia anche valida e quindi ritorna sempre $\perp$ in questo caso.

Ovviamente $\mathscr{B}$ è eseguibile in tempo polinomiale. Inoltre la vista di $\mathscr{A}$ nella simulazione definita da $\mathscr{B}$ è distribuita esattamente come la vista di $\mathscr{A}$ nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n)$ a patto che l'evento $\mathcal{E}_F$ non si verifichi. Ne segue che la probabilità che $\mathscr{B}$ abbia successo quando $\mathcal{E}_F$ non si verifica è la stessa che $\mathscr{A}$ abbia successo quando $\mathcal{E}_F$ non si verifica, ovvero

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n) = 1 \wedge \overline{\mathcal{E}_F}\right] = \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n) = 1 \wedge \overline{\mathcal{E}_F}\right].$$

Siccome, infine, Π_E è CPA-sicuro, abbiamo:

$$\begin{aligned}\frac{1}{2} + \epsilon_{\mathsf{CPA}} &\leq \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n) = 1 \wedge \overline{\mathcal{E}_F}\right] \\ &= \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi^*}(\mathscr{A}, n) = 1 \wedge \overline{\mathcal{E}_F}\right].\end{aligned}$$

Sostituendo nell'Eq. 7.2, troviamo:

$$\epsilon^* \leq Q_{\mathsf{MAC}} \cdot \epsilon_{\mathsf{MAC}} + \epsilon_{\mathsf{CPA}},$$

come desiderato. □

Osserviamo che si richiede che Π_M abbia *autenticatori univoci*, vale a dire per ogni chiave k e per ogni messaggio m esiste un unico valore ϕ tale che $\mathsf{Vrfy}_k(m, \phi) = 1$. Infatti, se ciò non fosse vero, potrebbe essere possibile modificare l'autenticatore ϕ per il crittotesto c in un diverso autenticatore ϕ' ancora valido per c. Quindi dato il crittotesto sfida (c_b, ϕ_b) un avversario potrebbe inviare (c_b, ϕ'_b) all'oracolo di decifratura, recuperando così m_b. Richiedere che gli autenticatori siano univoci non è molto stringente, in quanto tutti gli autenticatori che abbiamo visto (tranne XOR-MAC) soddisfano tale requisito.

Confidenzialità ed Autenticazione. Nel Capitolo 5 abbiamo studiato le tecniche simmetriche di cifratura. In questo capitolo abbiamo studiato le tecniche simmetriche per l'autenticazione di messaggio. In pratica potrebbe essere necessario soddisfare entrambi i requisiti, e si potrebbe erroneamente pensare che una combinazione arbitraria di un cifrario simmetrico sicuro con un codice autenticatore di messaggio sicuro è sufficiente per soddisfare tale necessità. Purtroppo non è questo il caso.

L'idea di base è quella di combinare un cifrario simmetrico con un MAC: volendo trasmettere un messaggio $m \in \mathcal{M}$ si invia una coppia di messaggi $(c, \phi) \in \mathcal{C} \times \Phi$ sul canale controllato dalla Regina. Il messaggio c sarà la cifratura di m e ϕ un opportuno autenticatore. Non introdurremo un modello formale (per il quale si rimanda ad esempio a [Kra01; BN08]). Intuitivamente, vorremmo definire uno schema che conservi insieme confidenzialità ed autenticazione: non deve essere possibile ricavare il contenuto di m a partire da c, né forgiare un autenticatore ϕ^* per un messaggio "fresco" m^*. Sia k_E una chiave di cifratura e sia k_M la chiave di un codice autenticatore di messaggi.[52] Possiamo individuare tre diversi approcci:

1. *Cifra ed autentica.* Cifratura ed autenticazione sono indirizzate indipendentemente. Vale a dire, dato m, Alice invia (c, ϕ), essendo:

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \qquad \phi \leftarrow \mathsf{Tag}_{k_M}(m).$$

   Il Bianconiglio recupera $m = \mathsf{Dec}_{k_E}(c)$ e controlla che $\mathsf{Vrfy}_{k_M}(m, \phi) = 1$. Se la verifica fallisce, ritorna $\perp$.

2. *Prima autentica e poi cifra.* Si calcola prima l'autenticatore e poi si cifrano messaggio ed autenticatore insieme. Vale a dire, dato m, Alice

[52] Un primo errore comune è usare la stessa chiave in entrambi gli schemi: ciò può essere fatto solo se si è dimostrato che tale uso delle chiavi è sicuro (cf. Esercizio 7.8).

trasmette c, dove:

$$\phi \leftarrow \mathsf{Tag}_{k_M}(m) \qquad c \leftarrow \mathsf{Enc}_{k_E}(m \mid\mid \phi).$$

Il Bianconiglio può decifrare il messaggio c e verificare l'autenticatore ϕ. Se $\mathsf{Vrfy}_{k_M}(m, \phi) \neq 1$, si restituisce $\perp$.

3. *Prima cifra e poi autentica.* Si cifra prima il messaggio m e se ne calcola l'autenticatore. Vale a dire, dato m, Alice invia (c, ϕ), dove:

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \qquad \phi \leftarrow \mathsf{Tag}_{k_M}(c).$$

Il Bianconiglio può verificare ϕ e quindi (se la verifica ha successo) decifrare c.

Si può vedere (cf. Esercizio 7.9 ed Esercizio 7.10) che i primi due approcci non sono sicuri per una combinazione *arbitraria* del cifrario Π_E e del MAC Π_M (pur sicuri quando usati singolarmente). Ciò nonostante, è bene sottolineare che questo non significa che non esistono opportune combinazioni di cifrari e codici autenticatori di messaggio in grado di ottenere sia confidenzialità che autenticazione quando usati come specificato, significa solo che non va bene usare un qualsiasi cifrario ed un qualsiasi autenticatore. (Un esempio è il caso del protocollo SSH [BKN06] che ha sicurezza dimostrabile [BKN04; PW10] seppure usi il paradigma "prima cifra e poi autentica"). Per quanto riguarda l'ultima combinazione, che è esattamente quella definita nel Crittosistema 7.5, abbiamo già mostrato che questa è CCA-sicura. Si può mostrare anche che non è possibile forgiare un autenticatore valido. Questa soluzione, pertanto, consente di ottenere sia confidenzialità che autenticazione (cf. Esercizio 7.11).

Esercizi

Esercizio 7.1. Sia $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ un MAC ufcma, e supponiamo che per una data chiave $k \xleftarrow{\$} \{0,1\}^n$ l'algoritmo Tag restituisca autenticatori di lunghezza $\ell(n)$. Mostrare che ℓ deve essere super-logaritmico, ovvero se $\ell(n) = O(\log n)$, allora $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ non può essere sicuro.

Esercizio 7.2. Estendere la Definizione 7.2 al caso in cui l'avversario ha accesso anche all'oracolo di verifica Vrfy.

1. In quali contesti questa definizione è importante?
2. Mostrare che un MAC ufcma con autenticatori univoci soddisfa anche la definizione data.
3. Mostrare che un MAC ufcma con autenticatori non univoci potrebbe non soddisfare la definizione data.

Esercizio 7.3. Sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una funzione pseudocasuale (t, ϵ)-sicura. Consideriamo il seguente MAC per autenticare messaggi lunghi $2n-2$. La chiave $k \in \{0,1\}^n$ è scelta a caso. L'autenticatore di un messaggio $m \in \{0,1\}^{2n-2}$ è calcolato dividendo m in $m = m_1||m_2$, con $|m_1| = |m_2| = n-1$ e ponendo

$$\phi = (F_k(0||m_0), F_k(1||m_1).$$

Mostrare che lo schema è insicuro.

Esercizio 7.4. Sia $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ una PRF. Dire se i seguenti MAC sono sicuri:

1. Sia $\mathcal{M} = \{0,1\}^{2n}$. Data la chiave condivisa $k \xleftarrow{\$} \{0,1\}^n$ ed un messaggio $m = m_1||m_2$ (dove $|m_1| = |m_2| = n$) si calcola $\phi = F_k(m_1, F_k(F_k(m_2)))$.
2. Sia $\mathcal{M} = \{0,1\}^{B \cdot n}$. Data la chiave condivisa $k \xleftarrow{\$} \{0,1\}^n$ ed un messaggio $m = m_1|| \ldots ||m_B$ (dove $|m_i| = n$) si calcola $\phi = F_k(m_1) \oplus \cdots \oplus F_k(m_B)$.

Esercizio 7.5. Consideriamo il seguente MAC costruito utilizzando un cifrario simmetrico $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ con $\mathcal{M} = \{0,1\}^n$ ed una funzione hash resistente alle collisioni H con codominio $\{0,1\}^n$. L'autenticatore ϕ di m è $\mathsf{Enc}_k(H(m))$ se $|m| > n$, altrimenti è semplicemente $\phi = \mathsf{Enc}_k(m)$.

1. Violare l'inforgiabilità esistenziale dello schema.
2. Suggerire una modifica che risolve il problema e giustificare la scelta.

Esercizio 7.6. Supponiamo di calcolare il MAC del messaggio m come $\phi = H^s(k||m)$, per una chiave k ed un'opportuna funzione hash. Abbiamo visto che se H è derivata dalla costruzione di Merkle-Damgård (cf. Crittosistema 4.1), questo MAC non è sicuro. Mostrare che, tuttavia, lo schema è sicuro nel modello dell'oracolo casuale.

Esercizio 7.7. Stabilire se le seguenti varianti di CBC-MAC sono sicure oppure no.

1. Lo schema CBC-MAC originale, utilizzato per autenticare messaggi a lunghezza variabile.
2. La variante in cui un vettore IV casuale è utilizzato ogni volta che un autenticatore è calcolato, e la stringa IV è inclusa nell'autenticatore.
3. La variante in cui tutti i blocchi intermedi sono inclusi nell'autenticatore.

Esercizio 7.8. Sia P una PRP forte. Consideriamo il seguente schema di cifratura: data una chiave $k \in \{0,1\}^n$ ed un messaggio $m \in \{0,1\}^{n/2}$ si estrae $\omega \xleftarrow{\$} \{0,1\}^{n/2}$ e si restituisce $c = P_k(m||\omega)$.

1. Mostrare che il cifrario è CCA-sicuro.
2. Considerare il MAC del Crittosistema 7.1, istanziato con P^{-1}. Dato $m \in \{0,1\}^n$ e la chiave k, l'autenticatore di m è $\phi = P_k^{-1}(m)$. Dire se il MAC è sicuro.
3. Supponiamo di combinare i due schemi secondo il paradigma "prima cifra e poi autentica", utilizzando *la stessa* chiave $k \in \{0,1\}^n$. Mostrare che lo schema risultante non è sicuro.

Esercizio 7.9. Mostrare che il paradigma "cifra ed autentica" del Paragrafo 7.4 non è sicuro in generale.
(Suggerimento: basta costruire un MAC sicuro che lascia trapelare informazione sul messaggio autenticato...)

Esercizio 7.10. Mostrare che il paradigma "prima autentica e poi cifra" del Paragrafo 7.4 non è sicuro in generale.

Esercizio 7.11. Proporre una definizione formale che catturi entrambi i requisiti di confidenzialità ed integrità. Mostrare che il paradigma "prima cifra e poi autentica" del Paragrafo 7.4 è sufficiente per ottenere tale nozione.

Letture consigliate

[ANS81] ANSI. *American national standard for financial institution message authentication (wholesale)*. ANSI X9.9. 1981.

[BCK96] Mihir Bellare, Ran Canetti e Hugo Krawczyk. "Keying Hash Functions for Message Authentication". In: *CRYPTO*. 1996, pp. 1–15.

[BGR95] Mihir Bellare, Roch Guérin e Phillip Rogaway. *XOR MACs: New Methods for Message Authentication Using Block Ciphers*. Rapp. tecn. RC 19970. IBM Research Report, mar. 1995.

[BKN04] Mihir Bellare, Tadayoshi Kohno e Chanathip Namprempre. "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm". In: *ACM Trans. Inf. Syst. Secur.* 7.2 (2004), pp. 206–241.

[BKN06] Mihir Bellare, Tadayoshi Kohno e Chanathip Namprempre. *The Secure Shell (SSH) Transport Layer Encryption Modes*. RFC 4344. `http://www.ietf.org/rfc/rfc4344.txt`. 2006.

[BKR00] Mihir Bellare, Joe Kilian e Phillip Rogaway. "The Security of the Cipher Block Chaining Message Authentication Code". In: *J. Comput. Syst. Sci.* 61.3 (2000), pp. 362–399.

[BN08] Mihir Bellare e Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *J. Cryptology* 21.4 (2008), pp. 469–491.

[BPR05] Mihir Bellare, Krzysztof Pietrzak e Phillip Rogaway. "Improved Security Analyses for CBC MACs". In: *Advances in Cryptology — CRYPTO '05*. Vol. 3621. Lecture Notes in Computer Science. Springer-Verlag, ago. 2005, pp. 527–545.

[Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1865.

[DDN00] Danny Dolev, Cynthia Dwork e Moni Naor. "Non-Malleable Cryptography". In: *SIAM J. Comput.* 30.2 (2000), pp. 391–437.

[FS03] Niels Ferguson e Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, 2003.

[GGM84] Oded Goldreich, Shafi Goldwasser e Silvio Micali. "On the Cryptographic Applications of Random Functions". In: *CRYPTO*. 1984, pp. 276–288.

[Gol01] Oded Goldreich. *Foundations of Cryptography, Vol. 2: Basic Applications*. Cambridge University Press, 2001.

[ISO89] ISO. *Data cryptographic techniques – data integrity mechanism using a cryptographic check function employing a block cipher algorithm*. ISO/IEC 9797. 1989.

[Kra01] Hugo Krawczyk. “The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)” In: *CRYPTO*. 2001, pp. 310–331.

[Mau02] Ueli M. Maurer. “Indistinguishability of Random Systems”. In: *EUROCRYPT*. 2002, pp. 110–132.

[NIS02] NIST. *The keyed-hash message authentication code (HMAC)*. Federal Information Processing Standard (FIPS). 2002.

[PR00] Erez Petrank e Charles Rackoff. “CBC MAC for Real-Time Data Sources”. In: *J. Cryptology* 13.3 (2000), pp. 315–338.

[PW10] Kenneth G. Paterson e Gaven J. Watson. “Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR”. In: *EUROCRYPT*. 2010, pp. 345–361.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

Capitolo 8

Tecniche asimmetriche di integrità

Furia disse a un topolino che trovò nel casottino: «rivolgiamoci alla legge ché ti voglio perseguire; suvvia forza non negare, qui bisogna processare; tanto più che stamattina non ho nulla di daffare.» Disse il topo alla bestiaccia: «Il processo, signor mio, senza giudice o giurato ci farebbe perder fiato! « Ti sarò giudice io, sarò io il tuo giurato,» disse Furia con scaltrezza. «La tua sorte in tribunale a me affida in sicurezza: e pertanto ti condanno alla pena capitale».

Lewis Carroll, Le avventure di Alice nel Paese delle Meraviglie [Car65]

Nel Capitolo 7 abbiamo studiato le tecniche simmetriche per soddisfare il requisito di *autenticazione di messaggio*, ovvero di integrità. In questo capitolo ci occupiamo delle *firme digitali*, che sono l'equivalente dei codici autenticatori di messaggio nel contesto della crittografia asimmetrica. La storia delle firme digitali ricalca quella della crittografia a chiave pubblica. L'idea è stata introdotta da Diffie ed Hellman, sempre in [DH76]. Tuttavia la prima realizzazione, peraltro insicura, è basata su RSA ed è datata 1978 [RSA78]. Goldwasser, Micali e Rivest [GMR88] sono stati i primi a formalizzare i requisiti di sicurezza per uno schema di firma digitale ed a presentare il primo schema dimostrabilmente sicuro.

Come abbiamo già discusso nel Capitolo 6, uno schema crittografico asimmetrico prevede che Alice ed il Bianconiglio posseggano due chiavi: una chiave pubblica ed una chiave segreta. Supponiamo che Alice voglia inviare al Bianconiglio il messaggio m attraverso un canale insicuro, preoccupandosi dell'integrità relativa al messaggio m trasmesso.

> *Esistono primitive che consentano di conservare l'integrità del messaggio m inviato (in chiaro) sul canale, senza condividere a priori un segreto?*

L'idea è che Alice produca una firma digitale da associare al messaggio m, in modo che il Bianconiglio possa verificarne l'integrità. Intuitivamente, la

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_8, © Springer-Verlag Italia 2012

firma deve essere generata attraverso la chiave segreta di Alice, in modo che nessun altro possa farlo al suo posto. D'altra parte, la verifica della firma che accompagna il messaggio m deve avvenire tramite la chiave pubblica di Alice, in modo che possa essere effettuata da tutti.

È bene osservare due differenze fondamentali tra il contesto a chiave simmetrica e quello a chiave asimmetrica:

- *Pubblica verificabilità.* A differenza di un MAC, una firma digitale è verificabile pubblicamente. Supponiamo di essere nel contesto a chiave segreta e che il Bianconiglio abbia ricevuto una coppia valida (m, ϕ) da Alice. Se ora il Bianconiglio volesse inoltrare l'autenticatore al Cappellaio Matto, quest'ultimo non potrà essere in grado di verificarlo, a meno che non conosca la chiave segreta k condivisa da Alice e dal Bianconiglio. Le firme digitali sono invece trasferibili: il Cappellaio Matto ha bisogno solo della chiave pubblica di Alice per poter verificare una sua firma.

- *Non ripudio.* Supponiamo che Alice abbia firmato un messaggio m, ma ad un certo punto neghi di averlo fatto. Alcune firme digitali consentono di verificare se una data firma è stata veramente prodotta da Alice, così che quest'ultima non possa negare di averla prodotta.

Guida per il lettore. La struttura del capitolo è la seguente. Nel Paragrafo 8.1 introdurremo formalmente il concetto di firma digitale e le relative nozioni di sicurezza. Vedremo quindi la soluzione naïve basata su RSA (insicura) ed una sua estensione nel Paragrafo 8.2. Studieremo le firme monouso e gli alberi di Merkle nel Paragrafo 8.3, e lo standard DSS nel Paragrafo 8.4. Infine, vedremo un esempio di firma innegabile nel Paragrafo 8.5.

8.1 Nozioni di sicurezza per firme digitali

La definizione formale di firma digitale e la relativa nozione di sicurezza sono una traduzione nel contesto asimmetrico delle Definizioni 7.1 e 7.2.

Definizione 8.1 (Firma digitale). Indichiamo con $\mathcal{M}$ lo spazio dei possibili testi in chiaro, con Σ lo spazio delle firme e con $\mathcal{K}$ lo spazio delle chiavi. Uno schema di firma digitale è una tripletta di algoritmi $\Pi_{\mathsf{SGN}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ definita come segue:

- `Generazioni delle chiavi.` L'algoritmo Gen è l'algoritmo di generazione della chiave segreta. Data la stringa 1^n (dove n quantificherà la sicurezza del cifrario), restituisce la coppia di chiavi (pk, sk) in $\mathcal{K}$.
- `Creazione delle firme.` L'algoritmo $\mathsf{Sign} : \mathcal{K} \times \mathcal{M} \to \Sigma$ è l'algoritmo di generazione delle firme digitali. Dato un messaggio $m \in \mathcal{M}$ da firmare e la chiave segreta sk, calcola la firma $\sigma = \mathsf{Sign}_{sk}(m)$.
- `Verifica.` L'algoritmo $\mathsf{Vrfy} : \mathcal{K} \times \mathcal{M} \times \Sigma \to \{0,1\}$ è l'algoritmo di verifica delle firme digitali. Data una coppia messaggio-firma (m, σ) e la chiave pubblica pk corrispondente ad sk, abbiamo:

$$\mathsf{Vrfy}_{pk}(m,\sigma) = \begin{cases} 1 & \text{se } \sigma = \mathsf{Sign}_{sk}(m) \\ 0 & \text{altrimenti.} \end{cases}$$

■

Quando $\mathsf{Vrfy}_{pk}(m,\sigma) = 1$, diremo che la firma σ è una firma valida del messaggio m. Quando l'algoritmo $\mathsf{Sign}(\cdot)$ è probabilistico, scriveremo $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$. Notare che se l'algoritmo $\mathsf{Sign}(\cdot)$ è deterministico, $\mathsf{Vrfy}(\cdot)$ calcola semplicemente $\sigma = \mathsf{Sign}_{sk}(m)$ e confronta il risultato con la firma ricevuta (pertanto non c'è bisogno di definire Vrfy esplicitamente). Come in ogni schema asimmetrico è cruciale accertare l'autenticità delle chiavi pubbliche. Tale problematica è indirizzata tramite il concetto d'infrastruttura a chiave pubblica (cf. Paragrafo 10.1).

Analogamente a quanto visto per i codici autenticatori di messaggio, la definizione di sicurezza per le firme digitali è quella di inforgiabilità universale contro attacchi a messaggio scelto: richiederemo che la Regina non sia in grado di forgiare la firma di un messaggio m^* a sua scelta (purché "fresco", cf. anche la Definizione 7.2). Formalmente, dato uno schema di firma $\Pi_{\mathsf{SGN}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\text{ufcma}}_{\mathsf{SGN},\Pi_{\mathsf{SGN}}}(\mathscr{F}, n)$:
1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$;
2. l'attaccante può richiedere $Q = \text{poly}(n)$ firme $\sigma \in \Sigma$ relative a messaggi $m \in \mathcal{M}$ a sua scelta;
3. quindi $\mathscr{F}$ sceglie una coppia $(m^*, \sigma^*) \in \mathcal{M} \times \Sigma$;
4. ritorna 1 se e solo se:
   (i) $\mathsf{Vrfy}_{sk}(m^*, \sigma^*) = 1$ e
   (ii) il messaggio m^* è diverso dai messaggi usati al passo (2).

Definizione 8.2 (Inforgiabilità universale contro attacchi a messaggio scelto). Diremo che lo schema di firma digitale Π_{SGN} è (t, Q, ϵ)-ufcma se, per ogni attaccante PPT $\mathscr{F}$ eseguibile in tempo t che effettua Q richieste nell'esperimento sopra definito, risulta:

$$\mathbb{P}\left[\mathbf{Exp}^{\text{ufcma}}_{\mathsf{SGN},\Pi_{\mathsf{SGN}}}(\mathscr{F}, n) = 1\right] \leq \epsilon.$$

■

8.2 Naïve RSA e hash a dominio pieno

Come primo esempio di schema di firma digitale useremo il crittosistema RSA (cf. Paragrafo 6.2) in modo diretto. Ricordiamo che RSA usa l'esponenziale modulare come permutazione unidirezionale con botola (cf. Definizione 6.3). Testi in chiaro e testi cifrati sono elementi di $\mathbb{Z}_N^*$. L'idea di base è quella di scambiare il ruolo degli elementi d ed e nel Crittosistema 6.1. Lo schema è mostrato nel Crittosistema 8.1. La correttezza è assicurata dal piccolo Teorema

Crittosistema 8.1. Schema di firma naïve basato su RSA

Sia $\mathcal{M} = \Sigma = \mathbb{Z}_N^*$ e $\mathcal{K}$ come nel Crittosistema 6.1.

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n, genera $(N, e, d) \leftarrow \mathsf{Gen}(1^n)$ dove $N = p \cdot q$ è prodotto di due primi di n bit, mentre d ed e sono interi tali che:

  $$d \cdot e \equiv 1 \pmod{\varphi(N)},$$

  essendo $\varphi(\cdot)$ la funzione toziente di Eulero. La chiave pubblica è $pk = (N, e)$, la chiave segreta $sk = (d, p, q)$.

- `Creazione della firma.` Dato un messaggio da autenticare $m \in \mathbb{Z}_N^*$ e la chiave segreta sk, calcola

  $$\sigma = \mathsf{Sign}_{sk}(m) = m^d \bmod N.$$

- `Verifica.` Data una coppia messaggio-firma $(m, \sigma) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$ e la chiave pubblica pk, l'algoritmo di verifica restituisce 1 se e solo se

  $$m = \sigma^e \bmod N.$$

di Fermat (cf. Teorema B.11) e dal fatto che $e \cdot d \equiv 1 \pmod{\varphi(N)}$, ovvero $e \cdot d = 1 + r \cdot \varphi(N)$ per qualche $r \in \mathbb{N}$. Possiamo infatti scrivere:

$$\sigma^e \equiv m^{e \cdot d} \equiv m^{1+r \cdot \varphi(N)} \equiv m \cdot m^{r \cdot \varphi(N)} \equiv m \pmod{N}.$$

Naïve RSA è insicuro. Mostreremo ora che l'uso naïve di RSA è totalmente insicuro. Il primo attacco che descriviamo è basato sulla sola conoscenza della chiave pubblica $pk = (N, e)$. L'attaccante sceglie $\sigma^* \xleftarrow{\$} \mathbb{Z}_N^*$ e calcola $m^* = (\sigma^*)^e \bmod N$: la coppia (m^*, σ^*) è ovviamente valida. Sebbene l'avversario non abbia alcun controllo[53] sul risultato (si parla infatti di *forgiatura esistenziale*), la Definizione 8.2 non è rispettata.

Possiamo produrre anche una forgiatura *selettiva* (ovvero una forgiatura in cui l'avversario ha pieno controllo sul messaggio m^* di cui forgia la firma). Supponiamo che $\mathscr{A}$ voglia produrre la firma relativa al messaggio $m \in \mathbb{Z}_N^*$. L'attaccante può scegliere $m_1 \xleftarrow{\$} \mathbb{Z}_N^*$, porre $m_2 = m/m_1 \bmod N$ e chiedere all'oracolo $\mathsf{Sign}_{sk}(\cdot)$ le firme σ_1, σ_2 relative ad m_1, m_2. Allora $\sigma = \sigma_1 \cdot \sigma_2$ è una firma valida per m. Infatti

$$\sigma^e \equiv (\sigma_1 \cdot \sigma_2)^e \equiv (m_1 \cdot m_2)^{ed} \equiv m^{ed} \equiv m \pmod{N}.$$

Hash a dominio pieno. Un modo per evitare questi attacchi è quello di usare una funzione hash (cf. Capitolo 4). Come vedremo la dimostrazione di sicurezza non è nel modello standard, ma nel modello dell'oracolo casuale (cf. Paragrafo 4.4). L'idea di base è quella di calcolare l'hash del messaggio *prima di produrre la firma*. Tra l'altro questo approccio ha il vantaggio che possiamo firmare messaggi a lunghezza arbitraria, in quanto l'hash comprime il messaggio iniziale in poche centinaia di bit (codificabili poi in $\mathbb{Z}_N^*$). Lo schema, detto hash a dominio pieno (*Full Domain Hash*, FDH), è presentato nel Crittosistema 8.2.

È lasciato come esercizio verificare che i due attacchi mostrati in precedenza per l'uso naïve di RSA, ora non sono più realizzabili (cf. Esercizio 8.3). Mostriamo invece che tale schema è sicuro quando $H(\cdot)$ è modellabile come un oracolo casuale.

Teorema 8.1 (FDH è ufcma nel modello dell'oracolo). *Se il problema RSA è (t, ϵ)-difficile, allora Π_{FDH} è $(t_{\mathsf{FDH}}, Q, \epsilon_{\mathsf{FDH}})$-ufcma nel modello dell'ora-*

[53] In realtà l'avversario ha un minimo controllo sul risultato. Ad esempio, se non sceglie σ^* completamente a caso può influenzare il messaggio m^* di cui forgerà la firma.

Crittosistema 8.2. Hash a dominio pieno

Sia $\mathcal{M} = \{0,1\}^*$, $\Sigma = \mathbb{Z}_N^*$ e $\mathcal{K}$ come nel Crittosistema 6.1. Sia inoltre $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}_H, H)$ una funzione hash, con $H : \{0,1\}^* \to \mathbb{Z}_N^*$. Consideriamo lo schema di firma $\Pi_{\mathsf{FDH}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ definito come segue:

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n, genera (N, e, d), come nel Crittosistema 8.1 ed estrai l'indice della funzione hash $s \leftarrow \mathsf{Gen}_H(1^n)$. La descrizione di $H(\cdot)$ è parte della chiave pubblica.
- `Creazione della firma.` Dato un messaggio da autenticare $m \in \mathbb{Z}_N^*$ e la chiave segreta sk, calcola:

$$\sigma = \mathsf{Sign}_{sk}(m) = (H^s(m))^d \bmod N.$$

- `Verifica.` Data la coppia (m, σ) e la chiave pubblica pk, l'algoritmo di verifica restituisce 1 se e solo se:

$$H^s(m) = \sigma^e \bmod N.$$

colo casuale, dove $Q = \mathrm{poly}(n)$ *e per:*

$$t_{\mathsf{FDH}} \approx t \qquad \epsilon_{\mathsf{FDH}} \leq e_{\mathrm{nep}} \cdot Q \cdot \epsilon.$$

(Qui $e_{\mathrm{nep}} \approx 2{,}7$ *è la costante di Neplero.)*

Dimostrazione. Dato un avversario PPT $\mathscr{F}$ eseguibile in tempo t_{FDH}, che effettua Q richieste ed è in grado di forgiare una firma per un messaggio m fresco con probabilità $\epsilon_{\mathsf{FDH}} > e_{\mathrm{nep}} \cdot Q \cdot \epsilon$ (come nell'enunciato del teorema), costruiamo un avversario $\mathscr{A}$ in grado di invertire l'esponenziale modulare (cf. Definizione 6.5) con vantaggio $> \epsilon$, contro l'ipotesi che il problema RSA sia (t, ϵ)-difficile. L'avversario $\mathscr{A}$, data un'istanza (N, e, y) deve calcolare x tale che $x^e \equiv y \pmod N$. Per fare ciò, usa $\mathscr{F}$ come segue:

1. Poni $pk = (N, e)$ e lancia $\mathscr{F}(1^n, pk)$.
2. Quando $\mathscr{F}$ interroga $H(\cdot)$ in corrispondenza del messaggio $m_i \in \mathbb{Z}_N^*$ rispondi come segue. Con probabilità α, estrai $\omega_i \xleftarrow{\$} \mathbb{Z}_N^*$ e restituisci $\omega_i^e \bmod N$ (diremo in questo caso che m_i è di tipo 1). Con probabilità

$1-\alpha$ estrai $\omega_i \xleftarrow{\$} \mathbb{Z}_N^*$ e restituisci $y \cdot \omega_i^e \bmod N$ (diremo in questo caso che m_i è di tipo 2).

3. Quando $\mathscr{F}$ richiede la firma relativa ad un messaggio $m_i \in \mathbb{Z}_N^*$, rispondi come segue: se m_i è di tipo 1 ritorna ω_i; altrimenti, se m_i è di tipo 2, ritorna $\perp$.

4. Quando $\mathscr{F}$ produce la forgiatura (m^*, σ^*), se m^* è di tipo 1 restituisci $\perp$. Altrimenti ritorna σ^*/ω^* (essendo ω^* il valore estratto al passo (2)).

Ovviamente $\mathscr{A}$ è eseguibile in tempo $t \approx t_{\mathsf{FDH}}$ (polinomiale). È semplice verificare che, quando $\mathscr{A}$ non ritorna $\perp$, la simulazione è perfetta, nel senso che la vista di $\mathscr{F}$ quando è lanciato da $\mathscr{A}$ è la stessa che nell'esperimento $\mathbf{Exp}^{\text{ufcma}}_{\mathsf{SGN},\Pi_{\mathsf{FDH}}}(\mathscr{F}, n)$. Infatti, quando $\mathscr{F}$ interroga l'oracolo casuale relativo ad $H(\cdot)$ riceve sempre come risposta un elemento casuale di $\mathbb{Z}_N^*$. (Questo è chiaro se m_i è di tipo 1. Se m_i è di tipo 2, basta osservare che poiché $\omega_i^e \bmod N$ è casuale, anche $y \cdot \omega_i^e \bmod N$ lo è.)

Osserviamo che quando $\mathscr{A}$ non ritorna $\perp$ e l'output (m^*, σ^*) di $\mathscr{F}$ è una forgiatura valida, $\mathscr{A}$ ha invertito l'esponenziale modulare. Infatti,

$$(\sigma^*)^e \equiv H^s(m^*) \equiv y \cdot (\omega^*)^e \pmod N \quad \Rightarrow \quad \left(\frac{\sigma^*}{\omega^*}\right)^e \equiv y \pmod N.$$

Resta quindi da calcolare la probabilità che $\mathscr{A}$ non ritorni $\perp$. Ogni richiesta di firma da parte di $\mathscr{F}$ è soddisfatta da $\mathscr{A}$ con probabilità α (perché m_i deve essere di tipo 1). D'altra parte quando $\mathscr{F}$ produce (m^*, σ^*), l'avversario $\mathscr{A}$ inverte l'esponenziale modulare con probabilità esattamente $1-\alpha$. Poiché $\mathscr{F}$ effettua Q richieste, concludiamo che la probabilità che $\mathscr{A}$ non ritorni $\perp$ è $\alpha^Q(1-\alpha)$. Calcoliamo la derivata, in modo da massimizzare la probabilità che $\mathscr{A}$ non ritorni $\perp$ rispetto ad α:

$$\begin{aligned} \frac{d}{d\alpha}\alpha^Q(1-\alpha) = Q\alpha^{Q-1} - (Q+1)\alpha^Q \stackrel{!}{=} 0 \quad &\Rightarrow \quad Q - (Q+1)\alpha = 0 \\ &\Rightarrow \quad \alpha = \frac{Q}{Q+1}. \end{aligned}$$

Quindi la probabilità massima per cui $\mathscr{A}$ non ritorna $\perp$ è:

$$\left(\frac{Q}{Q+1}\right)^Q \cdot \frac{1}{Q+1} = \frac{1}{Q}\left(1 - \frac{1}{Q+1}\right)^{Q+1} \approx \frac{1}{e_{\text{nep}} \cdot Q},$$

per Q sufficientemente grande (dove ora e_{nep} è la costante di Neplero).[54] Ma allora la probabilità che $\mathscr{A}$ inverta y è almeno $\epsilon_{\mathsf{FDH}}/(e_{\text{nep}} \cdot Q)$. Infine, sostituendo l'espressione per ϵ_{FDH}, abbiamo:

$$\mathbb{P}\left[\mathscr{A}(1^n, N, e, y) = x : \; x^e \equiv y \pmod{N}\right] \geq \frac{\epsilon_{\mathsf{FDH}}}{e_{\text{nep}} \cdot Q} > \epsilon,$$

contro l'ipotesi che l'ipotesi RSA sia (t, ϵ)-difficile. □

Riempimento. Nel Crittosistema 8.2 abbiamo supposto che esista una funzione $H : \{0,1\}^* \to \mathbb{Z}_N^*$. Bisogna essere più precisi su questo punto. Il modo in cui ciò è fatto in pratica è usando uno schema di riempimento. Tipicamente si parte con una funzione hash $H^s(\cdot)$ che mappa $\{0,1\}^*$ in $\{0,1\}^{\ell(n)}$ e si definisce poi una funzione $\mathsf{pad} : \{0,1\}^{\ell(n)} \to \{0,1\}^{|N|}$, in modo che il risultato sia codificabile con $|N|$ bit. Se ciò non è fatto in modo opportuno si possono introdurre falle nel sistema, come mostrato nell'esempio seguente.

Supponiamo di usare $e = 3$ e che il riempimento avvenga aggiungendo bit casuali (sulla destra) al valore dell'hash di m fino ad ottenere la lunghezza desiderata. Assumiamo che l'hash ritorni $\ell(n)$ bit e che $|N| = \ell'(n) > \ell(n)$. Scelto un messaggio m^* a caso, l'attaccante può calcolare $H^s(m^*)$, fare riempimento aggiungendo tanti 0 fino ad ottenere ℓ' bit e codificare il risultato come un elemento $u \in \mathbb{Z}_N^*$. Mostriamo che calcolando $\sigma^* = \lceil \sqrt[3]{u} \rfloor$ si ottiene la firma valida (m^*, σ^*). Notiamo che chi verifica la firma calcola $(\sigma^*)^3 \bmod N$ e si aspetta di vedere (in binario) $H^s(m^*)$ più un riempimento casuale sulla destra. Sia v l'intero ad ℓ bit corrispondente ad $H^s(m^*)$, possiamo scrivere

$$\begin{aligned} u = 2^{\ell'-\ell}v \quad &\Rightarrow \quad \sigma^* = \lceil \sqrt[3]{u} \rfloor = \left[2^{\ell'-\ell}v\right] + \eta \\ &\Rightarrow \quad (\sigma^*)^3 = 2^{\ell'-\ell}v + 3\eta\left[2^{\ell'-\ell}v\right]^{2/3} + 3\eta^2\left[2^{\ell'-\ell}v\right]^{1/3} + \eta^3, \end{aligned}$$

dove $0 \leq \eta < 1$ ed avendo indicato con $[\cdot]$ la funzione di approssimazione all'intero più vicino. Abbiamo così trovato $\sigma^* = 2^{\ell'-\ell}v + \xi$ dove ξ è un intero di al più $2 + 2\ell'/3$ bit. Basta dunque che $\ell + 2 + 2\ell'/3 \leq \ell'$, ovvero $\ell \leq \ell'/3 - 2$, affinché gli ℓ bit più significativi di $(\sigma^*)^3 \bmod N$ siano esattamente $H^s(m^*)$ e la firma forgiata sia dunque valida.

[54] Abbiamo usato il limite notevole per la costante di Neplero:

$$\lim_{x \to \pm\infty} \left(1 + \frac{1}{x}\right)^x = e_{\text{nep}}.$$

Paradigma "Hash & Firma". Le idee usate in FDH possono essere generalizzate nel paradigma "Hash & firma", l'equivalente asimmetrico del paradigma "Hash & MAC" del Crittosistema 7.4. Dato uno schema di firma digitale $\Pi_{\mathsf{SGN}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ per messaggi lunghi n bit ed una funzione hash $\Pi_{\mathsf{HASH}} = (\mathsf{Gen}_H, H)$ (tale che $H : \{0,1\}^* \to \{0,1\}^n$) è possibile combinare le due primitive per ottenere uno schema di firma digitale $\Pi^* = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$ per messaggi a lunghezza arbitraria. L'algoritmo di generazione delle chiavi estrae la coppia di chiavi $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ e l'indice $s \leftarrow \mathsf{Gen}_H(1^n)$. La descrizione della funzione hash è parte della chiave pubblica. Dato un messaggio $m \in \{0,1\}^*$ da firmare, l'algoritmo $\mathsf{Sign}^*(\cdot)$ calcola $\sigma \leftarrow \mathsf{Sign}_{sk}(H^s(m))$. L'algoritmo di verifica $\mathsf{Vrfy}^*(\cdot)$ ritorna 1 se e solo se $\mathsf{Vrfy}_{pk}(H(m), \sigma)$ restituisce 1.

In modo simile a quanto fatto nel Teorema 7.2, è possibile mostrare che tale combinazione è ufcma qualora Π_{SGN} sia ufcma e Π_{HASH} sia resistente alle collisioni. In particolare, la seconda ipotesi è fondamentale altrimenti:

- *Se* Π_{HASH} *non è resistente all'attacco della pre-immagine*, fissato un valore $y = H^s(m)$ l'attaccante può calcolare $m^* \neq m$ tale che $H^s(m^*) = y$. Sulla base della sola chiave pubblica pk può quindi trovare σ tale che $\mathsf{Vrfy}^*_{sk}(m, \sigma) = 1$ forgiando così la firma (m^*, σ).

- *Se* Π_{HASH} *non è resistente all'attacco della pre-immagine secondaria*, data la coppia valida (m, σ), l'avversario può trovare $m^* \neq m$ tale che $H^s(m^*) = H^s(m)$, e produrre la firma (valida) (m^*, σ).

- *Se* Π_{HASH} *non è resistente alle collisioni*, l'avversario può trovare m, m^* tali che $H^s(m) = H^s(m^*)$ ma $m \neq m^*$. Data la firma (m, σ), ha quindi forgiato (m^*, σ).

8.3 Firme monouso ed alberi di Merkle

Un modo naturale per rilassare la Definizione 8.2, è quello di richiedere che l'avversario possa richiedere la firma di un singolo messaggio prima di ritornare la forgiatura. Si parla di schemi di firma "monouso", introdotti per la prima volta da Lamport [Lam79]. Formalmente, possiamo considerare il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathsf{ufsma}}_{\mathsf{SGN},\Pi_{\mathsf{SGN}}}(\mathscr{F}, n)$:
1. $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$;
2. l'attaccante può richiedere la firma $\sigma \in \Sigma$ relativa ad

un singolo messaggio $m \in \mathcal{M}$ a sua scelta;
3. quindi $\mathscr{F}$ sceglie una coppia $(m^*, \sigma^*) \in \mathcal{M} \times \Sigma$;
4. ritorna 1 se e solo se:
   (i) $\mathsf{Vrfy}_{sk}(m^*, \sigma^*) = 1$ e
   (ii) il messaggio m^* è diverso dal messaggio m di cui al passo (2).

Definizione 8.3 (Inforgiabilità universale contro attacchi a messaggio singolo). Diremo che lo schema di firma digitale Π_{SGN} è (t, ϵ)-ufsma (universalmente inforgiabile contro attacchi a messaggio singolo, ovvero *universally unforgeable against single message attacks*) se, per ogni attaccante PPT $\mathscr{F}$ eseguibile in tempo t, risulta:

$$\mathsf{IP}\left[\mathbf{Exp}^{\mathsf{ufsma}}_{\mathsf{SGN},\Pi_{\mathsf{SGN}}}(\mathscr{F}, n) = 1\right] \leq \epsilon.$$

■

Lo schema di Lamport Π_{LMP}, rappresentato nel Crittosistema 8.3, è basato su una qualsiasi funzione unidirezionale (cf. Definizione 3.4). Notare che il sistema consente di firmare messaggi lunghi $\ell(n)$ bit.

Crittosistema 8.3. Firme monouso di Lamport

Sia $\mathcal{M} = \{0,1\}^{\ell}$ e $\Sigma = \{0,1\}^{\ell \cdot n}$. Sia inoltre $f : \{0,1\}^n \to \{0,1\}^n$ una funzione unidirezionale. Lo schema $\Pi_{\mathsf{LMP}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ è definito come segue.

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, genera $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ come descritto di seguito. Per ogni $i = 1, \ldots, \ell$ estrai $x_i^0, x_i^1 \xleftarrow{\$} \{0,1\}^n$. Calcola quindi $y_i^0 = f(x_i^0)$ ed $y_i^1 = f(x_i^1)$ e poni

$$pk = \begin{pmatrix} y_1^0 & y_2^0 & \ldots & y_\ell^0 \\ y_1^1 & y_2^1 & \ldots & y_\ell^1 \end{pmatrix} \quad sk = \begin{pmatrix} x_1^0 & x_2^0 & \ldots & x_\ell^0 \\ x_1^1 & x_2^1 & \ldots & x_\ell^1 \end{pmatrix}.$$

- `Creazione della firma.` Dato $m \in \{0,1\}^{\ell}$, con $m = (b_1, \ldots, b_\ell)$, e la chiave segreta sk produci la firma $\sigma = \mathsf{Sign}_{sk}(m) = (x_1^{b_1}, \ldots, x_\ell^{b_\ell})$.

- `Verifica.` Data la chiave pubblica pk e la firma (m, σ), con $\sigma = (\sigma_1, \ldots, \sigma_\ell)$, l'algoritmo $\mathsf{Vrfy}_{pk}(m, \sigma)$ ritorna 1 se e solo se

$$f(\sigma_i) = y_i^{b_i} \qquad \forall\ i = 1, 2, \ldots, \ell.$$

Teorema 8.2 (Π_{LMP} è ufsma). *Se $f(\cdot)$ è una funzione unidirezionale (t,ϵ)-sicura, lo schema di Lamport è $(t_{\mathsf{LMP}},\epsilon_{\mathsf{LMP}})$-ufsma per*

$$t_{\mathsf{LMP}} \approx t \qquad\qquad \epsilon_{\mathsf{LMP}} \leq 2\ell \cdot \epsilon.$$

Dimostrazione. Supponiamo che esista un avversario PPT $\mathscr{F}$ eseguibile in tempo t_{LMP} che ha vantaggio $\epsilon_{\mathsf{LMP}} > 2\ell \cdot \epsilon$ nell'esperimento $\mathbf{Exp}^{\mathsf{ufsma}}_{\mathsf{SGN},\Pi_{\mathsf{LMP}}}(\mathscr{F},n)$. Mostriamo come costruire un avversario $\mathscr{A}$ che, usando $\mathscr{F}$, inverte $f(\cdot)$ con vantaggio $> \epsilon$, contro l'ipotesi che quest'ultima sia una funzione unidirezionale (t,ϵ)-sicura. L'attaccante $\mathscr{A}$ riceve come input un valore $y \in \{0,1\}^n$ e deve trovare una pre-immagine $x \in \{0,1\}^n$ (dove $y = f(x)$). L'avversario simula quindi l'esperimento $\mathbf{Exp}^{\mathsf{ufsma}}_{\mathsf{SGN},\Pi_{\mathsf{LMP}}}(\mathscr{F},n)$ per $\mathscr{F}$ come segue:

1. Scegli un indice casuale $i^* \xleftarrow{\$} \{1,\ldots,\ell\}$ ed un bit $j^* \xleftarrow{\$} \{0,1\}$. Osserviamo che tali valori individuano una posizione precisa nella chiave pubblica del Crittosistema 8.3. Inserisci il valore y da invertire in posizione (i^*,j^*), ovvero poni $y^{j^*}_{i^*} = y$. Genera quindi la parte rimanente della chiave pubblica come previsto dallo schema di Lamport: per ogni $i = 1,\ldots,\ell$ e per $j = 0,1$ con $(i,j) \neq (i^*,j^*)$, estrai $x^j_i \xleftarrow{\$} \{0,1\}^n$ e calcola $y^j_i = f(x^j_i)$. Infine definisci
$$pk = \begin{pmatrix} y^0_1 & y^0_2 & \cdots & y^0_\ell \\ y^1_1 & y^1_2 & \cdots & y^1_\ell \end{pmatrix}.$$

2. Lancia $\mathscr{F}(1^n, pk)$. Quando $\mathscr{F}$ invia la sua richiesta $m = (b_1,\ldots,b_\ell)$ controlla se $b_{i^*} = j^*$. Se questo è il caso ritorna $\perp$, altrimenti calcola la firma σ relativa ad m come specificato dallo schema e restituisci il risultato ad $\mathscr{F}$.

3. Quando $\mathscr{F}$ ritorna la forgiatura (m^*,σ^*), dove $m^* = (b^*_1,\ldots,b^*_\ell)$, se $b^*_{i^*} \neq j^*$ ritorna $\perp$, altrimenti restituisci x_{i^*}.

Notare che $\mathscr{A}$ è eseguibile in tempo $t \approx t_{\mathsf{LMP}}$ (polinomiale). Osserviamo inoltre che la simulazione dell'esperimento è perfetta fintanto che $\mathscr{A}$ non ritorna $\perp$. In questo caso, allora, con probabilità ϵ_{LMP} la coppia (m^*,σ^*) è una forgiatura valida il che implica in particolare che $f(\sigma^*_{i^*}) = y^{b^*_{i^*}}_{i^*}$ ed $m^* \neq m$. Siccome $\mathscr{A}$ non ha restituito $\perp$, si deve avere anche $b^*_{i^*} = j^*$, così che $\mathscr{A}$ ha trovato una pre-immagine di y in quanto

$$f(\sigma^*_{i^*}) = y^{b^*_{i^*}}_{i^*} = y^{j^*}_{i^*} = y.$$

Resta da calcolare la probabilità con cui $\mathscr{A}$ riesce ad invertire y. Ciò accade solo quando si verifica quanto segue: (i) $\mathscr{A}$ è in grado di rispondere alla richiesta di $\mathscr{F}$ (vale a dire $b_{i^*} \neq j^*$), (ii) $\mathscr{F}$ ritorna una forgiatura valida e (iii) la forgiatura (m^*, σ^*) è tale che $b^*_{i^*} = j^*$. Siccome j^* è stato scelto a caso ed indipendentemente dalla vista di $\mathscr{F}$ la probabilità che (i) si verifichi è $1/2$. Supponendo che (i) si verifichi, la probabilità che (ii) si verifichi è ovviamente ϵ_{LMP}. Infine, supponendo che entrambe le condizioni (i) e (ii) si verifichino, abbiamo che $m^* \neq m$, pertanto esiste almeno un indice i per cui $m_i \neq m^*_i$. Basta allora che $i = i^*$, perché in questo caso (i) implica $b_{i^*} \neq j^*$ e quindi necessariamente $b^*_{i^*} = j^*$. Poiché m^* è lungo ℓ bit, ne segue che (iii) si verifica con probabilità almeno $1/\ell$.

Mettendo tutto insieme possiamo concludere che $\mathscr{A}$ inverte y con probabilità almeno $\epsilon_{\mathsf{LMP}}/(2\ell)$. Sostituendo l'espressione per ϵ_{LMP}, abbiamo:

$$\mathbb{P}\left[\mathscr{A}(1^n, y) = x : \ y = f(x), x \xleftarrow{\$} \{0,1\}^n\right] \geq \frac{\epsilon_{\mathsf{LMP}}}{2\ell} > \epsilon,$$

contro l'ipotesi che $f(\cdot)$ sia (t, ϵ)-sicura. □

Firme ad albero. Lo schema di Lamport è utile a firmare un solo messaggio (quando ciò non avviene esistono attacchi in grado di forgiare firme di messaggi a piacere, cf. Esercizio 8.4). Inoltre si possono firmare solamente messaggi lunghi $\ell(n)$ bit. Quest'ultimo problema può essere risolto usando una funzione hash resistente alle collisioni ed applicando il paradigma "Hash & Firma" del paragrafo precedente. Siccome è possibile dimostrare che l'esistenza delle funzioni hash resistenti alle collisioni implica l'esistenza delle funzioni unidirezionali, otteniamo come corollario che l'esistenza delle funzioni hash resistenti alle collisioni implica l'esistenza di schemi di firma digitale "monouso" per messaggi a lunghezza arbitraria.

Per risolvere il primo problema, si possono usare *alberi binari* come mostrato per la prima volta da Merkle [Mer87; Mer89]. Ci occorre il concetto di *primitiva con stato*. Lo *stato* associato ad una primitiva Π è un valore che la primitiva utilizza ed aggiorna durante il suo funzionamento. Sia $\mathcal{S}$ lo spazio degli stati; in fase di inizializzazione (ad esempio quando sono generate le chiavi) si genera uno stato iniziale $s_0 \in \mathcal{S}$. La primitiva quindi lavora in cicli: nel ciclo i lo stato attuale $s_{i-1} \in \mathcal{S}$, un valore di input $x_i \in \mathcal{X}$ ed un valore casuale $\omega_i \in \Omega$ (se Π è randomizzato), sono utilizzati per generare l'output $y_i \in \mathcal{Y}$ ed il nuovo stato $s_i \in \mathcal{S}$. Ciò è indicato con

$$(s_i, y_i) \overset{\Pi(s_{i-1})}{\hookleftarrow} (x_i, s_{i-1}, \omega_i).$$

Crittosistema 8.4. Alberi di Merkle

Sia $\mathcal{M} = \{0,1\}^n$ e consideriamo uno schema di firma "monouso" $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. Data una stringa binaria $\mathbf{m} \in \mathcal{M}$ indichiamo con $\mathbf{m}_i = (\mathbf{m}[1], \ldots, \mathbf{m}[i])$ il prefisso fino al bit i-simo di $\mathbf{m}$ (per convenzione poniamo $\varepsilon = \mathbf{m}_0$). Lo schema $\Pi_{\mathsf{Merkle}} = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo $\mathsf{Gen}^*(1^n)$ estrae $(pk_\varepsilon, sk_\varepsilon) \leftarrow \mathsf{Gen}(1^n)$ e rende pubblica pk_ε. La chiave segreta e lo stato iniziale sono rappresentati da sk_ε.
- `Creazione della firma.` Dato il messaggio $\mathbf{m} \in \{0,1\}^n$ genera la relativa firma come segue:
  (i) per ogni $i = 0, \ldots, n-1$, se $pk_{\mathbf{m}_i||0}$, $pk_{\mathbf{m}_i||1}$ e $\sigma_{\mathbf{m}_i}$ non sono nello stato, calcola $(pk_{\mathbf{m}_i||0}, sk_{\mathbf{m}_i||0}) \leftarrow \mathsf{Gen}(1^n)$, $(pk_{\mathbf{m}_i||1}, sk_{\mathbf{m}_i||1}) \leftarrow \mathsf{Gen}(1^n)$ e $\sigma_{\mathbf{m}_i} \leftarrow \mathsf{Sign}_{sk_{\mathbf{m}_i}}(pk_{\mathbf{m}_i||0} \,||\, pk_{\mathbf{m}_i||1})$ ed aggiungi tali valori allo stato;
  (ii) se $\sigma_{\mathbf{m}}$ non fa parte dello stato, calcola $\sigma_{\mathbf{m}} \leftarrow \mathsf{Sign}_{sk_{\mathbf{m}}}(\mathbf{m})$ ed aggiungi tale valore allo stato;
  (iii) la firma di $\mathbf{m}$ è
  $$\sigma = \left(\left\{ \sigma_{\mathbf{m}_i}, pk_{\mathbf{m}_i||0}, pk_{\mathbf{m}_i||1} \right\}_{i=0}^{n-1}, \sigma_{\mathbf{m}} \right).$$
- `Verifica.` Data la chiave pubblica pk_ε, e la firma $(\mathbf{m}, \sigma)$ accetta la firma come valida se e solo se:
  (i) $\mathsf{Vrfy}_{pk_{\mathbf{m}_i}}(pk_{\mathbf{m}_i||0} \,||\, pk_{\mathbf{m}_i||1}, \sigma_{\mathbf{m}_i}) = 1$ per ogni $i = 0, \ldots, n-1$;
  (ii) $\mathsf{Vrfy}_{pk_{\mathbf{m}}}(\mathbf{m}, \sigma_{\mathbf{m}}) = 1$.

Lo schema di Merkle è mostrato nel Crittosistema 8.4. Se il messaggio da firmare è lungo n bit, avremo a che fare con un albero binario di profondità n, avente 2^n foglie. Informalmente, la firma del messaggio corrisponde ad un sentiero "certificato" (da una foglia alla radice) nell'albero. Siccome l'albero ha dimensione esponenziale esso non verrà generato completamente, ma i nodi verranno aggiunti "al volo" quando necessario.

Più nel dettaglio, immaginiamo un albero binario di profondità n, etichet-

tiamo la radice con ε (la stringa vuota) e ciascun nodo con la stringa binaria $\mathbf{w}$ (di lunghezza inferiore ad n bit). Ogni nodo $\mathbf{w}$ avrà figlio destro $\mathbf{w}||0$ e figlio sinistro $\mathbf{w}||1$. Dato uno schema di firma "monouso" Π, si associa a ciascun nodo $\mathbf{w}$ dell'albero una coppia di chiavi $(pk_{\mathbf{w}}, sk_{\mathbf{w}})$ generate attraverso Π. La chiave pubblica pk_{ε} della radice sarà la chiave pubblica di chi produce la firma. Per firmare un messaggio $\mathbf{m} \in \{0,1\}^n$ si procede dunque come segue:

1. Se non già fatto in precedenza (per generare la firma di altri messaggi), si generano le chiavi per ogni nodo nel sentiero dalla radice al nodo con etichetta $\mathbf{m}$.

2. Si certifica il sentiero dalla radice al nodo con etichetta $\mathbf{m}$ calcolando la firma relativa a $pk_{\mathbf{w}||0} \,||\, pk_{\mathbf{w}||1}$, usando la chiave $sk_{\mathbf{w}}$, per *ogni* stringa $\mathbf{w}$ che è prefisso di $\mathbf{m}$.

3. Infine si certifica $\mathbf{m}$ calcolandone la firma con chiave $sk_{\mathbf{m}}$.

La firma di $\mathbf{m}$ è formata dalla firma di $\mathbf{m}$ stesso e da tutte le firme intermedie necessarie a certificare il sentiero verso la radice.

Osserviamo che ciascuna coppia di chiavi associata ad un nodo nello schema è usata per firmare *un singolo "messaggio"*. Inoltre, siccome ciascuna firma è relativa ad una coppia di chiavi, abbiamo bisogno di uno schema di firma monouso in grado di firmare messaggi più lunghi della chiave pubblica. (Questo è ottenibile usando ad esempio il paradigma "Hash & Firma".) Notare che lo schema di Merkle permette di firmare un numero illimitato di messaggi.[55] Inoltre la lunghezza della firma e l'efficienza del processo di verifica sono proporzionali alla lunghezza del messaggio $\mathbf{m}$, ma *indipendenti* dal numero di messaggi firmati. Per quanto riguarda la sicurezza abbiamo il seguente:

Teorema 8.3 (Π_{Merkle} è ufcma). *Se Π è uno schema di firma "monouso" (t,ϵ)-ufsma allora Π_{Merkle} è $(t_{\mathsf{Merkle}}, Q_{\mathsf{Merkle}}, \epsilon_{\mathsf{Merkle}})$-ufcma, per ogni $Q_{\mathsf{Merkle}} = \mathrm{poly}(n)$ e con:*

$$t_{\mathsf{Merkle}} \approx t \qquad \epsilon_{\mathsf{Merkle}} \leq (2n \cdot Q_{\mathsf{Merkle}} + 1)\epsilon.$$

Dimostrazione. Sia $\mathscr{F}_{\mathsf{Merkle}}$ un attaccante PPT eseguibile in tempo t_{Merkle} in grado di forgiare una firma valida per Π_{Merkle} effettuando Q_{Merkle} richieste e con vantaggio $\epsilon_{\mathsf{Merkle}} > (2nQ_{\mathsf{Merkle}} + 1)\epsilon$ (come nell'enunciato del teorema). Posto $Q(n) = 2n \cdot Q_{\mathsf{Merkle}} + 1$, osserviamo che $Q(n)$ è un limite superiore per il

[55] In realtà "solo" 2^n; comunque tale valore è più grande di una qualsiasi funzione polinomiale di n.

numero di chiavi pubbliche in Π necessarie a firmare Q_{Merkle} messaggi usando Π_{Merkle}. (Ciò si verifica perché ogni firma nello schema di Merkle richiede di generare al più $2n$ nuove chiavi per Π più la chiave pk_{ε}.)

Mostriamo come usare $\mathscr{F}_{\mathsf{Merkle}}$ per costruire un attaccante PPT $\mathscr{F}$ in grado di forgiare una firma valida per Π con vantaggio $> \epsilon$, contro l'ipotesi che Π sia (t, ϵ)-ufsma. L'attaccante $\mathscr{F}$ simula la vista di $\mathscr{F}_{\mathsf{Merkle}}$ nell'esperimento $\mathbf{Exp}^{\text{ufcma}}_{\mathsf{SGN},\Pi_{\mathsf{Merkle}}}(\mathscr{F}_{\mathsf{Merkle}}, n)$ come segue:

1. Scegli un indice casuale $i^* \xleftarrow{\$} \{1, \ldots, Q\}$ e costruisci la lista di chiavi pubbliche $pk_1, \ldots, pk_Q$ dove $pk_{i^*} = pk$ (la chiave pubblica relativa allo schema Π) e $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^n)$ per ogni $i \neq i^*$.

2. Lancia $\mathscr{F}_{\mathsf{Merkle}}(1^n, pk_{\varepsilon})$, avendo posto $pk_{\varepsilon} = pk_1$. Quindi, per ogni $i = 0, \ldots, n-1$:

   - Se gli elementi $pk_{\mathbf{m}_i||0}, pk_{\mathbf{m}_i||1}$ e $\sigma_{\mathbf{m}_i}$ non sono ancora stati definiti, definisci $pk_{\mathbf{m}_i||0}$ e $pk_{\mathbf{m}_i||1}$ pari alle prime due chiavi pubbliche pk_j e pk_{j+1} non ancora utilizzate. Calcola quindi la firma della stringa $pk_{\mathbf{m}_i||0}||pk_{\mathbf{m}_i||1}$ rispetto alla chiave pubblica $pk_{\mathbf{m}_i}$. (Notare che se $i \neq i^*$ l'avversario può calcolare la firma da solo, in quanto conosce la corrispondente chiave segreta. Quando $i = i^*$ invece deve inoltrare il messaggio al suo oracolo per ottenere la firma.)
   - Se $\sigma_{\mathbf{m}}$ non è ancora definito calcola la firma $\sigma_{\mathbf{m}}$ di $\mathbf{m}$ rispetto la chiave $pk_{\mathbf{m}}$.
   - Restituisci ad $\mathscr{F}_{\mathsf{Merkle}}$ la firma:

$$\sigma = \left(\left\{ \sigma_{\mathbf{m}_i}, pk_{\mathbf{m}_i||0}, pk_{\mathbf{m}_i||1} \right\}_{i=0}^{n-1}, \sigma_{\mathbf{m}} \right).$$

3. Sia $(\mathbf{m}^*, \sigma^*)$ la forgiatura ritornata da $\mathscr{F}_{\mathsf{Merkle}}$ (per un qualche messaggio $\mathbf{m}^*$ non richiesto in precedenza all'oracolo), dove:

$$\sigma^* = \left(\left\{ \sigma^*_{\mathbf{m}^*_i}, pk^*_{\mathbf{m}^*_i||0}, pk^*_{\mathbf{m}^*_i||1} \right\}_{i=0}^{n-1}, \sigma^*_{\mathbf{m}^*} \right).$$

   Distinguiamo due casi:

   Caso 1. Esiste un indice $j \in \{0, \ldots, n-1\}$ tale che $pk^*_{\mathbf{m}^*_j||0} \neq pk_{\mathbf{m}^*_j||0}$ oppure $pk^*_{\mathbf{m}^*_j||1} \neq pk_{\mathbf{m}^*_j||1}$. Sia j il minimo indice con questa proprietà, definiamo i come l'indice per cui $pk_i = pk_{\mathbf{m}^*_j} = pk^*_{\mathbf{m}^*_j}$ (tale i esiste sempre per la minimalità di j). Se $i = i^*$, ritorna $(pk^*_{\mathbf{m}^*_j||0}||pk^*_{\mathbf{m}^*_j||1}, \sigma^*_{\mathbf{m}^*_j})$.

Caso 2. Se il caso 1 non si verifica, abbiamo $pk^*_{\mathbf{m}^*} = pk_{\mathbf{m}^*}$. Sia i l'indice per cui $pk_i = pk_{\mathbf{m}^*}$. Se $i = i^*$, ritorna $(m^*, \sigma^*_{\mathbf{m}^*})$.

Ovviamente $\mathscr{F}$ è eseguibile in tempo $t \approx t_{\mathsf{Merkle}}$ (polinomiale). Inoltre è facile verificare che la simulazione dell'esperimento da parte di $\mathscr{F}$ è perfetta, quindi $\mathscr{F}_{\mathsf{Merkle}}$ ritorna una forgiatura valida con probabilità almeno $\epsilon_{\mathsf{Merkle}}$. Supponiamo che questo sia il caso, ed analizziamo i due casi definiti sopra distintamente:

Caso 1. Siccome i^* è stato scelto a caso ed indipendentemente dalla vista di $\mathscr{F}_{\mathsf{Merkle}}$, la probabilità che $i = i^*$ è $1/Q$. Quando $i = i^*$ l'avversario $\mathscr{F}$ ha richiesto al suo oracolo la firma del messaggio $pk_{\mathbf{m}_j||0}||pk_{\mathbf{m}_j||1}$ rispetto alla chiave pubblica $pk = pk_{i^*} = pk_{\mathbf{m}_j}$. D'altra parte:

$$pk^*_{\mathbf{m}^*_j||0} \,||\, pk^*_{\mathbf{m}^*_j||1} \neq pk_{\mathbf{m}^*_j||0} \,||\, pk_{\mathbf{m}^*_j||1},$$

e $\sigma^*_{\mathbf{m}^*_j}$ è una firma valida per $pk^*_{\mathbf{m}^*_j||0} \,||\, pk^*_{\mathbf{m}^*_j||1}$.

Caso 2. Anche in questo caso, siccome i^* è scelto a caso ed indipendentemente dalla vista di $\mathscr{F}_{\mathsf{Merkle}}$, la probabilità che $i = i^*$ è $1/Q$. Se $i = i^*$ l'avversario $\mathscr{F}$ non ha richiesto alcuna firma relativa alla chiave pubblica pk, comunque $\sigma^*_{\mathbf{m}}$ è una firma valida per $\mathbf{m}$ rispetto a pk.

Pertanto, indipendentemente da quale dei due casi si verifica, quando $\mathscr{F}_{\mathsf{Merkle}}$ ritorna una forgiatura valida, $\mathscr{F}$ ritorna una forgiatura valida con probabilità $1/Q$. Sostituendo l'espressione per $\epsilon_{\mathsf{Merkle}}$ abbiamo:

$$\mathbb{P}\left[\mathbf{Exp}^{\mathsf{ufsma}}_{\mathsf{SGN},\Pi}(\mathscr{F}, n) = 1\right] \geq \frac{\epsilon_{\mathsf{Merkle}}}{Q} > \epsilon,$$

contro l'ipotesi che Π sia (t, ϵ)-ufsma. □

Rimuovere lo stato. Lo stato in Π_{Merkle} dipende, di fatto, dal messaggio di cui si vuole generare la firma. Se si generassero tutte le possibili chiavi $\{(pk_{\mathbf{w}}, sk_{\mathbf{w}})\}$ e le relative firme $\{\sigma_{\mathbf{w}}\}$ per ogni possibile stringa binaria w di lunghezza al più n bit, infatti, non sarebbe necessario mantenere alcuno stato. Lo svantaggio di questa soluzione è che generare tutti questi valori richiede ovviamente un tempo *esponenziale* ed è quindi non efficiente. Si potrebbe pensare di memorizzare valori casuali da utilizzare per generare i valori $\{(pk_{\mathbf{w}}, sk_{\mathbf{w}})\}$ e le relative firme $\{\sigma_{\mathbf{w}}\}$ quando necessario. Ad esempio

Alice può memorizzare un valore $\omega_{\mathbf{w}}$ per ciascun nodo $\mathbf{w}$ e quando necessario calcolare $(pk_{\mathbf{w}}, sk_{\mathbf{w}}) = \mathsf{Gen}(1^n, \omega_{\mathbf{w}})$ (cioè la chiave è generata usando il valore casuale $\omega_{\mathbf{w}}$). Allo stesso modo, si può memorizzare un valore $\omega'_{\mathbf{w}}$ e porre $\sigma_{\mathbf{w}} = \mathsf{Sign}_{sk_{\mathbf{w}}}(pk_{\mathbf{w}||0}||pk_{\mathbf{w}||1}, \omega'_{\mathbf{w}})$. Questo richiede ancora complessità esponenziale.

La soluzione al problema è usare un misto delle due strategie. Invece di memorizzare le stringhe $\omega_{\mathbf{w}}, \omega'_{\mathbf{w}}$, Alice può memorizzare due chiavi k, k' di una (opportuna) funzione pseudocasuale $F(\cdot)$. Quando è necessario si estrae prima il valore $\omega_{\mathbf{w}} = F_k(\mathbf{w})$ e poi si calcola $(pk_{\mathbf{w}}, sk_{\mathbf{w}}) = \mathsf{Gen}(1^n, \omega_{\mathbf{w}})$ come in precedenza. Allo stesso modo si estrae $\omega'_{\mathbf{w}} = F_{k'}(\mathbf{w})$ e si calcola $\sigma_{\mathbf{w}} = \mathsf{Sign}_{sk_{\mathbf{w}}}(pk_{\mathbf{w}||0}||pk_{\mathbf{w}||1}, \omega'_{\mathbf{w}})$. Abbiamo così eliminato lo stato ed ottenuto una procedura per generare le chiavi in tempo polinomiale (non è complesso mostrare che tale schema rimane sicuro).

8.4 Lo standard DSS

Lo standard per le firme digitali (*Digital Signature Standard*, DSS) è stato sviluppato dal governo federale degli Stati Uniti. La prima versione risale al 1991 [NIS91]. Lo schema di firma è basato sul problema del logaritmo discreto e presenta diversi punti in comune con il crittosistema di ElGamal (cf. Paragrafo 6.3). Lo schema Π_{DSS} è mostrato nel Crittosistema 8.5. L'elemento g può essere generato come segue. Sia g' un elemento di $\mathbb{Z}_p^*$, si calcola $g = (g')^{(p-1)/q} \bmod p$, in modo che $g^q \equiv 1 \pmod{p}$ (eventualmente scartare il valore $g = 1$).

Osserviamo che una firma valida è accettata con probabilità 1 in quanto

$$u + av \equiv s^{-1} \cdot (H(m) + r \cdot a) \equiv s^{-1} bs \equiv b \pmod{q},$$

e quindi (lavorando modulo q all'esponente), abbiamo esattamente:

$$g^u \alpha^v \bmod p \equiv g^{u+av} \bmod p \equiv g^b \bmod p = r \pmod{q}.$$

Sebbene esistano alcune analisi formali di sicurezza [Nac+94; Vau03] per alcune varianti dello schema, ci limitiamo qui ad un'analisi euristica.

Dimensione dei primi p e q ed efficienza. Intuitivamente la sicurezza è connessa a due istanze distinte del problema del logaritmo discreto. Una ha luogo in $\mathbb{Z}_p^*$ dove è disponibile il metodo del calcolo dell'indice (cf. Appendice C.3): pertanto il NIST suggerisce di usare almeno $|p| = 1024$ bit. La seconda

Crittosistema 8.5. Lo standard DSS

Sia $\mathcal{M} = \{0,1\}^*$ e $\Sigma = \mathbb{Z}_q^* \times \mathbb{Z}_q^*$. Lo schema $\Pi_{\mathsf{DSS}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ è definito come segue:

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n genera $(p,q,g) \leftarrow \mathsf{Gen}(1^n)$, definiti di seguito. L'elemento g è un generatore del sottogruppo moltiplicativo di $\mathbb{Z}_p^*$ di ordine q, con p,q primi tali che $q \mid (p-1)$. Sia inoltre $H : \{0,1\}^* \to \mathbb{Z}_q$ una funzione hash. Ciascun utente genera la sua coppia di chiavi pubblica/privata come segue. Sceglie $a \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\alpha = g^a \bmod p$. La chiave segreta è $sk = a$, la chiave pubblica $pk = (H, p, q, g, \alpha)$.
- `Creazione della firma.` Dato un messaggio $m \in \{0,1\}^*$ e la chiave segreta sk, estrai $b \xleftarrow{\$} \mathbb{Z}_q$ e calcola:

$$r = (g^b \bmod p) \bmod q \qquad s = (H(m) + a \cdot r) \cdot b^{-1} \bmod q.$$

  La firma di m è $\sigma = (r,s)$.
- `Verifica.` Data la chiave pubblica pk e la firma $\sigma = (r,s)$ siano:

$$u = H(m) \cdot s^{-1} \bmod q \qquad v = r \cdot s^{-1} \bmod q.$$

  L'algoritmo $\mathsf{Vrfy}_{pk}(m,\sigma)$ restituisce 1 se e solo se:

$$r = (g^u \alpha^v \bmod p) \bmod q.$$

istanza ha luogo nel sottogruppo moltiplicativo di $\mathbb{Z}_p^*$ con ordine q. Qui gli algoritmi per il logaritmo discreto sono molto meno efficienti, per cui è sufficiente $|q| = 160$ bit.

Notare che la dimensione delle firme è di circa 320 bit, contro i 1024 di RSA. Inoltre sono necessarie solo due operazioni modulo p: una per il calcolo di r (che può essere pre-calcolato, in quanto non dipende da a) nella generazione della firma, e l'altra per il calcolo di v nel processo di verifica.

Evitare $r = 0$ ed $s \notin \mathbb{Z}_q^*$. Quando $r = 0$, abbiamo che $s = H(m) \cdot b^{-1} \bmod q$ è indipendente dalla chiave segreta $sk = a$ (il che ovviamente è indesiderato). Inoltre se s non è invertibile in $\mathbb{Z}_q$, non si può verificare la firma. Tali eventi

sono comunque poco probabili data la dimensione di q (probabilità $\approx 2^{-160}$, quando $|q| = 160$ bit).

Verifica dei parametri globali di sistema. Ogni utente deve verificare che i parametri globali di sistema siano generati correttamente. Vale a dire, p deve essere un primo di dimensione appropriata, q deve essere un fattore primo di $p-1$ della dimensione appropriata, $a \in \mathbb{Z}_p^*$ deve avere ordine q. Se ciò non è verificato potrebbero esistere attacchi particolari per violare la sicurezza dello schema.

Distruzione e freschezza di b. Il valore b deve essere usato e poi distrutto, infatti data la firma (m, σ) ed il valore b si può calcolare la chiave segreta

$$a \equiv (bs - H(m))r^{-1} \pmod{q}.$$

Inoltre il valore b non deve essere usato per firmare due messaggi diversi. Siano (m_1, σ_1) ed (m_2, σ_2) due firme generate con lo stesso valore di b. Osserviamo che quando il valore di b non cambia, anche il valore di r resta immutato nelle due firme. Quindi

$$\begin{aligned} s_1 &= (H(m_1) + a \cdot r) \cdot b^{-1} \bmod q \\ s_2 &= (H(m_2) + a \cdot r) \cdot b^{-1} \bmod q, \end{aligned}$$

e prendendo la differenza possiamo calcolare

$$b = (H(m_1) - H(m_2))(s_1 - s_2)^{-1} \bmod q.$$

Noto b si può quindi recuperare sk come già osservato.

Resistenza alle collisioni di $H(\cdot)$. Una collisione in $H(\cdot)$ consente immediatamente di forgiare una firma, quindi la funzione hash deve essere resistente alle collisioni.

DSS su Curve Ellittiche. Come abbiamo visto anche per il crittosistema di ElGamal, possiamo definire Π_{DSS} sul gruppo definito dai punti di una curva ellittica su un campo finito (cf. Appendice B.4). Consideriamo la curva $E : Y^2 \equiv X^3 + AX + B \pmod{p}$, dove p è un primo (oppure una potenza di 2) ed $A, B \in \mathbb{Z}_p$. Si può dimostrare che l'insieme:

$$E(\mathbb{Z}_p) = \left\{(x, y) \in \mathbb{Z}_p : \ y^2 \equiv x^3 + Ax + B\right\} \cup \infty,$$

è un gruppo con un'opportuna operazione di somma. Il punto ∞ è detto punto all'infinito della curva. Sia $\mathbb{H} \subset E(\mathbb{Z}_p)$ un sottogruppo ciclico di ordine q, con q primo. Per una scelta opportuna di p, q, il problema del logaritmo discreto è ritenuto difficile in $\mathbb{H}$. Poniamo $\mathcal{M} = \{0,1\}^*$. Ciascuna firma sarà un elemento di $\Sigma = \mathbb{Z}_q^* \times \mathbb{Z}_q^*$; l'insieme delle chiavi è dato da:

$$\mathcal{K} = \{E, p, q, P, R, x : \ R = [a]P\},$$

dove E, P, R, p, q sono i parametri pubblici ed $a \in \mathbb{Z}_q^*$ è la chiave segreta.

Per firmare il messaggio $m \in \{0,1\}^*$, si procede come segue. Si sceglie a caso $b \in \mathbb{Z}_q^*$ e si calcola il punto $[b]P$, con ascissa $\Upsilon_{[b]P} \in \mathbb{Z}_p$. La firma di m è del tipo $\sigma = (r, s)$, con:

$$r = \Upsilon_{[b]P} \bmod q \qquad s = (H(m) + a \cdot r)b^{-1} \bmod q,$$

dove $H : \{0,1\}^* \to \mathbb{Z}_q$ è una funzione hash resistente alle collisioni. La verifica della firma (m, σ) avviene calcolando:

$$u = H(m) \cdot s^{-1} \bmod q \qquad v = r \cdot s^{-1} \bmod q.$$

e recuperando l'ascissa $\Upsilon_{[b]P}$ del punto $[u]P + [v]R = [b]P$. L'algoritmo di verifica $\mathsf{Vrfy}_{pk}(m, \sigma)$ ritorna 1 se e solo se:

$$r = \Upsilon_{[b]P} \bmod q.$$

Notare che quando la firma è generata in modo corretto

$$u + av \equiv s^{-1} \cdot (H(m) + r \cdot a) \equiv s^{-1}bs \equiv b \pmod{q},$$

e quindi:

$$[u]P + [v]R = [u]P + [av]R = [u + av]P = [b]P,$$

e l'algoritmo di verifica ritorna sempre 1. Tutte le operazioni sono effettuate modulo q, dove tipicamente $|q| = 160$ bit. Sono necessarie solo due operazioni in $\mathbb{Z}_p$: per il calcolo di r nella generazione della firma (che può essere precalcolato, in quanto non dipende da a) e per il calcolo di v nel processo di verifica. Esistono curve in cui il problema del logaritmo discreto è difficile già per valori di $|p| = 160$ bit, portando quindi un considerevole vantaggio in termini di efficienza. Le precauzioni sono le stesse relative al DSS (validazione dei parametri globali, freschezza e distruzione di b, evitare $r = 0$ ed s non invertibile).

8.5 Firme innegabili

Tutti gli schemi di firma digitale che abbiamo incontrato finora sono composti da due fasi: la generazione della firma e la verifica. Si potrebbe pensare di rendere il processo di verifica *interattivo*: quando si verifica una firma si coinvolge il mittente nel processo di verifica. In questo modo i messaggi firmati non possono essere diffusi senza il consenso di chi li ha prodotti. Tipicamente questo è realizzato attraverso un paradigma a *sfida e risposta* (vedremo diversi esempi nel Capitolo 11). Il problema naturale che sorge in questo contesto, è che Alice potrebbe barare nel processo di verifica e negare di aver prodotto la firma di un certo messaggio. Le firme *innegabili* evitano che ciò possa accadere attraverso una terza procedura, detta *protocollo di ripudio*. Attraverso uno scambio di messaggi il Bianconiglio può verificare se la firma è stata effettivamente prodotta da Alice (senza che quest'ultima possa negare di averlo fatto), ed allo stesso tempo Alice può dimostrare che una data firma non è stata generata da lei, ma forgiata da un'attaccante.[56]

Non introdurremo un modello completamente formale. Comunque una definizione formale di sicurezza non è molto lontana dalla Definizione 8.2. Ci occuperemo qui invece di descrivere il primo schema di firma innegabile, introdotto da Chaum e Van Antwerpen [CA89] e basato sul problema del logaritmo discreto. A seguire, una serie di varianti sono state introdotte ed analizzate [Boy+90; CHP91; DY91; FOO91; Jak94]. Il primo schema basato su RSA è dovuto a Gennaro, Krawczyk e Rabin [GRK00; GRK07] ed è stato migliorato da Galbraith, Mao, Miyazaki e Pedersen [Miy00; GMP02; GM03].

Lo schema di Chaum e Van Antwepen è mostrato nel Crittosistema 8.6. Osserviamo che il protocollo di ripudio è essenzialmente costituito dall'invio di due sfide costruite in modo identico al protocollo di verifica (più la verifica finale). È immediato verificare che, se le due parti sono oneste, la verifica ha successo. Infatti

$$d \equiv c^{a^{-1} \bmod q} \equiv (\sigma^u \alpha^v)^{a^{-1} \bmod q} \equiv m^u g^v \pmod{p},$$

come desiderato.

Dobbiamo mostrare tre cose: che nessun attaccante (eventualmente limitato computazionalmente) sia in grado di forgiare una firma valida, che una firma generata in modo disonesto sia rinnegata dal protocollo di ripudio ed infine che Alice non sia in grado di rinnegare una firma da lei generata onestamente.

[56] Se Alice rifiuta di prendere parte al protocollo di ripudio, tale rifiuto può essere considerato una prova del fatto che ha qualcosa da nascondere.

Crittosistema 8.6. Firme innegabili di Chaum e Van Antwerpen

Schema di firma Π_{CVA}. Siano p, q primi, con q divisore di $p-1$. Sia inoltre $g \in \mathbb{Z}_p^*$ un generatore del sottogruppo $\mathbb{G}$ di $\mathbb{Z}_p^*$ di ordine q ed $\alpha = g^a \bmod p$, con $0 < a < q-1$. Poniamo $\mathcal{M} = \Sigma = \mathbb{G}$. Lo schema $\Pi_{\mathsf{CVA}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Dsvw})$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, genera $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ dove $pk = (\mathbb{G}, p, q, \alpha, g)$ ed $sk = a$.
- `Creazione della firma.` Dato $m \in \mathbb{G}$ calcola $\sigma = m^a \bmod p$.
- `Verifica.` Dati (m, σ) la verifica è eseguita in modo interattivo, come indicato di seguito:
  (i) il Bianconiglio sceglie $u, v \xleftarrow{\$} \mathbb{Z}_q^*$, ed invia ad Alice una sfida $c = \sigma^u \alpha^v \bmod p$;
  (ii) Alice invia la risposta $d = c^s \bmod p$, dove $s = a^{-1} \bmod q$;
  (iii) il Bianconiglio accetta la firma come valida se e solo se:
  $$d \equiv m^u g^v \pmod{p}.$$
- `Ripudio.` Dati (m, σ) il protocollo di ripudio procede come di seguito:
  (i) Alice ed il Bianconiglio eseguono la verifica della firma come previsto dal protocollo. Quindi il Bianconiglio verifica che $d \not\equiv m^u g^v \pmod{p}$. Sceglie allora $w, z \xleftarrow{\$} \mathbb{Z}_q^*$ ed invia ad Alice una nuova sfida $C = \sigma^w \alpha^z \bmod p$;
  (ii) Alice invia la risposta $D = C^s \bmod p$, dove $s = a^{-1} \bmod q$;
  (iii) il Bianconiglio controlla che $D \not\equiv m^w g^z \pmod{p}$. Infine conclude che la firma (m, σ) *non è stata prodotta da Alice* se e solo se
  $$(dg^{-v})^w \equiv (Dg^{-z})^u \pmod{p}.$$

Teorema 8.4 (Π_{CVA} è sicuro). *Lo schema di Chaum e Van Antwerpen verifica quanto segue:*

(i) nessun attaccante può forgiare una firma valida con probabilità migliore di $1/q$;

(ii) quando il protocollo di ripudio è eseguito usando una firma non valida, tale firma è rinnegata con probabilità 1*;*

(iii) quando il protocollo di ripudio è eseguito usando una firma valida, tale firma non è rinnegata almeno con probabilità $1 - 1/q$.

Dimostrazione. (i) Mostreremo che l'asserto è valido anche se l'attaccante (che esegue il protocollo di verifica al posto di Alice) non ha limite computazionale (quindi in particolare conosce $sk = a$), ovvero l'inforgiabilità è incondizionata. Osserviamo che, per costruzione, ad una sfida c nel protocollo di verifica corrispondono esattamente q coppie (u, v) (questo perché sia σ che α sono elementi di $\mathbb{G}$). Quando l'avversario riceve la sfida c, non ha modo di sapere quale coppia (u, v) è stata usata per costruire la sfida. Dobbiamo mostrare che se $\sigma \not\equiv m^a$ (cioè la firma è una forgiatura), allora ogni possibile risposta $d \in \mathbb{G}$ che l'attaccante può inviare è consistente con esattamente una delle q coppie (u, v). Ciò implica che la probabilitàà con cui l'attaccante invia una risposta d consistente con la firma forgiata è al più $1/q$, come desiderato.

Siccome g è un generatore di $\mathbb{G}$ possiamo scrivere ogni elemento in $\mathbb{G}$ come potenza di g, diciamo $c = g^i$, $d = g^j$, $m = g^k$ e $\sigma = g^\ell$ dove $i, j, \ell, k \in \mathbb{Z}_q$. Ora, per come c e d sono costruiti nel protocollo di verifica, dovrebbe essere verificato il seguente sistema di equazioni:

$$\begin{cases} i \equiv \ell u + av \pmod{q} \\ j \equiv ku + v \pmod{q}. \end{cases}$$

Siccome stiamo supponendo $\sigma \not\equiv m^a \pmod{p}$, segue $\ell \not\equiv ak \bmod q$. Pertanto, la matrice dei coefficienti del sistema ha determinante non nullo e la soluzione al sistema è unica:[57] ogni elemento d è la risposta corretta corrispondente ad esattamente una delle q possibili coppie (u, v), come volevasi dimostrare.
(ii) Siccome la firma in input al protocollo di ripudio non è valida, possiamo assumere $\sigma \not\equiv m^a \pmod{p}$. Per costruzione abbiamo la seguente congruenza:

$$\begin{aligned} (dg^{-v})^w &\equiv \left((\sigma^u \alpha^v)^s g^{-v}\right)^w \\ &\equiv \sigma^{usw} \cdot \alpha^{vsw} \cdot g^{-vw} \\ &\equiv \sigma^{usw} \cdot g^{ava^{-1}w} \cdot g^{-vw} \\ &\equiv \sigma^{usw} \pmod{p}. \end{aligned}$$

[57] È un fatto generale in algebra che un sistema di equazioni lineari in cui la matrice dei coefficienti ha determinante non nullo ammette una soluzione unica.

Una congruenza identica vale per la seconda sfida costruita usando C, D ovvero

$$(Dg^{-z})^u \equiv \sigma^{usw} \pmod{p},$$

quindi il protocollo di ripudio rinnega la firma (m, σ) con probabilità 1, come desiderato.
(iii) Siccome la firma in input al protocollo di ripudio è valida, possiamo assumere $\sigma \equiv m^a \pmod{p}$. Supponiamo inoltre che tutto quadri nel protocollo di ripudio, cioè che gli elementi d e D soddisfino (rispettivamente) $d \not\equiv m^u g^v \pmod{p}$ e $D \not\equiv m^w g^z \pmod{p}$. Dobbiamo mostrare che ciò implica $(dg^{-v})^w \not\equiv (Dg^{-z})^u \pmod{p}$ con probabilità almeno $1 - 1/q$.

Supponiamo che la firma sia rinnegata, ovvero $(dg^{-v})^w \equiv (Dg^{-z})^u \pmod{p}$ e deriviamo una contraddizione. L'ultima verifica nel protocollo di ripudio può essere scritta come $D \equiv (m^*)^w g^z \pmod{p}$, dove $m^* \equiv d^{1/u} g^{-v/u} \pmod{p}$. Usando quanto mostrato in (i) concludiamo che (m, σ^*) è una firma valida con probabilità $1 - 1/q$. Siccome stiamo assumendo che σ è anche la firma di m si deve avere con alta probabilità $m^a \equiv (m^*)^a \pmod{p}$, ovvero $m = m^*$. D'altra parte, il fatto che $d \not\equiv m^u g^v \pmod{p}$ implica $m \not\equiv d^{1/u} g^{-v/u} \pmod{p}$. Siccome $m^* \equiv d^{1/u} g^{-v/u} \pmod{p}$, segue $m \neq m^*$, chiaramente una contraddizione. □

Una soluzione più efficiente, che richiede solo tre messaggi tra Alice ed il Bianconglio è stata presentata in [KH05]. È anche possibile convertire una firma innegabile in una standard, si veda ad esempio [Boy+90; GMP02].

Esercizi

Esercizio 8.1. Alice pubblica i seguenti dati: $N = pq = 221$ ed $e = 13$. Quindi utilizza lo schema naïve RSA per firmare il messaggio $m = 65$ ed invia la relativa firma $\sigma = 182$ al Bianconiglio. Verificare la firma.

Esercizio 8.2. Considerare il seguente approccio alternativo per derivare una firma da RSA. Sia pad una funzione di codifica pubblica. Fissate (pk, sk) come nel Crittosistema 8.1, la firma di m è $\sigma = \mathsf{pad}(m)^e \bmod N$.

1. Spiegare come verificare la firma e come scegliere pad affinché l'attacco mostrato per naïve RSA sia evitato.

2. Mostrare che la codifica $\mathsf{pad}(m) = 0||m||0||m$, dove $|m| = (|N| - 1)/2$, è insicura.

Esercizio 8.3. Spiegare perché gli attacchi mostrati contro lo schema naïve RSA non sono attuabili nel caso di FDH.

Esercizio 8.4. Mostrare che lo schema di Lamport non è sicuro se utilizzato per cifrare più di un messaggio (con una data coppia di chiavi).

Esercizio 8.5. Dimostrare che l'esistenza degli schemi di firma monouso per messaggi lunghi 1 bit, implica l'esistenza delle funzioni unidirezionali.

Esercizio 8.6. Uno schema di firma monouso *forte* soddisfa la seguente definizione informale: Data una firma σ per un messaggio m, è difficile forgiare $(m, \sigma') \neq (m, \sigma)$ tale che σ' è una firma valida per m'. (Notare che a differenza della nozione di inforgiabilità standard, ora il caso $m' = m$ è ammesso, purché $\sigma \neq \sigma'$.)

1. Dare una definizione formale di firma monouso forte.

2. Assumendo l'esistenza delle funzioni unidirezionali, esibire una funzione unidirezionale f tale che lo schema di Lamport non è uno schema di firma monouso forte.

   (Suggerimento: sia f una funzione unidirezionale con dominio $\{0,1\}^n$, utilizzare la funzione $f'(x_1, \ldots, x_n) = f(x_1, \ldots, x_{n-1}, 0)$.)

3. Costruire uno schema di firma monouso forte usando una funzione unidirezionale appropriata nello schema di Lamport.

   (Suggerimento: sia H una funzione hash resistente alle collisioni che dimezza la lunghezza dell'input. Provare che H è unidirezionale. Quindi utilizzare H nello schema di Lamport.)

Esercizio 8.7. Consideriamo il seguente schema di firma monouso, basato su una permutazione unidirezionale f. La chiave pubblica comprende valori $y_1, \ldots, y_n$ e la chiave segreta include $x_1, \ldots, x_n$ tali che $f(x_i) = y_i$ per ogni $i = 1, \ldots, n$. Per firmare un messaggio $m = m_1 \ldots m_n$ lungo n bit, si calcola $\sigma = \{x_j\}_{m_j=1}$.

1. Mostrare che lo schema non è sicuro.

2. Suggerire una modifica che rende lo schema sicuro, aggiungendo un solo elemento alla chiave pubblica.

Esercizio 8.8. Supponiamo che Alice utilizzi il DSS con $q = 101$, $p = 7879$, $g = 170$, $a = 75$ ed $\alpha = 4567$. Determinare la firma di m tale che $H(m) = 10$, quando si usa la randomicità $b = 49$. Verificare la firma ottenuta.

Esercizio 8.9. Sia $\mathbb{G}$ il gruppo dei residui quadratici modulo $p = 467$, con generatore $g = 4$. Supponiamo di utilizzare lo schema di Chaum e Van Antwerpen in $\mathbb{G}$, con parametri $a = 101$, $\alpha = g^a \bmod p = 449$.

1. Alice firma il messaggio $m = 119$, ottenendo la firma $\sigma = 119^{101} \bmod 467 = 129$. Verificare la firma assumendo che il Bianconiglio sfidi Alice usando $u = 38$ e $v = 397$.

2. Sia $\sigma = 86$ una firma non autentica del messaggio $m = 286$. Alice vuole ripudiare σ. Eseguire il protocollo di ripudio, assumendo che il Bianconiglio utilizzi $u = 45$, $v = 237$, $w = 125$ e $z = 9$.

Esercizio 8.10. Sia Π uno schema di firma digitale sicuro nel modello standard. Sia H una funzione hash. Definiamo il seguente schema derivato Π_z dipendente da un valore z come segue. Quando $H(0) = z$ lo schema ritorna la chiave segreta sk, altrimenti esso è del tutto identico a Π. Mostrare che tale schema è sicuro nel modello dell'oracolo casuale per *ogni* valore di z. Mostrare che, tuttavia, esiste sempre un valore di z tale che Π_z è insicuro quando H non è un oracolo casuale.

Letture consigliate

[Boy+90] Joan Boyar, David Chaum, Ivan Damgård e Torben P. Pedersen. "Convertible Undeniable Signatures". In: *CRYPTO*. 1990, pp. 189–205.

[CA89] David Chaum e Hans Van Antwerpen. "Undeniable Signatures". In: *CRYPTO*. 1989, pp. 212–216.

[Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1865.

[CHP91] David Chaum, Eugène van Heijst e Birgit Pfitzmann. "Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer". In: *CRYPTO*. 1991, pp. 470–484.

[DH76] Whitfield Diffie e Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Trans. Inform. Theory* 22 (nov. 1976), pp. 644–654.

[DY91] Yvo Desmedt e Moti Yung. "Weakness of Undeniable Signature Schemes (Extended Abstract)". In: *EUROCRYPT*. 1991, pp. 205–220.

[FOO91] Atsushi Fujioka, Tatsuaki Okamoto e Kazuo Ohta. "Interactive Bi-Proof Systems and Undeniable Signature Schemes". In: *EUROCRYPT*. 1991, pp. 243–256.

[GM03] Steven D. Galbraith e Wenbo Mao. "Invisibility and Anonymity of Undeniable and Confirmer Signatures". In: *CT-RSA*. 2003, pp. 80–97.

[GMP02] Steven D. Galbraith, Wenbo Mao e Kenneth G. Paterson. "RSA-Based Undeniable Signatures for General Moduli". In: *CT-RSA*. 2002, pp. 200–217.

[GMR88] Shafi Goldwasser, Silvio Micali e Ronald L. Rivest. "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks". In: *SIAM J. Computing* 17.2 (apr. 1988), pp. 281–308.

[GRK00] Rosario Gennaro, Tal Rabin e Hugo Krawczyk. "RSA-Based Undeniable Signatures". In: *J. Cryptology* 13.4 (2000), pp. 397–416.

[GRK07] Rosario Gennaro, Tal Rabin e Hugo Krawczyk. "RSA-Based Undeniable Signatures". In: *J. Cryptology* 20.3 (2007), p. 394.

[Jak94] Markus Jakobsson. "Blackmailing using Undeniable Signatures". In: *EUROCRYPT*. 1994, pp. 425–427.

[KH05] Kaoru Kurosawa e Swee-Huay Heng. "3-Move Undeniable Signature Scheme". In: *EUROCRYPT*. 2005, pp. 181–197.

[Lam79] Leslie Lamport. *Constructing Digital Signatures from a One-Way Function*. Rapp. tecn. CSL-98. SRI International, ott. 1979.

[Mer87] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *CRYPTO*. 1987, pp. 369–378.

[Mer89] Ralph C. Merkle. "A Certified Digital Signature". In: *CRYPTO*. 1989, pp. 218–238.

[Miy00] Takeru Miyazaki. "An Improved Scheme of the Gennaro-Krawczyk-Rabin Undeniable Signature System Based on RSA". In: *ICISC*. 2000, pp. 135–149.

[Nac+94] David Naccache, David M'Raïhi, Serge Vaudenay e Dan Raphaeli. "Can D.S.A. be Improved? Complexity Trade-Offs with the Digital Signature Standard". In: *EUROCRYPT*. 1994, pp. 77–85.

[NIS91] NIST. *DSS: Digital Signature Standard*. Federal Information Processing Standard (FIPS). 1991.

[RSA78] Ronald L. Rivest, Adi Shamir e Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[Vau03] Serge Vaudenay. "The Security of DSA and ECDSA". In: *Public Key Cryptography*. 2003, pp. 309–323.

Capitolo 9

Reticoli e crittografia

Abbiamo visto nei capitoli precedenti, che il primo passo necessario per costruire un sistema sicuro è quello di individuare problemi computazionalmente "difficili". Affinché il sistema possa essere utile è necessario che esso sia: (i) difficile da violare anche per una scelta casuale delle chiavi e (ii) utilizzabile in modo efficiente dalle parti oneste.

In questo capitolo studieremo alcuni problemi computazionalmente difficili basati sui reticoli geometrici, ovvero insiemi di punti dello spazio multidimensionale con una qualche struttura periodica. Queste strutture hanno acquistato una grande attrattiva in crittografia per due motivi essenziali. Il primo motivo è che tali problemi sono ritenuti difficili anche contro avversari quantistici.[58] In particolare, non sono noti algoritmi quantistici per risolvere problemi basati sui reticoli che abbiano prestazioni migliori di quelli classici. Il secondo motivo è che i crittosistemi basati sui reticoli godono di una proprietà detta *difficoltà nel caso peggiore*. Ciò significa che esiste una prova matematica del fatto che violare il sistema è equivalente a risolvere un'istanza di un dato problema sui reticoli *nel caso peggiore*. In altri termini, dato un attaccante in grado di violare la sicurezza del sistema è possibile costruire un altro attaccante in grado di risolvere *ogni* istanza del problema computazionale sottostante.

Questa caratteristica distingue la crittografia basata sui reticoli da tutte le costruzioni che abbiamo visto finora, le quali invece godono di difficoltà nel caso medio. Ad esempio essere in grado di violare un crittosistema basato sul problema della fattorizzazione, può consentire di fattorizzare *alcuni* numeri scelti in accordo ad una certa distribuzione, ma non *tutti* i numeri.

Guida per il lettore. La struttura del capitolo è la seguente. Nel Paragrafo 9.1 introdurremo i reticoli geometrici e definiremo alcuni problemi com-

[58] Non ci soffermeremo su questo aspetto. Ci basti sapere che la realizzazione di un computer quantistico renderebbe attuabili alcuni attacchi in grado, ad esempio, di fattorizzare gli interi usati in RSA [Sho97].

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_9, © Springer-Verlag Italia 2012

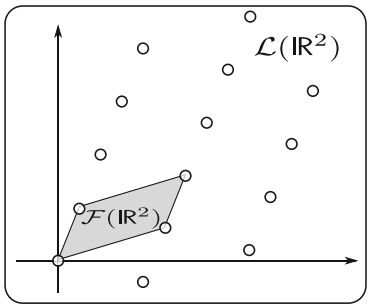


Fig. 9.1. Reticolo su $\mathbb{R}^2$

putazionali basati su di essi. Nel Paragrafo 9.2 discuteremo il problema di *imparare in presenza di errori*, intimamente connesso alla geometria dei reticoli e con svariate applicazioni crittografiche. Tale problema è dimostrabilmente tanto difficile nel caso medio quanto nel caso peggiore; nel Paragrafo 9.3 useremo questo fatto per definire un cifrario CPA-sicuro nel caso peggiore. Nel Paragrafo 9.4, infine, introdurremo il problema delle *somme di sottoinsiemi* e studieremo la sua connessione con il problema di imparare con gli errori.

9.1 La geometria dei numeri

Un *reticolo* è un insieme di punti nello spazio multidimensionale, con struttura periodica. Sia $\mathcal{V} \subset \mathbb{R}^n$ uno spazio vettoriale (ovvero $\mathcal{V}$ è chiuso rispetto alla somma ed alla moltiplicazione per uno scalare). La lunghezza di ciascun vettore $\mathbf{v} = (\mathbf{v}[1], \ldots, \mathbf{v}[n]) \in \mathbb{R}^n$ è espressa dall'usuale norma euclidea: $\|\mathbf{v}\| = \sqrt{(\mathbf{v}[1])^2 + \cdots + (\mathbf{v}[n])^2}$ (cf. Definizione A.1).

Dati n vettori *linearmente indipendenti* $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in $\mathbb{R}^n$, un reticolo è generato dall'insieme di tutte le combinazioni lineari a coefficienti interi di tali vettori:

$$\mathcal{L}(\mathbf{v}_1, \ldots, \mathbf{v}_n) = \sum_{i=1}^{n} \mathbb{Z}\mathbf{v}_i = \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{v}_i : \ \alpha_i \in \mathbb{Z} \right\}.^{59}$$

[59] Un insieme di vettori $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in $\mathcal{V}$ è linearmente indipendente se non esiste nessuna combinazione lineare a coefficienti in $\mathbb{R}$ che restituisce il vettore tutto nullo, ovvero se l'unico modo per ottenere:

$$\alpha_1 \mathbf{v}_1 + \ldots + \alpha_n \mathbf{v}_n = \mathbf{0} \qquad \text{con } \alpha_1, \ldots, \alpha_n \in \mathbb{R},$$

è porre $\alpha_1 = \ldots = \alpha_n = 0$.

Ogni insieme di vettori indipendenti che genera $\mathcal{L}$ è detto *base*. La dimensione di un reticolo è il numero di vettori in una base.

Alternativamente, indicata con $\mathbf{V} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ la matrice avente i vettori della base come colonne, possiamo definire un reticolo come l'insieme:

$$\mathcal{L}(\mathbf{V}) = \{\mathbf{V} \cdot \boldsymbol{\alpha} : \ \boldsymbol{\alpha} \in \mathbb{Z}^n\},$$

avendo indicato con $\mathbf{V} \cdot \boldsymbol{\alpha}$ l'usuale prodotto righe per colonne. Il determinante di un reticolo è il valore assoluto del determinante della matrice composta dai vettori della base, vale a dire $\det(\mathcal{L}(\mathbf{V})) = |\det(\mathbf{V})|$. (Si può dimostrare che tale quantità è un invariante, ovvero è indipendente dalla base scelta.) Si definisce *parallelepipedo fondamentale* (o dominio fondamentale) di un reticolo $\mathcal{L}(\mathbf{v}_1, \ldots, \mathbf{v}_n)$, l'insieme:

$$\mathcal{F}(\mathbf{V}) = \{\beta_1 \mathbf{v}_1 + \ldots + \beta_n \mathbf{v}_n : \ \beta_i \in [0, 1)\}.$$

Il motivo per cui tale insieme è detto dominio fondamentale è che ogni vettore dello spazio può esprimersi come somma di un unico vettore in $\mathcal{F}$ più un unico vettore in $\mathcal{L}$ (cf. Esercizio 9.1). In altri termini, $\mathbb{R}^n = \{\mathcal{F}(\mathbf{V}) + \mathbf{x} : \ \mathbf{x} \in \mathcal{L}(\mathbf{V})\}$. Si veda la Fig. 9.1 per un esempio di reticolo in $\mathbb{R}^2$.

La branca della teoria dei numeri che studia le proprietà dei reticoli è detta *geometria dei numeri* ed è stata iniziata da Hermann Minkowski nel 1910. Il seguente teorema riporta alcune proprietà fondamentali dei reticoli geometrici. Si veda [HPS08, Capitolo 6] per una dimostrazione (cf. anche Esercizio 9.2 ed Esercizio 9.3).

Teorema 9.1 (Alcune proprietà dei reticoli). *Ogni reticolo n-dimensionale $\mathcal{L}$ verifica quanto segue:*

(i) Disuguaglianza di Hadamard. *Sia $\mathbf{v}_1, \ldots, \mathbf{v}_n$ una base di $\mathcal{L}$, allora*

$$\det(\mathcal{L}) \leq \|\mathbf{v}_1\| \|\mathbf{v}_2\| \cdots \|\mathbf{v}_n\|.$$

Inoltre il valore $\det(\mathcal{L})$ *è un'invariante per il reticolo $\mathcal{L}$.*

(ii) Teorema di Hermite. *Esiste un vettore $\mathbf{v} \in \mathcal{L}$ non nullo tale che:*

$$\|\mathbf{v}\| \leq \sqrt{n} \det(\mathcal{L})^{1/n}.$$

(iii) Teorema di Minkowski. *Sia $\mathcal{S} \subset \mathbb{R}^n$ un insieme simmetrico convesso*[60] *tale che* $\mathrm{Vol}(\mathcal{S}) > 2^n \det(\mathcal{L})$. *Allora $\mathcal{S}$ contiene un vettore di $\mathcal{L}$ non nullo.*

Problemi sui reticoli. Come anticipato, i reticoli consentono di definire numerosi problemi computazionali. Uno dei problemi più famosi è il problema del vettore più corto (*Shortest Vector Problem*, SVP): data una base $\mathbf{V}$ per un reticolo $\mathcal{L}$, si richiede di trovare il vettore *più corto* in $\mathcal{L}(\mathbf{V})$, indicato con $\boldsymbol{\lambda}_1(\mathcal{L})$. Un altro problema è il problema del vettore più vicino (*Closest Vector Problem*, CVP): data una base $\mathbf{V}$ per un reticolo $\mathcal{L}$ ed un elemento $\mathbf{w} \in \mathbb{R}^n$, si richiede di trovare l'elemento in $\mathcal{L}(V)$ *più vicino* a $\mathbf{w}$. Il problema dei vettori più corti indipendenti (*Shortest Independent Vectors Problem*, SIVP), invece, data una base $\mathbf{V}$ per un reticolo $\mathcal{L}$, richiede di trovare n vettori linearmente indipendenti $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$ (con $\mathbf{s}_i \in \mathcal{L}(\mathbf{V})$) che minimizzino la quantità $\|\mathbf{S}\| = \max_i \|\mathbf{s}_i\|$. Ciascuno di questi problemi ammette una variante approssimata, indicata rispettivamente con SVP_γ, CVP_γ e SIVP_γ, in cui si rilassa il problema accettando una soluzione approssimata fino ad una soglia γ.

Un famoso algoritmo[61] — detto LLL e dovuto ad A. H. Lenstra, H. W. Lenstra e Lovàsz [LLL82] — consente di risolvere SVP_γ per $\gamma > 2^{n \log\log n / \log n}$ in tempo polinomiale (si vedano anche [Sch87; AKS02]). D'altra parte sia SVP_γ che CVP_γ sono noti essere **NP**-difficili per $\gamma < n^{c/\log\log n}$ (si vedano rispettivamente [HR07] e [Din+03]). I casi $\gamma = n$, $\sqrt{n}/\log n$ e $\sqrt{n}$ sono stati studiati (rispettivamente) in [LJS90; GG98; AR04]. Solitamente le applicazioni crittografiche sono possibili a partire da $\gamma > n$.

Uso diretto dei reticoli in crittografia. Esistono crittosistemi la cui sicurezza è riconducibile in modo diretto ad istanze dei problemi SVP e CVP (o le loro versioni approssimate). I più importanti in questo contesto sono: il crittosistema di Ajtai-Dwork [AD97], il crittosistema GGH di Goldreich, Goldwasser ed Halevi [GGH97] ed il crittosistema NTRU di Hoffstein, Pipher e Silverman [HPS98]. Solitamente questi crittosistemi hanno un'efficienza com-

[60] Un sottoinsieme $\mathcal{S} \subset \mathbb{R}^n$ è:

- *limitato*, se la lunghezza dei vettori in $\mathcal{S}$ è limitata;
- *simmetrico* se ogni per ogni $\mathbf{a}$ in $\mathcal{S}$ anche $-\mathbf{a}$ appartiene ad $\mathcal{S}$;
- *convesso*, se per ogni coppia $\mathbf{a}, \mathbf{b}$ di elementi in $\mathcal{S}$, la linea che congiunge $\mathbf{a}$ e $\mathbf{b}$ giace completamente in $\mathcal{S}$.

[61] Tale algoritmo è stato introdotto in un contesto più generale di quello dei reticoli, in particolare per la fattorizzazione di polinomi a coefficienti in $\mathbb{Q}$.

putazionale molto più elevata di RSA o ElGamal: fissato un dato livello di sicurezza (diciamo n bit) i secondi richiedono tipicamente $O(n^3)$ operazioni, mentre i primi solo $O(n^2)$. Tuttavia la dimensione delle chiavi è spesso non pratica, come evidenziato dagli esempi di seguito.

Ajtai e Dwork hanno mostrato che il loro schema è tanto sicuro quanto risolvere un'istanza *nel caso peggiore* di SVP. La chiave ha lunghezza $O(n^4)$ il che rende lo schema non utilizzabile in pratica. In seguito Nguyen e Stern [NS98] hanno mostrato che ogni scelta dei parametri per il crittosistema che lo rende utilizzabile in pratica, è intrinsecamente insicura.

Nel crittosistema GGH la dimensione della chiave è $O(n^2)$. Nguyen [Ngu06] ha mostrato un attacco in grado di violare la sicurezza dello schema fino a valori di n intorno a 350. Per $n > 400$ la chiave ha dimensione 128 kB.

Nel crittosistema NTRU la dimensione della chiave è $O(n \log n)$. Non ci sono attacchi noti.

9.2 Imparare in presenza di errori

In questo paragrafo introduciamo un problema computazionale che sembra essere del tutto sconnesso dai reticoli, ma che si rivelerà essere intimamente connesso ad essi: il problema di imparare in presenza di errori (*Learning With Errors*, LWE). Nel Paragrafo 9.3 vedremo un esempio di crittosistema a chiave pubblica CPA-sicuro basato sul problema LWE.

Versione computazionale. Il problema LWE, nella sua versione computazionale, è stato introdotto da Regev [Reg05] e può essere formulato come segue. Sia q un primo, ed $\mathbf{s} \in \mathbb{Z}_q^n$. Si consideri la seguente lista di "equazioni con errore":

$$\begin{aligned} \mathbf{r}_1^\mathsf{T} \cdot \mathbf{s} &\approx_\chi z_1 \pmod q \\ \mathbf{r}_2^\mathsf{T} \cdot \mathbf{s} &\approx_\chi z_2 \pmod q \\ &\dots \\ &\dots \end{aligned}$$

dove i vettori $\mathbf{r}_i$ sono scelti indipendentemente ed uniformemente in $\mathbb{Z}_q^n$ e $z_i \in \mathbb{Z}_q$. L'errore è specificato da una distribuzione $\chi : \mathbb{Z}_q \to \mathbb{R}^+$ su $\mathbb{Z}_q$, in particolare abbiamo $z_i \equiv \mathbf{r}_i \cdot \mathbf{s} + e_i \pmod q$ con $e_i \xleftarrow{\$} \chi$. Un esempio tipico

per la distribuzione χ è la distribuzione Gaussiana discreta $\mathsf{Gau}_{q,\tau}$ definita come segue: si estrae $x \in \mathbb{R}$ in accordo alla distribuzione Gaussiana con ampiezza τ (ovvero x è scelto con probabilità $\frac{1}{\tau}\exp(-\pi x^2/\tau^2)$) e si restituisce $\lfloor x \cdot q \rceil \bmod q$.

Il problema LWE computazionale richiede di recuperare $\mathbf{s}$ data la lista di equazioni di cui sopra. Scriveremo $\mathrm{LWE}^q_{\tau,n}$ per indicare un'istanza di tale problema. Più precisamente, sia $\Lambda^q_{\tau,n}(\mathbf{s})$ la distribuzione di probabilità su $\mathbb{Z}_q^n \times \mathbb{Z}_q$ ottenuta estraendo uniformemente $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^n$ ed $e \xleftarrow{\$} \mathsf{Gau}_{\tau,q}$ e restituendo $(\mathbf{r}, \mathbf{r}^\mathsf{T} \cdot \mathbf{s} + e \bmod q)$.

Definizione 9.1 (Problema LWE computazionale). Diremo che il problema $\mathrm{LWE}^q_{\tau,n}$ computazionale è (t, Q, ϵ)-difficile se nessun attaccante PPT $\mathscr{A}$ eseguibile in tempo t che richiede Q campioni all'oracolo $\Lambda^q_{\tau,n}(\mathbf{s})$, può recuperare $\mathbf{s}$ con probabilità migliore di ϵ. In simboli:

$$\mathbb{P}\left[\mathscr{A}^{\Lambda^q_{\tau,n}(\mathbf{s})}(1^n) = \mathbf{s} : \ \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n\right] \leq \epsilon.$$

■

Osserviamo che se non fosse per l'errore, sarebbe possibile recuperare $\mathbf{s}$ una volta note n equazioni linearmente indipendenti semplicemente risolvendo un sistema lineare di n equazioni in n incognite. L'applicazione dello stesso metodo quando l'errore è presente porta ad un'amplificazione incontrollata dell'errore stesso.

Difficoltà di LWE. Un modo naïve per risolvere il problema LWE è il seguente: continuare ad interrogare l'oracolo $\Lambda^q_{\tau,n}(\mathbf{s})$ finchè non si ottengono $\mathrm{poly}(n)$ campioni della forma $(\mathbf{r}, z)$ tali che $\mathbf{r} = (1, 0, \ldots, 0)$), così da poter recuperare $\mathbf{s}[1]$. Si ripete quindi il procedimento per ogni elemento $\mathbf{s}[i]$ in $\mathbf{s}$. La probabilità di vedere una tale equazione è q^{-n}, per cui l'algoritmo richiede $Q = 2^{O(n \log n)}$ interrogazioni ed ha un tempo d'esecuzione esponenziale. Un altro algoritmo dovuto a Blum, Kalai e Wasserman [BKW03] richiede solamente $2^{O(n)}$ campioni e tempo d'esecuzione simile. Questo è il miglior algoritmo noto.

Più in generale, il problema LWE è stato mostrato essere equivalente ad un problema computazionale sui reticoli [Reg05; Pei09]: il problema della decodifica a distanza limitata (*Bounded Decoding Distance*, BDD). Per un qualche parametro $d > 0$, sono dati un reticolo $\mathcal{L}$ ed un vettore $\mathbf{x}$ a distanza d da $\mathcal{L}$ e si richiede di trovare il vettore del reticolo più vicino ad $\mathbf{x}$. Si può mostrare che

quando $d < \|\boldsymbol{\lambda}_1(\mathcal{L})\|$ la soluzione è unica. Regev [Reg05], ha costruito una *riduzione quantistica* del problema LWE al problema BDD (nel caso peggiore). In altre parole dato un algoritmo in grado di risolvere un'istanza casuale del problema LWE, è possibile costruire un algoritmo quantistico in grado di risolvere *ogni* istanza del problema BDD. Tale algoritmo è poi utilizzabile per risolvere SIVP o SVP_γ [Reg05]. Esiste anche una riduzione puramente classica [Pei09].

Versione decisionale. È possibile anche definire una versione *decisionale* del problema LWE. In breve, in questo caso il problema richiede di distinguere la distribuzione $\Lambda^q_{\tau,n}(\mathbf{s})$ su $\mathbb{Z}_q^n \times \mathbb{Z}_q$ (per un certo $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$) dalla distribuzione uniforme su $\mathbb{Z}_q^{n+1}$ (indicata nel seguito con U^q_{n+1}).

Definizione 9.2 (Problema LWE decisionale). Diremo che il problema $\text{LWE}^q_{\tau,n}$ decisionale è (t, Q, ϵ)-difficile se, per ogni attaccante PPT $\mathscr{D}$ eseguibile in tempo t e che richiede Q campioni, si ha:

$$\left| \mathbb{P}\left[\mathscr{D}^{\Lambda^q_{\tau,n}(\mathbf{s})}(1^n) = 1 : \ \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n \right] - \mathbb{P}\left[\mathscr{D}^{U^q_{n+1}}(1^n) = 1 \right] \right| \leq \epsilon.$$

■

La versione decisionale è in realtà equivalente a quella computazionale:

Lemma 9.2 (LWE decisionale vs. LWE computazionale). *Sia $n \geq 1$ un intero e $2 \leq q \leq \text{poly}(n)$ un primo. Se esiste un attaccante PPT $\mathscr{D}$ in grado di risolvere la versione decisionale del problema LWE (con probabilità esponenzialmente vicina ad 1) per un qualche $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, allora esiste un attaccante PPT $\mathscr{A}$ che recupera $\mathbf{s}$ (con probabilità esponenzialmente vicina ad 1).*

Dimostrazione (bozza). Mostriamo come sia possibile usare $\mathscr{D}$ per recuperare $\mathbf{s}[1]$, ovvero il primo elemento di $\mathbf{s}$. (Il procedimento può poi essere ripetuto per recuperare gli altri elementi e quindi tutto il vettore $\mathbf{s}$.) L'attaccante $\mathscr{A}$ riceve campioni della forma $(\mathbf{r}, z) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ dove $z = \mathbf{r}^\mathsf{T} \cdot \mathbf{s} + e \bmod q$ con $e \xleftarrow{\$} \mathsf{Gau}_{q,\tau}$. Quindi $\mathscr{A}$ procede come segue:

1. Estrai $\mathbf{r}'[1], \mathbf{s}'[1] \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\mathbf{r}' = \mathbf{r} + (\mathbf{r}'[1], 0, \ldots, 0) \bmod q \in \mathbb{Z}_q^n$ e $z' = z + \mathbf{s}'[1] \cdot \mathbf{r}'[1] \in \mathbb{Z}_q$.
2. Lancia $\mathscr{D}(\mathbf{r}', z')$. Se l'output è 0 torna al passo (1).
3. Quando $\mathscr{D}$ restituisce 1 ritorna $\mathbf{s}'[1]$ e fermati.

È immediato vedere che quando (e solo quando) $\mathbf{s}'[1] = \mathbf{s}[1]$ la coppia $(\mathbf{r}', z')$ è distribuita come se fosse stata estratta da $\Lambda^q_{\tau,n}(\mathbf{s})$. D'altra parte, se ciò non accade, la coppia $(\mathbf{r}', z')$ è uniformemente casuale in $\mathbb{Z}_q^{n+1}$. Possiamo allora stabilire se $\mathbf{s}'[1] = \mathbf{s}[1]$ invocando $\mathscr{D}$ al più $Q = \text{poly}(n)$ volte. □

In [Pei09] il lemma precedente è esteso al caso in cui q è prodotto di primi distinti.

Dal caso medio al caso peggiore. Mostriamo ora l'equivalenza tra complessità nel caso medio e nel caso peggiore. In generale nelle applicazioni crittografiche di LWE il vettore $\mathbf{s}$ è scelto uniformemente a caso. Come vedremo risolvere il problema decisionale con probabilità non trascurabile per una scelta uniforme del vettore $\mathbf{s}$ in $\mathbb{Z}_q^n$ è sufficiente a risolvere il problema per *tutti* i possibili $\mathbf{s}$.

Lemma 9.3 (Dal caso medio al caso peggiore). *Siano $n, q > 1$ interi. Supponiamo che esista un attaccante PPT $\mathscr{D}$ in grado di distinguere $\Lambda^q_{\tau,n}(\mathbf{s})$ da U^q_{n+1} per una frazione non trascurabile di tutti i possibili vettori $\mathbf{s}$. Allora esiste un attaccante PPT $\mathscr{D}'$ che distingue $\Lambda^q_{\tau,n}(\mathbf{s})$ da U^q_{n+1} per ogni possibile vettore $\mathbf{s}$.*

Dimostrazione (bozza). Consideriamo la mappa $f_{\mathbf{s}'} : \mathbb{Z}_q^n \times \mathbb{Z}_q \to \mathbb{Z}_q^n \times \mathbb{Z}_q$ che, per ogni $\mathbf{s}' \in \mathbb{Z}_q^n$, calcola:

$$f_{\mathbf{s}'}(\mathbf{r}, z) = (\mathbf{r}, z + \mathbf{r}^\mathsf{T} \cdot \mathbf{s}').$$

È facile vedere che tale mappa trasforma $\Lambda^q_{\tau,n}(\mathbf{s})$ in $\Lambda^q_{\tau,n}(\mathbf{s} + \mathbf{s}')$. D'altra parte la distribuzione U^q_{n+1} è mappata su se stessa.

L'algoritmo $\mathscr{D}'$ riceve come input un campione $(\mathbf{r}, z)$ prelevato da $\Lambda^q_{\tau,n}(\mathbf{s})$ e deve stabilire a quale distribuzione appartiene. Per far ciò ripete quanto segue un numero polinomiale di volte:

1. Estrai un vettore $\mathbf{s}' \xleftarrow{\$} \mathbb{Z}_q^n$.
2. Stima la probabilità che $\mathscr{D}$ restituisca 1 quando è lanciato con un input prelevato da U^q_{n+1} o con input $f_{\mathbf{s}'}(\mathbf{r}, z)$.
3. Se la differenza tra le due stime è "grande" restituisci 1, altrimenti ritorna 0.

Ovviamente quando l'input è prelevato da U^q_{n+1}, per ogni $\mathbf{s}'$ l'input dato a $\mathscr{D}$ è prelevato dalla stessa distribuzione, quindi la probabilità che $\mathscr{D}'$ distingua le due distribuzioni è esponenzialmente vicina a 0. Se, d'altra parte, utilizziamo $(\mathbf{r}, z)$, allora con alta probabilità troveremo un vettore $\mathbf{s}'$ per cui $\mathscr{D}$ distingue $\Lambda^q_{\tau,n}(\mathbf{s}+\mathbf{s}')$ da U^q_{n+1}. In questo caso, quindi, $\mathscr{D}'$ distingue le due distribuzioni con probabilità esponenzialmente vicina ad 1. □

Imparare la parità in presenza di rumore. Il problema LWE può essere visto come una generalizzazione del problema di imparare la parità in presenza di rumore (*Learning Parity with Noise*, LPN), già noto ed utilizzato prima del lavoro di Regev, ad esempio da Hopper e Blum [HB01]. In questo caso tutte le stringhe sono binarie (i.e., $q = 2$) e tutte le operazioni sono effettuate modulo 2; inoltre la distribuzione χ è scelta pari alla distribuzione Bernoulliana Ber_τ con parametro τ (i.e., $\mathbb{P}[e = 1] = \tau$ se $e \xleftarrow{\$} \mathsf{Ber}_\tau$). Incontreremo nuovamente il problema LPN nel Capitolo 11, quando ci occuperemo di autenticazione (cf. Paragrafo 11.4).

9.3 Il cifrario di Regev

Costruiamo uno schema di cifratura a chiave pubblica basato su LWE. Il cifrario Π_{Regev}, dovuto a Regev, è mostrato nel Crittosistema 9.1. Notare che il sistema è in grado di cifrare un singolo bit[62] e che tutte le operazioni sono effettuate modulo q. È facile verificare che la decifratura restituisce il bit cifrato con alta probabilità. Innanzitutto, se non fosse per l'errore, la quantità $z - \mathbf{r}^\mathsf{T} \cdot \mathbf{s}$ sarebbe esattamente 0 quando viene trasmesso 0 e $\lfloor \frac{q}{2} \rfloor$ quando viene trasmesso 1. Ne deriva che è presente un errore in decifratura solo quando la somma degli errori e_i su tutto $\mathcal{S}$ eccede $q/4$. Siccome stiamo sommando al più ℓ termini Gaussiani, ognuno con deviazione standard $\tau \cdot q$, la deviazione standard della somma è al più $\sqrt{\ell}\tau \cdot q < q/\log n$. Invocando il limite di Chernoff (cf. Teorema A.5), possiamo concludere che la probabilità che tale somma ecceda $q/4$ è esponenzialmente piccola.

Resta da discutere la sicurezza del sistema:

Lemma 9.4 (Π_{Regev} è CPA-sicuro). *Se il problema* $\mathrm{LWE}^q_{\tau,n}$ *è difficile,* Π_{Regev} *è CPA-sicuro.*

[62] Si ricorda che, per il Teorema 6.1, è possibile cifrare un messaggio a lunghezza arbitraria cifrando i singoli bit indipendentemente.

Crittosistema 9.1. Il cifrario di Regev

Sia n il parametro di sicurezza, ℓ un intero, q un primo e τ il parametro di rumore. Posto $\mathcal{M} = \{0,1\}$ e $\mathcal{C} = \mathbb{Z}_q^{n+1}$ Il cifrario $\Pi_{\mathsf{Regev}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è definito come segue:

- `Generazione delle chiavi.` Dato come input il parametro di sicurezza n genera $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$, dove $sk = \mathbf{s}$ (con $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$) e $pk = (\mathbf{r}_i, z_i)_{i=1}^{\ell}$ consiste di ℓ campioni prelevati da $\Lambda_{\tau,n}^q(\mathbf{s})$.
- `Cifratura.` Dato un bit $b \in \{0,1\}$ da trasmettere, scegli un insieme $\mathcal{S}$ uniformemente a caso tra i 2^ℓ sottoinsiemi di $[\ell]$. La cifratura di b è:

$$\mathbf{c} \leftarrow \mathsf{Enc}_{pk}(b) = \begin{cases} \left(\sum_{i\in\mathcal{S}} \mathbf{r}_i, \sum_{i\in\mathcal{S}} z_i\right) & \text{se } b = 0 \\ \left(\sum_{i\in\mathcal{S}} \mathbf{r}_i, \left\lfloor \frac{q}{2} \right\rfloor \sum_{i\in\mathcal{S}} z_i\right) & \text{se } b = 1 \end{cases}$$

- `Decifratura.` La decifratura di una coppia $\mathbf{c} = (\mathbf{r}, z) \in \mathbb{Z}_q^{n+1}$ è 0 se $z - \mathbf{r}^\mathsf{T} \cdot \mathbf{s}$ è più vicino a 0 che a $\left\lfloor \frac{q}{2} \right\rfloor$ modulo q ed 1 altrimenti.

Dimostrazione (bozza). Supponiamo che il sistema non sia CPA-sicuro, per cui esiste un attaccante PPT $\mathscr{A}$ che esegue l'esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{PKE},\Pi_{\mathsf{Regev}}}(\mathscr{A}, n)$ con vantaggio non trascurabile (cf. Definizione 6.2). In altre parole $\mathscr{A}$, data la chiave pubblica $pk = (\mathbf{r_i}, z_i)_{i=1}^{\ell}$ prelevata dalla distribuzione $\Lambda_{\tau,n}^q(\mathbf{s})$ e la cifratura $\mathbf{c}_b$ di un bit b a caso, può distinguere la cifratura del bit 0 da quella del bit 1 con probabilità almeno $1/2 + \epsilon(n)$ per una frazione non trascurabile di tutti i possibili vettori $\mathbf{s} \in \mathbb{Z}_q^n$. Mostriamo come usare $\mathscr{A}$ per costruire un algoritmo PPT $\mathscr{D}$ in grado di risolvere il problema $\mathrm{LWE}_{\tau,n}^q$ con vantaggio non trascurabile per una frazione di tutti i possibili vettori $\mathbf{s} \in \mathbb{Z}_q^n$. Applicando i Lemmi 9.2 e 9.3, otteniamo una soluzione del problema LWE per ogni istanza, contro l'ipotesi che il problema LWE sia difficile. L'attaccante $\mathscr{D}$ ha accesso ad un oracolo che restituisce campioni della forma $(\mathbf{r}, z) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ e deve stabilire se l'oracolo è $\Lambda_{\tau,n}^q(\mathbf{s})$ oppure U_{n+1}^q. Per far ciò, $\mathscr{D}$ agisce come segue:

1. Interroga l'oracolo ℓ volte e poni $pk = (\mathbf{r}_i, z_i)_{i=1}^{\ell}$.
2. Lancia $\mathscr{A}(pk)$ e scegli un bit $b \xleftarrow{\$} \{0,1\}$. Cifra quindi b come nel Crittosistema 9.1, calcolando $\mathbf{c}_b \leftarrow \mathsf{Enc}_{pk}(b)$. Inoltra $\mathbf{c}_b$ ad $\mathscr{A}$.
3. Ritorna lo stesso bit che restituisce $\mathscr{A}$.

Ora è chiaro che quando gli ℓ campioni che $\mathscr{D}$ riceve dal suo oracolo sono distribuiti secondo $\Lambda^q_{\tau,n}(\mathbf{s})$ la simulazione dell'esperimento $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{PKE},\Pi_{\text{Regev}}}(\mathscr{A}, n)$ da parte di $\mathscr{D}$ è perfetta. Quindi:

$$\mathbb{P}\left[\mathscr{D}^{\Lambda^q_{\tau,n}(\mathbf{s})}(1^n) = 1\right] \geq \frac{1}{2} + \epsilon(n).$$

D'altra parte supponiamo che l'oracolo di $\mathscr{D}$ sia U^q_{n+1}. In questo caso la chiave pubblica simulata da $\mathscr{D}$ è costituita da coppie $(\mathbf{r}_i, z_i)_{i=1}^{\ell}$ uniformemente casuali in $\mathbb{Z}_q^{n+1}$. Si può allora verificare che le condizioni del lemma dell'hash residuo (cf. Teorema 3.8) sono verificate, così che con probabilità alta sulle scelte possibili per $(\mathbf{r}_i, z_i)_{i=1}^{\ell}$, la distribuzione su $\mathcal{S}$ di un sottoinsieme casuale $(\sum_{i\in\mathcal{S}} \mathbf{r}_i, \sum_{i\in\mathcal{S}} z_i)$ è vicina alla distribuzione uniforme (in distanza statistica, cf. Definizione A.3). Ne segue che il nostro algoritmo in questo caso non ha modo di distinguere il caso $b = 0$ dal caso $b = 1$ con probabilità migliore di $1/2$. Pertanto:

$$\mathbb{P}\left[\mathscr{D}^{U^q_{n+1}}(1^n) = 1\right] \leq \frac{1}{2}.$$

Mettendo tutto insieme, abbiamo mostrato:

$$\left|\mathbb{P}\left[\mathscr{D}^{\Lambda^q_{\tau,n}(\mathbf{s})}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{U^q_{n+1}}(1^n) = 1\right]\right| > \epsilon(n),$$

contro l'ipotesi che il problema LWE sia difficile. □

Una scelta possibile dei parametri è $n^2 \leq q \leq 2n^2$, $\ell = 1.1 \cdot n \log q$ e $\tau = 1/(\sqrt{n}\log^2 n)$. Notare che il crittosistema è altamente inefficiente: la chiave pubblica ha dimensione $O(\ell \cdot n \log q)$ e la cifratura aumenta la dimensione del messaggio di un fattore $O(n \log q)$.

Altre applicazioni di LWE. Il problema LWE ha numerosissime applicazioni in crittografia e come notato dallo stesso Regev [Reg10], esso era già implicito nel crittosistema di Ajtai-Dwork ed in GGH. In particolare è stato usato per costruire: crittosistemi CPA-sicuri [Reg05; KTX07; PVW08] e CCA-sicuri [PW08; Pei09], protocolli per il trasferimento oblivio [PVW08] (cf. Paragrafo 14.2), crittosistemi su base identità [GPV08; Cas+10; ABB10] (cf. Capitolo 10), diverse forme di crittosistemi resistenti alle perdite [AGV09; App+09; Dod+10; Gol+10], MAC e protocolli di autenticazione [Kil+11]. Molte di queste soluzioni sono efficienti e quindi utilizzabili in pratica.

Il problema SIS ed estensioni su anelli di polinomi. Il problema della soluzione agli interi più piccoli (*Smallest Integer Solution*, SIS) può essere visto come il problema duale di LWE: data una sequenza di vettori $\mathbf{r}_1, \mathbf{r}_2 \ldots$ scelti uniformemente in $\mathbb{Z}_q^n$ si richiede di trovare un sottoinsieme di essi (oppure una loro combinazione lineare con piccoli coefficienti) la cui somma sia zero modulo q. Il problema è stato proposto per la prima volta in [MR04], ma era già implicito nel lavoro di Ajtai e Dwork. Tale problema è stato usato per realizzare alcuni membri di Minicrypt: funzioni unidirezionali [Ajt96], funzioni hash resistenti alle collisioni [GGH96], firme digitali [GPV08; Cas+10] ed alcuni schemi di identificazione [MV03; Lyu08; KTX08]. Si può mostrare che per ogni primo $q = \text{poly}(n)$, una soluzione del problema SIS permette di risolvere SIVP oppure SVP_γ.

Solitamente gli schemi basati sui problemi SIS o LWE richiedono chiavi di grandi dimensioni, tipicamente dell'ordine di n^2. Un modo per superare questa limitazione è dotare i campioni LWE di una qualche *struttura*. Ad esempio è possibile rimpiazzare $\mathbb{Z}_q^n$ con l'anello di polinomi a coefficienti in $\mathbb{Z}_q$ modulo il polinomio $X^n + 1$ (solitamente indicato con $\mathbb{Z}_q[X]/(X^n + 1)$); si parla in questo caso di LWE e SIS su anelli (rispettivamente RLWE ed RSIS). (Normalmente n deve essere una potenza di 2, altrimenti alcune cose cominciano ad andare storte.) Il problema RSIS è riconducibile a problemi standard sui reticoli [Mic02; PR06; LM06] ed è stato utilizzato per costruire funzioni hash resistenti alle collisioni [PR06; LM06; Lyu+08], alcuni schemi di identificazione [Lyu09] e firme digitali [LM08; Lyu09] (tutti efficienti). Il problema RLWE è riducibile ad RSIS [Ste+09; LPR10].

9.4 Somme di sottoinsiemi

Il problema delle somme di sottoinsiemi (*Subset Sum*, SS) è parametrizzato da due interi n ed M. Indicheremo un'istanza del problema con $\text{SS}(n, M)$. Dato un vettore segreto $\mathbf{s} \in \mathbb{Z}_2^n$, si sceglie $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_M^n$ e si restituisce $(\mathbf{r}, S)$ dove $S = \mathbf{r}^\mathsf{T} \cdot \mathbf{s} \bmod M$. Il problema richiede di recuperare $\mathbf{s}$, data la coppia $(\mathbf{r}, S)$. In generale la complessità del problema dipende dal rapporto tra n e $\log M$: quando $n/\log^2 n < n/\log M < 1/n$ il problema può essere risolto in tempo polinomiale [LO85; Fri86; FP05; Lyu05; Sha08], quando $n/\log M$ è costante oppure $O(1/\log n)$ gli algoritmi noti hanno complessità esponenziale in n. A volte il problema è anche detto problema del sacco (*knapsack* in inglese) in quanto può essere interpretato come segue: data una serie di oggetti con un certo peso in chilogrammi (contenuto nel vettore $\mathbf{r}$) determinare un sottoinsie-

me di essi (rappresentato dal vettore $\mathbf{s}$) tale da riempire un sacco contenente esattamente S chilogrammi.

Già nel 1970, Merkle ed Hellman [MH78] hanno proposto un crittosistema a chiave pubblica basato sul problema SS. Per fare ciò hanno considerato istanze particolari del problema in cui gli elementi del vettore $\mathbf{s}$ costituiscono una sequenza *super-crescente*: $\mathbf{s}[i] \geq 2\mathbf{s}[i-1]$ per ogni $i = 1, \ldots, n$. In seguito altre varianti sono state proposte, tuttavia questi schemi si sono poi mostrati insicuri (si veda [Odl90] per una panoramica), poiché basati su istanze speciali del problema SS che introducono debolezze intrinseche (si veda anche l'Esercizio 9.8). Impagliazzo e Naor [IN96], hanno invece costruito alcuni membri di Minicrypt: funzioni hash resistenti alle collisioni, generatori pseudocasuali e schemi di impegno (cf. Paragrafo 14.3). A differenza degli altri schemi, la sicurezza di queste costruzioni è basata su un'istanza *casuale* del problema SS. Recentemente Lyubashevsky, Palacio e Segev [LPS10], hanno costruito un crittosistema a chiave pubblica CPA-sicuro basato sull'ipotesi che SS (nel caso medio) sia difficile.

Connessione con LWE. Gli autori in [LPS10], hanno mostrato il seguente elegante collegamento tra SS ed LWE. Consideriamo un'istanza $\mathrm{SS}(n, q^m)$ dove q è un piccolo intero. Quando $\mathbf{r} \in \mathbb{Z}_{q^m}^n$ ed $\mathbf{s} \in \mathbb{Z}_2^n$, il prodotto scalare $\mathbf{r}^\mathsf{T} \cdot \mathbf{s} \bmod q^m$ può essere scritto in base q come $\mathbf{R}^\mathsf{T} \cdot \mathbf{s} + \mathbf{e} \bmod q$, dove $\mathbf{R}^\mathsf{T} \in \mathbb{Z}_{q^m}^{m \times n}$ è la matrice la cui i-sima colonna è il vettore $\mathbf{r}_i$ contenente le cifre in base q dell'elemento $\mathbf{r}[i]$ ed $\mathbf{e} \in \mathbb{Z}_q^n$ corrisponde al vettore contenente i "riporti" calcolati così come previsto dall'algoritmo elementare di addizione. Un esempio chiarisce immediatamente le idee. Sia $q = 10$, $m = n = 3$, $\mathbf{r} = (738, 916, 375)$ ed $\mathbf{s} = (0, 1, 1)$. Allora $\mathbf{r}^\mathsf{T} \cdot \mathbf{s} \bmod q^m = 916 + 375 \bmod 10^3 = 291$ può essere scritto come:

$$\begin{pmatrix} 7 & 9 & 3 \\ 3 & 1 & 7 \\ 8 & 6 & 5 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 9 \\ 1 \end{pmatrix},$$

dove tutte le operazioni sono eseguite modulo q.

Si può dimostrare che gli elementi del vettore $\mathbf{e}$ sono a distribuzione Gaussiana intorno a $n/2$, con deviazione standard $\sqrt{n}$. Quindi il vettore $\mathbf{e}$ ci ricorda il termine di rumore (anch'esso a distribuzione Gaussiana) nel problema LWE. La differenza sta nel fatto che in LWE il rumore è estratto uniformemente a caso, mentre in SS è univocamente determinato dal valore dei riporti ottenuti effettuando la somma. Il fatto che il vettore $\mathbf{e}$ non sia casuale non ha effetto sulla sicurezza, in quanto Impagliazzo e Naor [IN96] hanno mostrato che distin-

guere $(\mathbf{r}, \mathbf{r}^\mathsf{T} \cdot \mathbf{s} \bmod q^m)$ dalla distribuzione uniforme è equivalente a calcolare $\mathbf{s}$. Ne segue che anche la distribuzione $(\mathbf{R}, \mathbf{R}^\mathsf{T} \cdot \mathbf{s} + \mathbf{e} \bmod q)$ (ovvero la rappresentazione modulo q della distribuzione precedente) è indistinguibile dalla distribuzione uniforme.

Connessione con i reticoli. Mostriamo come un'istanza $(\mathbf{r}, S)$ del problema $\mathrm{SS}(n, M)$ può essere trasformata in un'istanza del problema SVP_γ. Consideriamo il reticolo

$$\mathcal{L} = \left\{ \mathbf{v} \in \mathbb{Z}_M^{n+1} : \ (\mathbf{r} \ || \ -S)^\mathsf{T} \cdot \mathbf{v} \bmod M = 0 \right\}.$$

Osserviamo che il vettore $\mathbf{v} = (\mathbf{s}||1)$ è un elemento di $\mathcal{L}$ quando $\mathbf{s}$ soddisfa $\mathbf{r}^\mathsf{T} \cdot \mathbf{s} \bmod M = S$, così che la norma del vettore più corto in $\mathcal{L}$ è circa $\sqrt{n}$. Il prossimo vettore più corto non parallelo è quello che soddisfa il limite di Hermite (cf. Teorema 9.1) $\sqrt{n+1} \cdot (\det(\mathcal{L}))^{\frac{1}{n+1}} \approx \sqrt{n} M^{1/n}$, che è più grande del vettore più corto di un fattore $M^{1/n}$. Ne segue che risolvere $\mathrm{SVP}_{n^{1.5}}$ consente di risolvere istanze del problema $\mathrm{SS}(n, M)$ con $M \approx n^{1.5}$.

Esercizi

Esercizio 9.1. Sia $\mathcal{L} \subset \mathbb{R}^n$ un reticolo di dimensione n, ed $\mathcal{F}$ un suo dominio fondamentale. Mostrare che ogni vettore $\mathbf{w} \in \mathbb{R}^n$ può essere scritto in un unico modo come $\mathbf{w} = \mathbf{v} + \mathbf{u}$, dove $\mathbf{v} \in \mathcal{L}$ e $\mathbf{u} \in \mathcal{F}$.
(Suggerimento: indicare con $\mathbf{v}_1, \ldots, \mathbf{v}_n$ una base di $\mathcal{L}$. Notare che essa è anche una base di $\mathbb{R}^n$ ed in particolare abbiamo $\mathbf{w} = \sum_i \alpha_i \cdot \mathbf{v}_i$, con $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ non tutti nulli. Scriviamo quindi $\alpha_i = u_i + a_i$, dove $a_i \in \mathbb{Z}$ e $0 \leq u_i \leq 1$...)

Esercizio 9.2. Dimostrare la disuguaglianza di Hadamard. *(Suggerimento: osservare che il volume di $\mathcal{L}$ è massimo quando i vettori della base sono ortogonali.)*

Esercizio 9.3. Mostrare che il teorema di Minkowski implica il Teorema di Hermite. *(Suggerimento: applicare il teorema di Minkowski all'insieme:*

$$\mathcal{S} = \{(x_1, \ldots, x_n) : \ -B \leq x_i \leq B \text{ per ogni } i = 1, \ldots, n\}$$

di volume $\mathrm{Vol}(\mathcal{S}) = (2B)^n$*, avendo posto* $B := \det(\mathcal{L})^{1/n}$*).*

Esercizio 9.4. Consideriamo la seguente versione decisionale del problema della somma di sottoinsiemi. Fissato il modulo M e dati $(\mathbf{r}, S)$, si chiede di determinare se esiste $\mathbf{s}$ tale che $S = \mathbf{r}^{\mathsf{T}} \cdot \mathbf{s} \bmod M$. Mostrare che questo problema è equivalente alla versione computazionale.

Esercizio 9.5. Ricordiamo che un sequenza di interi $r_1, \ldots, r_n$ è detta super-crescente se $r_{i+1} \geq 2 \cdot r_i$ per ogni $i = 1, \ldots, n-1$. Mostrare che tale sequenza soddisfa:

$$r_i > \sum_{j=1}^{i-1} r_j \qquad \text{per ogni } i = 2, \ldots, n.$$

(Suggerimento: usare induzione su n.)

Esercizio 9.6. Sia $(\mathbf{r}, S)$ un'istanza del problema SS senza riduzione modulare (ovvero quando il modulo M non è presente), dove il vettore $\mathbf{r}$ contiene una sequenza super-crescente. Consideriamo il seguente algoritmo:

Per ogni $i = n, \ldots, 1$:

- se $S \geq \mathbf{r}[i]$, poni $\mathbf{s}[i] = 1$ ed aggiorna il valore di S come in $S := S - \mathbf{r}[i]$;
- altrimenti poni $\mathbf{s}[i] = 0$.

Ritorna $\mathbf{s} = (\mathbf{s}[1], \ldots, \mathbf{s}[n])$.

1. Mostrare che l'algoritmo risolve l'istanza $(\mathbf{r}, S)$ data, e che la soluzione è unica. *(Suggerimento: indicare con* $\mathbf{y}$ *la soluzione prodotta dall'algoritmo e mostrare, per induzione su* n*, che* $\mathbf{y}[i] = \mathbf{s}[i]$*.)*

2. Applicare l'algoritmo per risolvere l'istanza $(\mathbf{r} = (3, 11, 24, 50, 115), S = 142)$.

Esercizio 9.7. Questo esercizio descrive il crittosistema di Merkle-Hellman. Alice seleziona una sequenza super-crescente, nel vettore $\mathbf{r} = (\mathbf{r}[1], \ldots, \mathbf{r}[n])$. Inoltre vengono scelti due interi segreti A, B, tali che $B > \mathbf{r}[n]$ e $\gcd(A, B) = 1$. Alice genera la sua chiave pubblica $\hat{\mathbf{r}}$, calcolando $\hat{\mathbf{r}}[i] = A \cdot \mathbf{r}[i] \bmod B$ per ogni $i = 1, \ldots, n$. Dato un messaggio $\mathbf{m} \in \mathbb{Z}_2^n$, il Bianconiglio calcola il crittotesto

$$c = \hat{S} = \hat{\mathbf{r}}^{\mathsf{T}} \cdot \mathbf{m} = \sum_{i=1}^{n} \hat{\mathbf{r}}[i] \cdot \mathbf{m}[i].$$

Per decifrare, Alice calcola dapprima $S = A^{-1} \cdot \hat{S} \bmod B$. Quindi usa l'algoritmo dell'Esercizio 9.6 precedente per risolvere l'istanza $(\mathbf{r}, S)$ del problema SS.

1. Spiegare perché il processo di decifratura restituisce m. *(Suggerimento: usare l'osservazione dimostrata nell'Esercizio 9.5.)*

2. Sia $\mathbf{r} = (3, 11, 24, 50, 115)$ la sequenza super-crescente di Alice, e supponiamo che $A = 113$ e $B = 250$. Cifrare il messaggio $\mathbf{m} = (1, 0, 1, 0, 1)$. Decifrare il corrispondente crittotesto.

3. Attaccare il crittosistema, quando $\mathbf{r}[1] < 2^n$. Mostrare che porre $\mathbf{r}[1] > 2^n$, implica $S, B = O(2^{2n})$.

Esercizio 9.8. Sia $(\mathbf{r}, S)$ un'istanza del problema SS. Per ogni insieme di interi che soddisfa

$$\mathcal{I} \subset \{i : \ 1 \leq i \leq n/2\} \qquad \mathcal{J} \subset \{j : \ n/2 < j \leq n\},$$

definiamo gli interi:

$$A(\mathcal{I}) = \sum_{i \in \mathcal{I}} \mathbf{r}[i] \qquad B(\mathcal{J}) = \sum_{j \in \mathcal{J}} \mathbf{r}[j].$$

1. Mostrare che esistono sempre due insiemi $\mathcal{I}^*, \mathcal{J}^*$ tali che $A(\mathcal{I}^*) = B(\mathcal{J}^*)$ e che
$$S = \sum_{i \in \mathcal{I}^*} \mathbf{r}[i] + \sum_{j \in \mathcal{J}^*} \mathbf{r}[j],$$
ovvero gli insiemi $\mathcal{I}^*, \mathcal{J}^*$ costituiscono una soluzione $\mathbf{s}$ del problema SS di partenza.

2. Calcolare le prestazioni dell'algoritmo in termini di spazio di memoria e di tempo di esecuzione. Fissare il valore del parametro n in modo che si abbia sicurezza 2^{80} nel crittosistema dell'Esercizio 9.7.

Letture consigliate

[ABB10] Shweta Agrawal, Dan Boneh e Xavier Boyen. "Efficient Lattice (H)IBE in the Standard Model". In: *EUROCRYPT*. 2010, pp. 553–572.

[AD97] Miklós Ajtai e Cynthia Dwork. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence". In: *STOC*. 1997, pp. 284–293.

[AGV09] Adi Akavia, Shafi Goldwasser e Vinod Vaikuntanathan. "Simultaneous Hardcore Bits and Cryptography against Memory Attacks". In: *TCC*. 2009, pp. 474–495.

[Ajt96] Miklós Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *STOC*. 1996, pp. 99–108.

[AKS02] Miklós Ajtai, Ravi Kumar e D. Sivakumar. "Sampling Short Lattice Vectors and the Closest Lattice Vector Problem". In: *IEEE Conference on Computational Complexity*. 2002, pp. 53–57.

[App+09] Benny Applebaum, David Cash, Chris Peikert e Amit Sahai. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *CRYPTO*. 2009, pp. 595–618.

[AR04] Dorit Aharonov e Oded Regev. "Lattice Problems in NP ∩ coNP". In: *FOCS*. 2004, pp. 362–371.

[BKW03] Avrim Blum, Adam Kalai e Hal Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *J. ACM* 50.4 (2003), pp. 506–519.

[Cas+10] David Cash, Dennis Hofheinz, Eike Kiltz e Chris Peikert. "Bonsai Trees, or How to Delegate a Lattice Basis". In: *EUROCRYPT*. 2010, pp. 523–552.

[Din+03] Irit Dinur, Guy Kindler, Ran Raz e Shmuel Safra. "Approximating CVP to Within Almost-Polynomial Factors is NP-Hard". In: *Combinatorica* 23.2 (2003), pp. 205–243.

[Dod+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert e Vinod Vaikuntanathan. "Public-Key Encryption Schemes with Auxiliary Inputs". In: *TCC*. 2010, pp. 361–381.

[FP05] Abraham Flaxman e Bartosz Przydatek. "Solving Medium-Density Subset Sum Problems in Expected Polynomial Time". In: *STACS*. 2005, pp. 305–314.

[Fri86] Alan M. Frieze. "On the Lagarias-Odlyzko Algorithm for the Subset Sum Problem". In: *SIAM J. Comput.* 15.2 (1986), pp. 536–539.

[GG98] Oded Goldreich e Shafi Goldwasser. "On the Limits of Non Approximability of Lattice Problems". In: *STOC*. 1998, pp. 1–9.

[GGH96] Oded Goldreich, Shafi Goldwasser e Shai Halevi. “Collision-Free Hashing from Lattice Problems”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 3.42 (1996).

[GGH97] Oded Goldreich, Shafi Goldwasser e Shai Halevi. “Public-Key Cryptosystems from Lattice Reduction Problems”. In: *CRYPTO*. 1997, pp. 112–131.

[Gol+10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert e Vinod Vaikuntanathan. “Robustness of the Learning with Errors Assumption”. In: *ICS*. 2010, pp. 230–240.

[GPV08] Craig Gentry, Chris Peikert e Vinod Vaikuntanathan. “Trapdoors for hard lattices and new cryptographic constructions”. In: *STOC*. 2008, pp. 197–206.

[HB01] Nicholas J. Hopper e Manuel Blum. “Secure Human Identification Protocols”. In: *ASIACRYPT*. 2001, pp. 52–66.

[HPS08] Jeffrey Hoffstein, Jill Pipher e Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. First. Springer, 2008.

[HPS98] Jeffrey Hoffstein, Jill Pipher e Joseph H. Silverman. “NTRU: A Ring-Based Public Key Cryptosystem”. In: *ANTS*. 1998, pp. 267–288.

[HR07] Ishay Haviv e Oded Regev. “Tensor-based hardness of the shortest vector problem to within almost polynomial factors”. In: *STOC*. 2007, pp. 469–477.

[IN96] Russell Impagliazzo e Moni Naor. “Efficient Cryptographic Schemes Provably as Secure as Subset Sum”. In: *J. Cryptology* 9.4 (1996), pp. 199–216.

[Kil+11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain e Daniele Venturi. “Efficient Authentication from Hard Learning Problems”. In: *EUROCRYPT*. 2011, pp. 7–26.

[KTX07] Akinori Kawachi, Keisuke Tanaka e Keita Xagawa. “Multi-bit Cryptosystems Based on Lattice Problems”. In: *Public Key Cryptography*. 2007, pp. 315–329.

[KTX08] Akinori Kawachi, Keisuke Tanaka e Keita Xagawa. “Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems”. In: *ASIACRYPT*. 2008, pp. 372–389.

[LJS90] J. C. Lagarias, Hendrik W. Lenstra Jr. e Claus-Peter Schnorr. “Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice”. In: *Combinatorica* 10.4 (1990), pp. 333–348.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra e László Lovász. "Factoring polynomials with rational coefficients". In: *Mathematische Ann.* 261 (1982), pp. 513–534.

[LM06] Vadim Lyubashevsky e Daniele Micciancio. "Generalized Compact Knapsacks Are Collision Resistant". In: *ICALP (2)*. 2006, pp. 144–155.

[LM08] Vadim Lyubashevsky e Daniele Micciancio. "Asymptotically Efficient Lattice-Based Digital Signatures". In: *TCC*. 2008, pp. 37–54.

[LO85] J. C. Lagarias e Andrew M. Odlyzko. "Solving Low-Density Subset Sum Problems". In: *J. ACM* 32.1 (1985), pp. 229–246.

[LPR10] Vadim Lyubashevsky, Chris Peikert e Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *EUROCRYPT*. 2010, pp. 1–23.

[LPS10] Vadim Lyubashevsky, Adriana Palacio e Gil Segev. "Public-Key Cryptographic Primitives Provably as Secure as Subset Sum". In: *TCC*. 2010, pp. 382–400.

[Lyu+08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert e Alon Rosen. "SWIFFT: A Modest Proposal for FFT Hashing". In: *FSE*. 2008, pp. 54–72.

[Lyu05] Vadim Lyubashevsky. "The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem". In: *APPROX-RANDOM*. 2005, pp. 378–389.

[Lyu08] Vadim Lyubashevsky. "Lattice-Based Identification Schemes Secure Under Active Attacks". In: *Public Key Cryptography*. 2008, pp. 162–179.

[Lyu09] Vadim Lyubashevsky. "Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures". In: *ASIACRYPT*. 2009, pp. 598–616.

[MH78] Ralph C. Merkle e Martin E. Hellman. "Hiding Information and Signatures in Trapdoor Knapsacks". In: *IEEE Transactions on Information Theory* 24 (set. 1978), pp. 525–530.

[Mic02] Daniele Micciancio. "Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions from Worst-Case Complexity Assumptions". In: *FOCS*. 2002, pp. 356–365.

[MR04] Daniele Micciancio e Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *FOCS*. 2004, pp. 372–381.

[MV03] Daniele Micciancio e Salil P. Vadhan. "Statistical Zero-Knowledge Proofs with Efficient Provers: Lattice Problems and More". In: *CRYPTO*. 2003, pp. 282–298.

[Ngu06] Phong Q. Nguyen. *A Note on the Security of NTRUSign*. Cryptology ePrint Archive, Report 2006/387. `http://eprint.iacr.org/`. 2006.

[NS98] Phong Q. Nguyen e Jacques Stern. "Cryptanalysis of the Ajtai-Dwork Cryptosystem". In: *CRYPTO*. 1998, pp. 223–242.

[Odl90] Andrew M. Odlyzko. "The Rise and Fall of the Knapsack Problem". In: *Proc. of the AMS Symposia in Applied Mathematics: Computational Number Theory and Cryptography*. American Mathematical Society, 1990, pp. 75–88.

[Pei09] Chris Peikert. "Public-key cryptosystems from the worst-case shortest vector problem: extended abstract". In: *STOC*. 2009, pp. 333–342.

[PR06] Chris Peikert e Alon Rosen. "Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices". In: *TCC*. 2006, pp. 145–166.

[PVW08] Chris Peikert, Vinod Vaikuntanathan e Brent Waters. "A Framework for Efficient and Composable Oblivious Transfer". In: *CRYPTO*. 2008, pp. 554–571.

[PW08] Chris Peikert e Brent Waters. "Lossy trapdoor functions and their applications". In: *STOC*. 2008, pp. 187–196.

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *STOC*. 2005, pp. 84–93.

[Reg10] Oded Regev. *The Learning with Errors Problem*. Invited survey in CCC. `http://www.cs.tau.ac.il/~odedr/`. 2010.

[Sch87] Claus-Peter Schnorr. "A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms". In: *Theor. Comput. Sci.* 53 (1987), pp. 201–224.

[Sha08] Andrew Shallue. "An Improved Multi-set Algorithm for the Dense Subset Sum Problem". In: *ANTS*. 2008, pp. 416–429.

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509.

[Ste+09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka e Keita Xagawa. "Efficient Public Key Encryption Based on Ideal Lattices". In: *ASIACRYPT*. 2009, pp. 617–635.

CAPITOLO 10

Crittografia su base identità

Inutile che ficchino la testa qua sotto e dicano: « Torna su tesoro!». Io non farò altro che guardare in su e dire: «Chi sono io allora? Ditemi prima questo, e poi, se mi andrà di essere quella persona, verrò su; se no, resterò quaggiù finché non sarò qualchedun'altra.»

Lewis Carroll, Le Avventure di Alice nel Paese delle Meraviglie [Car65]

Abbiamo studiato due insiemi di tecniche per la progettazione di sistemi sicuri: tecniche simmetriche e tecniche asimmetriche. Nel contesto delle tecniche simmetriche si assume che le parti coinvolte nel sistema condividano una chiave segreta; pertanto il problema principale è quello di distribuire le chiavi in modo sicuro. (Ci occuperemo di questo, in parte, nel Paragrafo 11.1.) L'uso di tecniche asimmetriche evita a priori questa problematica: Alice possiede una coppia di chiavi (una pubblica e l'altra privata) e se il Bianconiglio vuole inviarle un messaggio, necessita solamente della chiave *pubblica* di Alice. Sorge un problema di "fiducia":

> *Come può il Bianconiglio essere sicuro che la chiave pubblica di Alice sia "autentica", ovvero corrisponda effettivamente alla chiave segreta di Alice?*

Rispondere a questa domanda è fondamentale, altrimenti la Regina Rossa potrebbe semplicemente scambiare la sua chiave pubblica con la chiave pubblica di Alice, così da poter, ad esempio, decifrare tutti i messaggi indirizzati a questa. In questo capitolo studieremo essenzialmente due soluzioni al dilemma dell'autenticità delle chiavi pubbliche. La prima soluzione, detta su base *infrastruttura a chiave pubblica*, prevede di "certificare" l'autenticità delle chiavi pubbliche attraverso le firme digitali. La seconda soluzione, detta su base *identità*, prevede di "incorporare" in qualche modo l'identità di Alice nella sua chiave pubblica; in questo modo l'appartenenza di una chiave pubblica ad uno specifico utente è inequivocabile.

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_10,

Guida per il lettore. La struttura del capitolo è la seguente. Introdurremo il concetto di infrastruttura a chiave pubblica nel Paragrafo 10.1 ed i crittosistemi su base identità nel Paragrafo 10.2. Nel Paragrafo 10.3 parleremo degli accoppiamenti bilineari che saranno quindi utilizzati, nel Paragrafo 10.4, per costruire un cifrario su base identità CPA-sicuro (nel modello dell'oracolo casuale). Non studieremo altre primitive su base identità, pertanto il capitolo si conclude con una breve panoramica dei risultati noti in quest'area (cf. Paragrafo 10.5).

10.1 Infrastrutture a chiave pubblica

La prima soluzione al problema dell'autenticità delle chiavi pubbliche risiede nel concetto di infrastruttura a chiave pubblica (*Public Key Infrastructure*, PKI). L'idea di base consiste nell'istituzione di una terza parte fidata detta autorità di certificazione (*Certification Authority*, CA) che si occupi di certificare l'autenticità delle chiavi pubbliche degli utenti. Purtroppo non è possibile contare su una singola CA, essenzialmente perché è impossibile pensare che esista un'entità di cui *tutti* si fidano *indipendentemente dal contesto* (e che sia utilizzabile in ogni contesto). Inoltre, una soluzione con una singola CA sarebbe comunque poco scalabile. È bene tener presente che la messa in campo di una PKI, in generale, dipende dalla specifica applicazione crittografica e dal contesto in cui essa è utilizzata.

Certificati e catene di certificati. Per semplicità iniziamo a descrivere il caso in cui esista una singola CA di cui tutti si fidano. Tipicamente una CA non è una singola persona, ma ad esempio un'agenzia di governo. La CA ha associate una coppia di chiavi pubblica/privata (pk_{CA}, sk_{CA}). Chiunque voglia usufruire dei servizi offerti dalla CA deve possedere una copia *autentica* della chiave pubblica pk_{CA}. A questo punto la situazione può sembrare paradossale: per risolvere il problema dell'autenticità delle chiavi pubbliche abbiamo bisogno di credere che pk_{CA} sia autentica! In realtà, possiamo dire di aver isolato il problema: l'unica cosa di cui bisogna preoccuparsi ora è della distribuzione di pk_{CA}. Questo in pratica significa che la chiave pubblica della CA deve essere distribuita attraverso un *canale autenticato*; qualora la CA fosse una piccola azienda la chiave pk_{CA} potrebbe essere consegnata fisicamente, ad esempio per mezzo di un supporto multimediale come un CD-ROM. Nei sistemi operativi odierni, invece, la chiave pubblica della CA è contenuta direttamente

nel browser web ed è usata per verificare i certificati provenienti dai siti web visitati.

Supponiamo ora che Alice voglia usufruire della PKI. Per far ciò è sufficiente che Alice generi la sua coppia di chiavi pubblica/privata (pk_A, sk_A) ed inoltri alla CA la sua identità (nel seguito rappresentata dalla stringa `Alice`) insieme alla sua chiave pubblica pk_A. La CA, dopo aver controllato l'identità di Alice, certifica l'autenticità della sua chiave pubblica attraverso uno schema di firma digitale $\Pi_{\mathsf{SGN}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, producendo il certificato:

$$\mathsf{cert}_{CA \to A} = \mathsf{Sign}_{sk_{CA}}(pk_A, \texttt{Alice}).$$

(Come vedremo, tipicamente, il messaggio che viene firmato contiene anche altre informazioni.) Ovviamente il requisito fondamentale è che non sia possibile forgiare una firma senza conoscere la chiave segreta sk_{CA}, ovvero Π_{SGN} deve essere universalmente inforgiabile contro attacchi a messaggio scelto (cf. Definizione 8.2). Quando Alice vuole comunicare con il Bianconiglio, invia il certificato $\mathsf{cert}_{CA \to A}$; il Bianconiglio (che come tutti si fida della CA) usa pk_{CA} per verificare la firma ed eventualmente accetta pk_A come chiave pubblica *autentica* di Alice. La struttura completa di un certificato è standardizzata dall'ITU [ITU88].

Come anticipato, ci sono diversi problemi relativi all'uso di una singola CA. Il primo di essi è che risulta difficile credere esista un'entità di cui tutti si fidano. Inoltre la (singola) CA sarebbe un singolo punto di fallimento dell'intero sistema: se essa fosse compromessa, l'intera infrastruttura crollerebbe. Infine la CA dovrebbe essere sempre disponibile. Per evitare questi problemi si possono utilizzare CA multiple. Alice può avere ora diversi certificati (generati da diverse CA) relativi alla sua chiave pubblica ed il Bianconiglio può scegliere la CA di cui si fida di più per verificare l'autenticità di un dato certificato. In questo modo si crea una gerarchia di CA.

Per alleggerire il compito di una singola CA, si possono impiegare *catene di certificati*. Supponiamo che una CA crei un certificato $\mathsf{cert}_{CA \to B}$ che attesti l'autenticità della chiave pubblica del Bianconiglio. In questo caso gli utenti possono generare i propri certificati autonomamente. Ad esempio, il Bianconiglio può usare la sua coppia di chiavi pubblica/privata (pk_B, sk_B) per autenticare la chiave pubblica di Alice, calcolando:

$$\mathsf{cert}_{B \to A} = \mathsf{Sign}_{sk_B}(pk_A, \texttt{Alice}).$$

Se ora Alice vuole comunicare con il Cappellaio Matto, che conosce la chiave

pubblica della CA (ma non quella del Bianconiglio), Alice può inviare:

$$pk_A, \mathsf{cert}_{B\to A}, pk_B, \mathsf{cert}_{CA\to B};$$

in questo modo il Cappellaio Matto può verificare l'autenticità della chiave pubblica del Bianconiglio e quindi usare pk_B per verificare il certificato $\mathsf{cert}_{B\to A}$ ed autenticare la chiave pubblica di Alice. Abbiamo così "propagato la fiducia". In generale, possiamo pensare di avere una singola CA di primo livello ed n CA di secondo livello. La CA madre crea certificati per ogni CA di secondo livello, le quali a loro volta possono creare certificati per le chiavi degli utenti. In questo modo le parti comunicanti possono far riferimento alla CA di secondo livello che è a loro più vicina.

Modello su base "ragnatela di fiducia". Possiamo anche pensare ad un modello pienamente distribuito, senza alcuna terza parte fidata. (Una variante di tale modello è usata ad esempio nel sistema di cifratura per la posta elettronica PGP, ideato da Zimmerman nel 1991 e standardizzato dall'IETF [IET07].) In pratica Alice può generare certificati per autenticare, ad esempio, la chiave pubblica del Bianconiglio; chi riceve un dato certificato deve decidere se validarlo o meno sulla base della fiducia che ha nell'utente che ha inviato il certificato. Vediamo un esempio concreto. Supponiamo che Alice possegga le chiavi pubbliche autentiche pk_D, pk_E e pk_F relative agli utenti D, E ed F. Supponiamo che il Bianconiglio per comunicare con Alice invii

$$pk_B, \mathsf{cert}_{D\to B}, \mathsf{cert}_{F\to B}, \mathsf{cert}_{G\to B}.$$

Alice non può verificare $\mathsf{cert}_{G\to B}$ in quanto non conosce la chiave pubblica di G. Deve quindi decidere se accettare pk_B solo sulla base della fiducia che ha in D ed F. Ad esempio se si fida ciecamente di D può accettare pk_B come autentica. Oppure potrebbe accettarla perché ritiene improbabile che entrambi D ed F siano corrotti. In generale, quindi, l'idea è quella di costruire un elenco centralizzato contenente chiavi pubbliche e relativi certificati: se Alice vuole comunicare con il Bianconiglio, può scaricare dall'elenco la chiave pubblica del Bianconiglio e tutti i certificati che ne attestano l'autenticità. Alice deciderà quindi se accettare la chiave del Bianconiglio come valida, sulla base della fiducia che pone negli utenti che hanno generato tali certificati.

Gestione dei certificati. Restano da discutere una serie di questioni relative alla gestione dei certificati. Ad esempio i certificati non possono avere validità

illimitata (perché Alice potrebbe smarrire la sua chiave segreta oppure cambiare azienda). Una prima soluzione è quella di associare una data (diciamo nella forma `gg/mm/aaaa`) di scadenza ad ogni certificato. Quindi il certificato è generato come:

$$\mathsf{cert}_{B\to A} = \mathsf{Sign}_{sk_B}(pk_A, \texttt{Alice}, \texttt{gg/mm/aaaa}).$$

In questo modo chi verifica il certificato non solo deve fidarsi di B, ma deve anche controllare che questo non sia scaduto. Purtroppo questa è solo una soluzione parziale al problema, in quanto c'è un evidente compromesso tra la frequenza con cui un certificato va rinnovato e l'affidabilità della soluzione. Se, ad esempio, un certificato generato da un'azienda ha validità un anno ed un impiegato si licenzia il secondo giorno di lavoro, tale impiegato può usare la sua chiave pubblica in modo illegittimo ancora per 364 giorni!

In casi come questo — oppure quando una chiave segreta viene rubata — vorremmo prendere un provvedimento con azione immediata: vorremmo cioè avere la possibilità di *revocare* un certificato. Un modo di fare ciò è quello di associare un numero di serie *univoco* ad ogni certificato. Se ora la chiave segreta di Alice viene rubata, Alice può avvertire la CA di competenza. La CA provvederà prima di tutto a verificare l'identità di Alice — così che nessuno al di fuori di Alice possa revocare i certificati relativi alla chiave pubblica di Alice — e quindi ad inserire in una lista di certificati revocati (*Certificate Revocation List*, CRL) tutti i numeri di serie relativi a certificati che autenticano la chiave pubblica di Alice. Tale CRL deve essere firmata dalla CA e resa pubblica. In questo modo un utente che riceve un certificato deve innanzitutto verificare che questo non sia stato revocato. La revoca dei certificati è il problema più spinoso nella gestione di una PKI.

Sicurezza dimostrabile. Le politiche e le procedure riguardanti una PKI cambiano di continuo e spesso sono accompagnate da descrizioni lunghe e tediose [AF99]. Inoltre restano una serie di domande senza risposta precisa. Cos'è esattamente una CA? Si può ottenere un livello di sicurezza accettabile anche quando una CA è corrotta o si comporta in modo malizioso? È possibile dimostrare formalmente che una primitiva crittografica con un qualche grado di sicurezza dimostrabile è sicura anche quando viene calata nel contesto delle PKI? Si veda [Bol+07] per una risposta formale a questi interrogativi.

10.2 Un'alternativa alle PKI

Come accennato all'inizio del capitolo, un'alternativa all'uso delle PKI è quella di "incorporare" l'identità di ciascun utente nella sua chiave pubblica. L'idea seminale della crittografia su base identità è dovuta a Shamir [Sha84] e risale al 1985. Come realizzare questa idea è stato un problema aperto per quasi vent'anni: solo nel 2001 Boneh e Franklin [BF01] hanno mostrato come realizzare l'idea di Shamir (nel modello dell'oracolo casuale).

L'idea principale è quella di semplificare il processo di autenticazione delle chiavi pubbliche facendo in modo che la chiave pubblica di Alice sia direttamente la sua "identità digitale". A seconda delle applicazioni tale "identità digitale" può assumere la forma di un indirizzo di posta elettronica oppure di un numero di telefono. In questo modo la chiave pubblica di Alice non necessita di essere preventivamente autenticata. Di conseguenza non sono necessari certificati, la cui gestione (come abbiamo visto) è l'aspetto più complesso nelle PKI. È bene sottolineare sin da ora che la realizzazione concreta del collegamento tra utente ed identità digitale è tutt'altro che banale.

Siccome la chiave pubblica di Alice *non* è ricavata dalla chiave segreta, ma è rappresentata di fatto dalla sua identità, segue che, al contrario di come succede nei sistemi basati sulle PKI, la chiave segreta di Alice deve essere derivata dalla sua identità. Non possiamo però pensare che Alice determini la sua chiave segreta da sola, altrimenti la Regina sarebbe chiaramente in grado di fare lo stesso! Per questo motivo si introduce una terza parte detta centro di generazione delle chiavi (*Key Generation Center*, KGC) il cui scopo è esattamente quello di generare la chiave segreta di un utente. Per fare ciò il KGC usa una chiave segreta principale msk ed alcuni parametri pubblici $ppub$ noti a tutti gli utenti del sistema. Il risultato è la chiave segreta sk_u relativa all'utente con identità u; questi a sua volta potrà usare la chiave segreta per le normali operazioni previste dal crittosistema.

Cifrari. Ci occuperemo solo di cifratura e non di firme digitali, sebbene esistano anche schemi di firma digitale su base identità (cf. Paragrafo 10.5 per alcuni cenni). Iniziamo con la definizione formale di un cifrario su base identità.

Definizione 10.1 (Cifrario su base identità). Indichiamo con $\mathcal{M}$ lo spazio dei possibili testi in chiaro, con $\mathcal{C}$ lo spazio dei testi cifrati, con $\mathcal{U}$ l'insieme di tutte le identità degli utenti e con $\mathcal{K}$ lo spazio delle rispettive chiavi segrete. Un cifrario su base identità è un insieme di algoritmi $\Pi_{\mathsf{IBE}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ definita come segue:

- `Inizializzazione del sistema.` Dato il parametro di sicurezza n, l'algoritmo di inizializzazione Setup è usato per generare la chiave segreta principale $msk \in \{0,1\}^*$ del KGC insieme ai parametri pubblici del sistema $ppub$, i.e. $(msk, ppub) \leftarrow \mathsf{Setup}(1^n)$.
- `Generazioni delle chiavi.` Data la chiave segreta principale msk e l'identità u di un certo utente, l'algoritmo di generazione delle chiavi $\mathsf{Gen} : \{0,1\}^* \times \mathcal{U} \to \mathcal{K}$ restituisce la chiave segreta $sk_u \leftarrow \mathsf{Gen}_{msk}(u)$ corrispondente all'utente con identità u.
- `Cifratura.` Dato un messaggio $m \in \mathcal{M}$ l'algoritmo $\mathsf{Enc} : \mathcal{U} \times \{0,1\}^* \times \mathcal{M} \to \mathcal{C}$ usa i parametri pubblici $ppub$ e l'identità $u \in \mathcal{U}$ del ricevente per generare il crittotesto $c \leftarrow \mathsf{Enc}_u(ppub, m)$.
- `Decifratura.` Dato un crittotesto $c \in \mathcal{C}$ destinato all'utente con identità u, l'algoritmo $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ usa la chiave segreta sk_u per recuperare $m = \mathsf{Dec}_{sk_u}(c)$. ■

La sicurezza è definita in modo simile a quanto visto per i crittosistemi a chiave pubblica (cf. Definizione 6.2). Consideriamo il seguente esperimento.

Esperimento $\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{IBE},\Pi_{\mathsf{IBE}}}(\mathscr{A}, n)$:
1. $(msk, ppub) \leftarrow \mathsf{Setup}(1^n)$;
2. l'attaccante può richiedere le chiavi segrete sk_{u_i} corrispondenti ad un numero polinomiale di identità u_i;
3. $(m_0, m_1, u^*) \leftarrow \mathscr{A}(ppub)$, dove $m_0, m_1 \xleftarrow{\$} \mathcal{M}$ ed u^* è distinta dalle identità richieste al passo precedente;
4. $c_b \leftarrow \mathsf{Enc}_{u^*}(ppub, m_b)$, con $b \xleftarrow{\$} \{0,1\}$;
5. l'attaccante può richiedere chiavi segrete corrispondenti ad altre identità distinte da u^*. Sceglie infine un bit b';
6. restituisci 1 se e solo se $b' = b$ e $|m_0| = |m_1|$.

Definizione 10.2 (Sicurezza CPA di un cifrario su base identità). Diremo che il cifrario Π_{IBE} è CPA-(t, Q, ϵ)-sicuro se, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t e che richiede Q identità, risulta:

$$\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cpa}}_{\mathsf{IBE},\Pi_{\mathsf{IBE}}}(\mathscr{A}, n) = 1\right] \leq \frac{1}{2} + \epsilon.$$

■

Esiste anche una definizione di sicurezza CCA, che però non discuteremo (cf. il Paragrafo 10.5 per qualche cenno).

Paragone con le PKI. Ci sono diversi aspetti da discutere affinché un crittosistema su base identità possa essere utilizzato in pratica. Osserviamo che tali sistemi ipotizzano che il collegamento tra Alice e la sua identità digitale sia *non ambiguo*. Realizzare questo collegamento non è banale.[63] Per questo è necessaria la presenza di un'entità a parte detta autorità di registrazione (*Registration Authority*, RA) che si occupi di assegnare le identità digitali agli utenti in modo non ambiguo.

Concludiamo il paragrafo con un paragone tra crittografia su base identità e PKI, evidenziando vantaggi e svantaggi dei due approcci.

- *Autenticità dei parametri pubblici.* L'autenticità dei parametri pubblici deve essere assicurata in qualche modo, altrimenti un attaccante potrebbe creare dei parametri pubblici "fasulli" ed ingannare gli utenti del sistema facendo credere loro che tali parametri sono genuini. Ciò consentirebbe all'attaccante di derivare tutte le chiavi segrete corrispondenti alle identità degli utenti e quindi di decifrare ogni messaggio cifrato con tali chiavi (o di forgiare firme arbitrarie con le stesse chiavi).

  Questo problema è presente anche nelle PKI, dove bisogna assicurare l'autenticità della chiave pubblica delle CA.

- *Registrazione.* In entrambi i sistemi è necessario che Alice si registri presso una qualche autorità di registrazione. (Tale entità può essere co-locata con il KGC.)

- *"Estrazione" della chiave.* Il KGC ha tutti gli elementi per estrarre la chiave segreta di un dato utente. Sebbene ciò sia ragionevole in alcuni casi (ad esempio se il direttore di un'azienda vuole poter controllare i messaggi scambiati dai suoi impiegati) è del tutto inaccettabile in altri: nel contesto delle firme digitali, ad esempio, ciò preclude immediatamente la possibilità di ottenere non ripudio.

  In un sistema su base PKI in cui gli utenti generano da soli la loro coppia di chiavi pubblica/privata (che quindi non sono memorizzate dalle CA) questo problema è assente. Un modo per mitigare questo problema può essere quello di replicare il KGC in $N \geq 2$ KGC: quando Alice richiede la sua chiave segreta, riceve una "chiave parziale" da ogni KGC. La chiave segreta può essere ottenuta mettendo insieme tutte le chiavi parziali così generate. In questo modo l'estrazione della chiave di Alice richiede che tutti i KGC colludano.

[63] Ad esempio non possiamo pensare di usare nome e cognome, in quanto esistono casi di omonimia.

- *Gestione delle chiavi.* Quando un certificato viene revocato in un sistema basato sulle PKI, gli utenti vengono notificati di ciò attraverso una lista di certificati revocati (cf. Paragrafo 10.1). Se ad un certo punto Alice vuole generare una nuova coppia di chiavi può semplicemente ottenere un nuovo certificato per esse. La situazione è più problematica in un sistema su base identità, in quanto l'identità digitale di Alice è (e deve essere) in qualche modo "univoca". Per affrontare questo problema, solitamente, si includono informazioni aggiuntive all'identità di un utente, in modo da renderla più granulare.[64]

  La distribuzione delle chiavi pubbliche degli utenti è il vantaggio principale dei sistemi su base identità: la chiave pubblica coincide di fatto con l'identità stessa. In un sistema PKI il Bianconiglio deve controllare il certificato corrispondente alla chiave pubblica di Alice, verificare la firma in esso contenuta e controllare che il certificato non sia scaduto. D'altra parte, quando Alice vuole estrarre la sua chiave segreta deve comunicare con il KGC ed è necessario prevedere un canale sicuro tra Alice e KGC per trasportare tale chiave in modo sicuro. Inoltre la chiave principale del KGC deve essere protetta, in quanto il suo smarrimento comprometterebbe la sicurezza dell'intero sistema.

In conclusione, i crittosistemi su base identità costituiscono un'alternativa valida alle PKI con vantaggi e svantaggi. La scelta della soluzione da adottare, in generale, dipende dal contesto.

10.3 Accoppiamenti bilineari

Per poter descrivere il primo cifrario su base identità, abbiamo bisogno del concetto di *gruppo bilineare*. Un gruppo bilineare è tale se possiede un *accoppiamento bilineare*. Siccome la matematica necessaria alla costruzione esplicita degli accoppiamenti è alquanto non elementare (in particolare richiede concetti non banali di geometria algebrica), ci limiteremo ad usare gli accoppiamenti come oggetti astratti, senza costruirli in modo esplicito. Comunque gli accoppiamenti possono essere costruiti in modo efficiente, esempi concreti sono l'accoppiamento di Weil e quello di Tate-Lichtenbaum [Sil86, Capitolo 3]. Si veda anche [Maa04] per una buona introduzione.

[64] Ad esempio si può aggiungere la data oppure l'anno. Ritroviamo qui il compromesso tra efficienza e granularità già incontrato in precedenza: se l'informazione è troppo granulare l'identità deve essere rinnovata molto frequentemente, a discapito dell'efficienza.

Definizione 10.3 (Accoppiamento bilineare). Sia $\mathbb{G}$ un gruppo moltiplicativo di ordine primo p e con generatore g. La mappa $\Xi : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ è una mappa bilineare se soddisfa quanto segue:

- *Efficiente.* La funzione $\Xi(\cdot, \cdot)$ è calcolabile efficientemente.
- *Non degenere.* Per ogni elemento $g_1, g_2 \in \mathbb{G}$ non nullo, $\Xi(g_1, g_2)$ genera $\mathbb{G}_T$.
- *Bilineare.* Per ogni $a, b \in \mathbb{Z}_p$ risulta $\Xi(g^a, g^b) = \Xi(g, g)^{a \cdot b}$, dove $g^a, g^b \in \mathbb{G}$ e $\Xi(g^a, g^b) \in \mathbb{G}_T$.

Il gruppo $\mathbb{G}_T$ è detto *gruppo obiettivo*. Solitamente $\mathbb{G}$ è il gruppo di una curva ellittica (cf. Appendice B.4) mentre $\mathbb{G}_T$ è un gruppo finito. ■

Problemi difficili. Nel paragrafo 6.3 abbiamo introdotto l'ipotesi computazionale/decisionale di Diffie-Hellman (CDH e DDH rispettivamente). Il problema CDH richiede di calcolare g^{ab} dati g e $g^a, g^b \in \mathbb{G}$. Il problema DDH, invece, dati g, g^a, g^b ed un elemento $\zeta = g^c \in \mathbb{G}$ richiede di stabilire se $\zeta = g^{ab}$ (ovvero se $c = a \cdot b$) oppure se $\zeta \xleftarrow{\$} \mathbb{G}$ è un elemento casuale in $\mathbb{G}$.

Come sappiamo esistono gruppi in cui i problemi CDH e DDH sono ritenuti difficili (ovvero non risolvibili in tempo polinomiale). Purtroppo, questo non è il caso dei gruppi bilineari:

Lemma 10.1 (CDH e DDH in gruppi bilineari). *Se $\mathbb{G}$ è un gruppo bilineare, allora il problema del logaritmo discreto in $\mathbb{G}$ non è più difficile del problema del logaritmo discreto in $\mathbb{G}_T$. Inoltre il problema DDH è risolvibile efficientemente in $\mathbb{G}_T$.*

Dimostrazione. La prima affermazione equivale a dire che se possiamo risolvere il logaritmo discreto in $\mathbb{G}_T$, allora possiamo risolverlo anche in $\mathbb{G}$. Sia infatti $\alpha = g^a \in \mathbb{G}$ e supponiamo di voler calcolare il logaritmo discreto in base g di α (ovvero a). Calcoliamo prima $\xi_1 = \Xi(g, g)$ e quindi $\xi_2 = \Xi(g, g^a) = \Xi(g, g)^a = \xi_1^a$ (il che è fattibile in tempo polinomiale siccome $\Xi(\cdot, \cdot)$ è una mappa efficiente). Se il problema del logaritmo discreto è risolvibile in $\mathbb{G}_T$, allora sappiamo calcolare $a = \log_{\xi_1}(\xi_2)$, risolvendo così il problema del logaritmo discreto in $\mathbb{G}$.

D'altra parte, consideriamo un'istanza $(g, g^a, g^b, \zeta = g^c)$ del problema DDH. Calcoliamo $\xi_1 = \Xi(g^a, g^b)$ e $\xi_2 = \Xi(g, \zeta)$. Se $\xi_1 = \xi_2$ possiamo concludere che $\zeta = g^{ab}$, infatti quando questo è il caso si ha

$$\xi_1 = \Xi(g^a, g^b) = \Xi(g, g)^{ab} = \Xi(g, g^{ab}) = \Xi(g, \zeta) = \xi_2.$$

Siccome la mappa $\Xi(\cdot,\cdot)$ è non-degenere ciò implica immediatamente $c = a \cdot b$. Quando invece $\zeta \xleftarrow{\$} \mathbb{G}$, avremo $\xi_1 \neq \xi_2$ con alta probabilità. □

Si è soliti allora introdurre un altro problema computazionale, detto problema (decisionale) bilineare di Diffie-Hellman (*Bilinear Decisional Diffie-Hellman problem*, BDDH). Sia $\mathbb{G}$ un gruppo bilineare con ordine un primo p, generatore g e gruppo obiettivo $\mathbb{G}_T$. Dati $g^a, g^b, g^c \xleftarrow{\$} \mathbb{G}$, si richiede di distinguere $\xi = \Xi(g,g)^{abc}$ da $\xi \xleftarrow{\$} \mathbb{G}_T$.

Definizione 10.4 (Difficoltà del problema BDDH). Diremo che il problema decisionale bilineare di Diffie-Hellman (BDDH) è (t,ϵ)-difficile se, per ogni attaccante PPT $\mathscr{D}$ eseguibile in tempo t, risulta:

$$\Big| \mathbb{P}\left[\mathscr{D}\left(1^n, \mathbb{G}, \mathbb{G}_T, p, g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi = \Xi(g,g)^{abc}\right] - \mathbb{P}\left[\mathscr{D}\left(1^n, \mathbb{G}, \mathbb{G}_T, p, g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi \xleftarrow{\$} \mathbb{G}_T\right] \Big| \leq \epsilon(n).$$

■

10.4 Lo schema di Boneh e Franklin

La prima costruzione esplicita di cifrario su base identità è dovuta a Boneh e Franklin[BF01], i quali hanno usato l'accoppiamento di Weil e l'ipotesi BDDH per dimostrare la sicurezza del loro crittosistema.[65]

Lo schema di Boneh e Franklin, Π_{BF}, è mostrato nel Crittosistema 10.1. Non è difficile accorgersi che tale schema è una traduzione del cifrario di ElGamal (cf. Paragrafo 6.3) su gruppi bilineari. Gli spazi dei testi in chiaro $\mathcal{M}$ e dei testi cifrati $\mathcal{C}$ coincidono con il gruppo obiettivo $\mathbb{G}_T$. Verifichiamo che l'operazione di decifratura inverte effettivamente quella di cifratura. L'osservazione chiave è che $\Xi(sk_u, c_1) = \Xi(H(u)^a, g^\omega) = \Xi(H(u), g)^{a\cdot\omega} = \xi$ e quindi

$$\frac{c_2}{\Xi(sk_u, c_1)} = \frac{\xi \cdot m}{\xi} = m,$$

come desiderato. La prova di sicurezza fornita da Boneh e Franklin ipotizza che la funzione $H(\cdot)$ sia un oracolo casuale (cf. Paragrafo 4.4).

[65] Nello stesso anno, in modo del tutto indipendente, Cocks [Coc01] ha presentato una costruzione basata sul problema della residuosità quadratica (cf. Paragrafo 6.4).

Crittosistema 10.1. Il cifrario su base identità di Boneh e Franklin

Sia $\mathbb{G}$ un gruppo bilineare con ordine un primo p, generatore g e gruppo obiettivo $\mathbb{G}_T$. Sia inoltre $H : \{0,1\}^* \to \mathbb{G}$ una funzione hash. Posto $\mathcal{U} = \{0,1\}^*$, $\mathcal{M} = \mathcal{C} = \mathbb{G}_T$ e $\mathcal{K} = \mathbb{Z}_p$, il cifrario $\Pi_{\mathsf{BF}} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ è definito come segue:

- `Inizializzazione del sistema.` Dato il parametro di sicurezza n genera $(ppub, a) \leftarrow \mathsf{Setup}(1^n)$, dove $ppub = (\mathbb{G}, \mathbb{G}_T, g, g^a, H)$ e $g^a \xleftarrow{\$} \mathbb{G}$. La chiave segreta principale è $msk = a$.
- `Generazione delle chiavi.` Dato un identificativo $u \in \{0,1\}^*$, calcola la corrispondente chiave segreta $sk_u = \mathsf{Gen}_{msk}(u, ppub) = H(u)^a$.
- `Cifratura.` Dato un messaggio $m \in \mathbb{G}_T$ da cifrare (diretto ad un utente con identità $u \in \{0,1\}^*$), estrai $\omega \xleftarrow{\$} \mathbb{Z}_p$ e calcola $\xi = \Xi(H(u), g^a)^\omega = \Xi(H(u), g)^{a \cdot \omega}$. Restituisci il crittotesto:

$$c \leftarrow \mathsf{Enc}_u(m) = (g^\omega, m \cdot \xi) \in \mathbb{G}_T.$$

- `Decifratura.` Dato un testo cifrato $c = (c_1, c_2)$, calcola:

$$m = \mathsf{Dec}_{sk_u}(c) = \frac{c_2}{\Xi(sk_u, c_1)} \in \mathbb{G}_T.$$

Teorema 10.2 (Π_{BF} è CPA-sicuro nel modello dell'oracolo casuale). *Se il problema BDDH è (t, ϵ)-difficile, il cifrario di Boneh e Franklin Π_{BF} è CPA-$(t_{\mathsf{BF}}, Q, \epsilon_{\mathsf{BF}})$-sicuro nel modello dell'oracolo casuale, per ogni $Q = \mathrm{poly}(n)$ e con*

$$t_{\mathsf{BF}} \approx t \qquad \epsilon = \epsilon_{\mathsf{BF}}/Q.$$

Dimostrazione. Dato un attaccante PPT $\mathscr{A}$ eseguibile in tempo t_{BF} in grado di violare la sicurezza CPA del cifrario con probabilità maggiore di $1/2 + \epsilon_{\mathsf{BF}}$ (come nell'enunciato del teorema), mostriamo come costruire un attaccante PPT $\mathscr{D}$ eseguibile in tempo $t \approx t_{\mathsf{BF}}$ in grado di risolvere il problema BDDH con vantaggio ϵ, contro l'ipotesi che il problema BDDH sia (t, ϵ)-difficile. L'attaccante $\mathscr{D}$ riceve come input (g, g^a, g^b, g^c, ξ) e deve decidere se $\xi = \Xi(g, g)^{abc}$ oppure se $\xi \xleftarrow{\$} \mathbb{G}_T$. Per fare ciò $\mathscr{D}$ simula l'"ambiente" per $\mathscr{A}$ come segue:

1. Poni $ppub = (\mathbb{G}, \mathbb{G}_T, g, g^a, H)$, dove la funzione $H(\cdot)$ è definita nel modo seguente.
   (a) scegli un valore $i^* \xleftarrow{\$} \{1, 2, \ldots, Q\}$ ed ipotizza che $\mathscr{A}$ attacchi u^* esattamente alla richiesta i^*-sima;
   (b) quando ricevi la i-sima richiesta per $H(\cdot)$ con input x, se $i \neq i^*$ scegli $\omega_i \xleftarrow{\$} \mathbb{Z}_p$ e restituisci $H(x) = g^{\omega_i}$;
   (c) quando ricevi la richiesta i^*-sima ritorna $H(x) = g^b$.
2. Quando l'attaccante richiede la chiave segreta corrispondente all'identità u:
   (a) se u è la i-sima richiesta ad $H(\cdot)$ ed $i \neq i^*$, ritorna $sk_u = (g^a)^{\omega_i} = H(u)^a$;
   (b) altrimenti se tale richiesta coincide con l'identificativo ipotizzato al passo (1a) (ovvero l'ipotesi fatta per il valore i^*), ritorna $\perp$.
3. Quando l'attaccante sceglie (m_0, m_1, u^*):
   (a) se u^* non è la i^*-sima richiesta ritorna $\perp$;
   (b) altrimenti scegli $j \xleftarrow{\$} \{0, 1\}$ e restituisci $c_j = (g^c, m_d \cdot \xi)$.
4. Sia j' il bit ritornato da $\mathscr{A}$. Restituisci 1 se $j' = j$, altrimenti ritorna 0.

Ovviamente $\mathscr{D}$ è eseguibile in tempo polinomiale $t \approx t_{\mathsf{BF}}$. Possiamo poi distinguere due casi, a seconda della natura della sfida ricevuta da $\mathscr{D}$.

Il caso $\xi = \Xi(g, g)^{abc}$. Sia $\mathcal{E}_\perp$ l'evento che $\mathscr{D}$ ritorni $\perp$. In questo caso non è difficile accorgersi che la vista di $\mathscr{A}$ nella simulazione realizzata da $\mathscr{D}$ è identica a quella in un'esecuzione dell'esperimento $\mathbf{Exp}^{\text{ind-cpa}}_{\mathsf{IBE},\Pi_{\mathsf{BF}}}(\mathscr{A}, n)$ a meno che non si verifichi l'evento $\mathcal{E}_\perp$. D'altra parte se $\mathcal{E}_\perp$ si verifica, il che accade con probabilità $1 - 1/Q$, allora $\mathscr{D}$ ha successo solamente con probabilità $1/2$. Pertanto:

$$\begin{aligned}
&\mathbb{P}\left[\mathscr{D}\left(g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi = \Xi(g, g)^{abc}\right] \\
&= \mathbb{P}\left[j = j' \mid \mathcal{E}_\perp\right] \cdot \mathbb{P}\left[\mathcal{E}_\perp\right] + \mathbb{P}\left[j = j' \mid \overline{\mathcal{E}_\perp}\right] \cdot \left(1 - \mathbb{P}\left[\mathcal{E}_\perp\right]\right) \\
&\geq \frac{1}{2} \cdot \left(1 - \frac{1}{Q}\right) + \left(\frac{1}{2} + \epsilon_{\mathsf{BF}}\right) \cdot \frac{1}{Q} \\
&= \frac{1}{2} + \frac{\epsilon_{\mathsf{BF}}}{Q}.
\end{aligned}$$

Il caso $\xi \xleftarrow{\$} \mathbb{G}$. Siccome ξ è uniformemente casuale, anche il crittotesto c_d prodotto da $\mathscr{D}$ è casuale. Ne segue che in questo caso la probabilità che $\mathscr{D}$ ritorni 1 è sempre $1/2$, anche se $\mathcal{E}_\perp$ non si verifica. Quindi:

$$\begin{aligned}
&\mathbb{P}\left[\mathscr{D}\left(g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi \xleftarrow{\$} \mathbb{G}_T\right] \\
&= \mathbb{P}\left[j = j' \mid \mathcal{E}_\perp\right] \cdot \mathbb{P}\left[\mathcal{E}_\perp\right] + \mathbb{P}\left[j = j' \mid \overline{\mathcal{E}_\perp}\right] \cdot \left(1 - \mathbb{P}\left[\mathcal{E}_\perp\right]\right) \\
&= \frac{1}{2} \cdot \left(1 - \frac{1}{Q}\right) + \frac{1}{2} \cdot \frac{1}{Q} \\
&= \frac{1}{2}.
\end{aligned}$$

Mettendo tutto insieme, abbiamo trovato:

$$\left| \mathbb{P}\left[\mathscr{D}\left(g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi = \Xi(g,g)^{abc}\right] - \mathbb{P}\left[\mathscr{D}\left(g, g^a, g^b, g^c, \xi\right) = 1 : \ \xi \xleftarrow{\$} \mathbb{G}_T\right] \right| \geq \frac{\epsilon_{\mathsf{BF}}}{Q},$$

e sostituendo l'espressione per ϵ_{BF} segue che $\mathscr{D}$ ha vantaggio (almeno) ϵ nel risolvere il problema BDDH, contro l'ipotesi che quest'ultimo sia (t, ϵ)-difficile. □

10.5 Miscellanea

Il risultato di Boneh e Franklin ha aperto la strada ad un vero e proprio filone di ricerca, che si è sviluppato molto negli anni successivi. Una trattazione esaustiva esula dagli scopi del testo, ci limitiamo quindi a fornire una panoramica dei risultati più rilevanti.

Sicurezza CCA e modello standard. Gli stessi Boneh e Franklin hanno proposto anche uno schema CCA-sicuro nel modello dell'oracolo casuale. Nel 2004 Boneh e Boyen [BB04a] hanno costruito il primo cifrario su base identità sicuro nel modello standard (basato sull'ipotesi DDH). La definizione di sicurezza data da Boneh e Boyen, tuttavia, è più debole di quella discussa nel Paragrafo 10.2 (cf. Definizione 10.2): l'attaccante deve decidere *in anticipo* (all'inizio dell'esperimento) l'identità u^* attaccare.

Il primo schema CPA-sicuro nel modello standard (nel senso della Definizione 10.2) è dovuto a Boneh, Boyen a Waters [Wat05]. A partire da quest'idea sono stati poi realizzati cifrari CCA-sicuri (sempre nel modello standard) via via più efficienti [Bon+07; BMW05; KG06; KV08]. D'altra parte Gentry [Gen06], ha costruito un cifrario CCA-sicuro modificando direttamente lo schema di Boneh e Boyen.

Firme digitali. Dal punto di vista teorico costruire firme digitali su base identità è molto più semplice che costruire cifrari. Già nel suo lavoro originale Shamir [Sha84] aveva osservato che una firma digitale standard implica una firma digitale su base identità. È sufficiente usare la firma due volte: una per generare le chiavi degli utenti e l'altra per realizzare le firme vere e proprie. In particolare la chiave segreta di Alice consiste di una coppia di chiavi pubblica/privata insieme ad un certificato che attesti la validità della chiave privata. Il certificato è generato dal KGC usando la chiave segreta principale msk per firmare la corrispondente chiave pubblica dell'utente, insieme alla sua identità. Sulla base di questa idea Bellare, Neven e Namprempre [BNN04], hanno presentato una costruzione generica di firme su base identità a partire da una firma digitale su base PKI. Se la firma digitale è sicura nel modello standard anche quella su base identità lo è. Più avanti Galindo, Herranz e Kiltz [GHK06], hanno mostrato che lo stesso principio si applica a firme digitali con proprietà aggiuntive (ad esempio le firme innegabili).

Lo svantaggio principale di questo approccio è l'efficienza: la firma risultante dovrà contenere la chiave pubblica di chi ha firmato e due firme digitali (il certificato e la firma vera e propria). Esistono anche soluzioni "dirette" (i.e., senza utilizzare firme digitali su base PKI) più efficienti nel modello standard, come quella di Paterson e Schuldt [PS06].

Un'osservazione interessante (attribuita a Naor) è che, d'altra parte, le firme su base identità implicano le firme digitali ordinarie. La chiave segreta principale è la chiave privata ed i parametri pubblici del sistema sono la chiave pubblica. La firma digitale di un messaggio m è la chiave segreta corrispondente all'identità $u = m$: la verifica può avvenire scegliendo un messaggio casuale m', cifrandolo con la chiave pubblica corrispondente ad u e controllando che sia possibile decifrare usando la firma come chiave di decifratura. Quest'idea è stata usata da Boneh, Lynn e Shacham [BLS04], e da Boneh e Boyen [BB04b], per ottenere firme digitali molto efficienti nel modello standard.

Sistemi gerarchici. Gli schemi *gerarchici* su base identità [GS02; HL02] sono una naturale estensione dei sistemi su base identità, in cui le identità degli utenti sono *vettori* di stringhe binarie. L'entità alla radice dell'albero genera le chiavi segrete degli utenti al primo livello; a loro volta gli utenti a livello ℓ derivano le chiavi per i loro "figli" a livello $\ell+1$. In questo modo il potere del KGC è in un certo senso distribuito e quest'ultimo cessa di essere il collo di bottiglia del sistema. Allo stesso tempo ciò riflette la struttura gerarchica di diverse istituzioni. Ad esempio il capo del gruppo di crittografia dell'università potrebbe ricevere le chiavi (`it`, `uniroma1`, `critto`) ed usarle per derivare le chiavi relative alle identità (`it`, `uniroma1`, `critto`, `alice`), corrispondenti all'indirizzo email `alice@critto.uniroma1.it`.

In [Cas+10], Cash, Hofheinz, Kiltz e Peikert definiscono (e realizzano) un oggetto curioso che metaforicamente può essere visto come un "albero bonsai crittografico". Gli alberi bonsai possono crescere in modo naturale, oppure un "curatore" può applicare tecniche di crescita naturale, indiretta e di propagazione per ottenere le forme estetiche più svariate. Allo stesso modo un "albero bonsai crittografico" è una collezione di tecniche che può essere "controllata" in diversi modi e che ha svariate applicazioni. Informalmente, l'albero è una gerarchia di "funzioni con botola" con alcune proprietà speciali. La cura dell'albero può essere realizzata da ogni entità del sistema, indipendentemente dal fatto che essa sia un utente che genera una firma oppure un simulatore in una dimostrazione di sicurezza. Una delle applicazioni è proprio un cifrario su base identità di tipo gerarchico (*Hierarchical Identity-Based Encryption*, HIBE). Una seconda applicazione è una realizzazione del paradigma "hash & firma" (cf. Paragrafo 8.2) sicura nel modello standard.

Crittografia su base attributi. Nel 2005 Sahai e Waters [SW05], hanno generalizzato il principio dei crittosistemi su base identità descrivendone una versione "sfocata" (*fuzzy* in inglese) in cui le identità degli utenti sono un insieme di "attributi descrittivi". In un sistema di questo tipo, Alice, con chiave segreta corrispondente all'identità u, può decifrare un testo cifrato con l'identità u' (del Bianconiglio) a patto che u ed u' siano "vicine" rispetto ad una qualche metrica. Ci sono due principali applicazioni: la prima è nel contesto dei sistemi che utilizzano *identificativi biometrici*. Supponiamo ad esempio che l'identità di Alice sia l'iride del suo occhio. Siccome i lettori biometrici (un lettore d'iride nel nostro caso) soffrono molto del rumore associato alle misure, in questo contesto la tolleranza agli errori di un sistema "sfocato" come quello descritto sopra può rivelarsi molto utile.

La seconda applicazione è quella della *crittografia su base attributi*. Qui Alice vuole cifrare un messaggio diretto ad un certo insieme d'utenti che rispettano alcuni attributi specifici. Questo filone ha avuto diversi sviluppi negli ultimi anni [Goy+06; BSW07; OSW07; Goy+08; Lew+10].

Esercizi

Esercizio 10.1. Supponiamo che la revoca dei certificati sia gestita come segue: quando il Bianconiglio dichiara che la chiave segreta corrispondente alla sua chiave pubblica pk_B è compromessa, invia alla CA un messaggio contenente tale affermazione, firmato rispetto alla chiave pubblica pk_B. Ricevuto il messaggio, la CA revoca il relativo certificato.
Spiegare perché non ha alcuna importanza il fatto che un avversario che sia in possesso della chiave segreta del Bianconiglio sia in grado di forgiare firme rispetto pk_B.

Esercizio 10.2. Dare una definizione di schema di firma digitale su base identità e definirne la sicurezza.

Esercizio 10.3. Supponiamo che nello schema di Boneh e Franklin la funzione $H(u)$ sia definita come $H(u) = (g')^u \cdot h$, per $g', h \xleftarrow{\$} \mathbb{G}$. Dire se il sistema è CPA-sicuro, oppure esibire un attacco ed analizzarne la complessità.

Esercizio 10.4. Esibire un attacco CCA contro lo schema di Boneh e Franklin.

Esercizio 10.5. Consideriamo il seguente schema, che trasforma un cifrario su base identità $\Pi = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in un cifrario a chiave pubblica CCA-sicuro $\Pi^* = (\mathsf{Gen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$. L'algoritmo Gen^* lancia $\mathsf{Setup}(1^n)$ e pone $pk = ppub$, $sk = msk$. L'algoritmo Enc^*, dato un messaggio m, seleziona un'identità casuale $u \xleftarrow{\$} \mathcal{U}$ e ritorna $c = (c_0, c_1) = (\mathsf{Enc}_u(ppub, m), u)$. L'algoritmo Dec^*, dato $c = (c_0, c_1)$, lancia $\mathsf{Gen}_{msk}(c_1)$ ottenendo così sk_u e recupera il messaggio originale $m = \mathsf{Dec}_{sk_u}(c_0)$.

1. Mostrare che non è possibile ridurre la sicurezza CCA di Π^* alla sicurezza CPA di Π. Indicare dove la riduzione fallisce.

2. Sia $\Pi' = (\mathsf{Gen}', \mathsf{Sign}, \mathsf{Vrfy})$ uno schema di firma digitale. Consideriamo la seguente modifica dello schema descritto. La chiave pubblica contiene anche $(pk', sk') \leftarrow \mathsf{Gen}'(1^n)$. Il crittotesto comprende la firma $\sigma \leftarrow \mathsf{Sign}_{sk'}(c_0)$. In decifratura si verifica prima che $\mathsf{Vrfy}_{pk'}(c_0, \sigma) = 1$, altrimenti si restituisce $\perp$. Se la verifica va a buon fine, si recupera m usando (c_0, c_1) come sopra. Mostrare che Π^* è CCA sicuro se Π è CPA-sicuro e se Π' è ufcma. Perché lo schema di firma ha risolto il problema incontrato al punto precedente?

Letture consigliate

[AF99] C. Adams e S. Farrell. *Internet X.509 Public Key Infrastructure Certificate Management Protocols*. RFC 2510. 1999. URL: `http://www.ietf.org/rfc/rfc2510.txt`.

[BB04a] Dan Boneh e Xavier Boyen. "Secure Identity Based Encryption Without Random Oracles". In: *CRYPTO*. 2004, pp. 443–459.

[BB04b] Dan Boneh e Xavier Boyen. "Short Signatures Without Random Oracles". In: *EUROCRYPT*. 2004, pp. 56–73.

[BF01] Dan Boneh e Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. 2001, pp. 213–229.

[BLS04] Dan Boneh, Ben Lynn e Hovav Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptology* 17.4 (2004), pp. 297–319.

[BMW05] Xavier Boyen, Qixiang Mei e Brent Waters. "Direct chosen ciphertext security from identity-based techniques". In: *ACM Conference on Computer and Communications Security*. 2005, pp. 320–329.

[BNN04] Mihir Bellare, Chanathip Namprempre e Gregory Neven. "Security Proofs for Identity-Based Identification and Signature Schemes". In: *EUROCRYPT*. 2004, pp. 268–286.

[Bol+07] Alexandra Boldyreva, Marc Fischlin, Adriana Palacio e Bogdan Warinschi. "A Closer Look at PKI: Security and Efficiency". In: *Public Key Cryptography*. 2007, pp. 458–475.

[Bon+07] Dan Boneh, Ran Canetti, Shai Halevi e Jonathan Katz. "Chosen-Ciphertext Security from Identity-Based Encryption". In: *SIAM J. Comput.* 36.5 (2007), pp. 1301–1328.

[BSW07] John Bethencourt, Amit Sahai e Brent Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *IEEE Symposium on Security and Privacy*. 2007, pp. 321–334.

[Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1865.

[Cas+10] David Cash, Dennis Hofheinz, Eike Kiltz e Chris Peikert. "Bonsai Trees, or How to Delegate a Lattice Basis". In: *EUROCRYPT*. 2010, pp. 523–552.

[Coc01] Clifford Cocks. "An Identity Based Encryption Scheme Based on Quadratic Residues". In: *IMA Int. Conf.* 2001, pp. 360–363.

[Gen06] Craig Gentry. "Practical Identity-Based Encryption Without Random Oracles". In: *EUROCRYPT*. 2006, pp. 445–464.

[GHK06] David Galindo, Javier Herranz e Eike Kiltz. "On the Generic Construction of Identity-Based Signatures with Additional Properties". In: *ASIACRYPT*. 2006, pp. 178–193.

[Goy+06] Vipul Goyal, Omkant Pandey, Amit Sahai e Brent Waters. "Attribute-based encryption for fine-grained access control of encrypted data". In: *ACM Conference on Computer and Communications Security*. 2006, pp. 89–98.

[Goy+08] Vipul Goyal, Abhishek Jain, Omkant Pandey e Amit Sahai. "Bounded Ciphertext Policy Attribute Based Encryption". In: *ICALP (2)*. 2008, pp. 579–591.

[GS02] Craig Gentry e Alice Silverberg. "Hierarchical ID-Based Cryptography". In: *ASIACRYPT*. 2002, pp. 548–566.

[HL02] Jeremy Horwitz e Ben Lynn. "Toward Hierarchical Identity-Based Encryption". In: *EUROCRYPT*. 2002, pp. 466–481.

[IET07] IETF. *An Open Specification for Pretty Good Privacy (openpgp)*. RFC 4880. 2007. URL: `http://www.ietf.org/html.charters/openpgp-charter.html`.

[ITU88] ITU. *Information technology — Open systems interconnection — The Directory: Public-key and attribute certificate frameworks*. ITU-T X.509. 1988.

[KG06] Eike Kiltz e David Galindo. "Direct Chosen-Ciphertext Secure Identity-Based Key Encapsulation Without Random Oracles". In: *ACISP*. 2006, pp. 336–347.

[KV08] Eike Kiltz e Yevgeniy Vahlis. "CCA2 Secure IBE: Standard Model Efficiency through Authenticated Symmetric Encryption". In: *CT-RSA*. 2008, pp. 221–238.

[Lew+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima e Brent Waters. "Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption". In: *EUROCRYPT*. 2010, pp. 62–91.

[Maa04] Martijn Maas. "Pairing-Based Cryptography". Tesi di dott. Technische Universiteit Eindhoven, 2004.

[OSW07] Rafail Ostrovsky, Amit Sahai e Brent Waters. "Attribute-based encryption with non-monotonic access structures". In: *ACM Conference on Computer and Communications Security*. 2007, pp. 195–203.

[PS06] Kenneth G. Paterson e Jacob C. N. Schuldt. "Efficient Identity-Based Signatures Secure in the Standard Model". In: *ACISP*. 2006, pp. 207–222.

[Sha84] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes". In: *CRYPTO*. 1984, pp. 47–53.

[Sil86] Joseph H. Silverman. *Arithmetic of Elliptic Curves*. Springer-Verlang, 1986.

[SW05] Amit Sahai e Brent Waters. "Fuzzy Identity-Based Encryption". In: *EUROCRYPT*. 2005, pp. 457–473.

[Wat05] Brent Waters. "Efficient Identity-Based Encryption Without Random Oracles". In: *EUROCRYPT*. 2005, pp. 114–127.

Parte II

Protocolli

CAPITOLO 11

"Scambi di mano" sicuri

«Tu chi sei?» disse il Bruco. [...] «Io non mi posso spiegare, dolente, signore,» disse Alice, «perché non sono me stessa, capisce?».

Lewis Carroll, Le avventure di Alice nel Paese delle Meraviglie [Car65]

Nella prima parte del testo abbiamo studiato i "fondamenti" della crittografia teorica, ovvero le tecniche di base per realizzare i requisiti fondamentali in un contesto di "comunicazione sicura". In particolare abbiamo visto come sia possibile ottenere confidenzialità ed integrità di messaggio. In questa seconda parte vedremo come le tecniche che abbiamo studiato possano essere applicate in contesti più generali, per costruire protocolli crittografici complessi.

Il potere dell'interazione. Immaginiamo che Alice sia prigioniera in una cella la cui porta è chiusa con due serrature: una protetta da un lucchetto blu e l'altra da un lucchetto rosso. La cella è completamente buia. Lo Stregatto compare in aiuto di Alice con in mano due chiavi, dichiarando che le due chiavi sono di colori diversi: una rossa per aprire il lucchetto rosso ed una blu per aprire il lucchetto blu. Purtroppo Alice non si fida dello Stregatto; esiste un modo in cui Alice possa essere sicura che le chiavi proposte dello Stregatto abbiano effettivamente due colori diversi? In effetti un modo esiste, grazie all'*interazione*.

Alice chiede allo Stregatto di identificare i colori delle due chiavi, quindi prende le chiavi e le mischia facendo attenzione a ricordare la posizione di ciascuna. A questo punto chiede allo Stregatto di identificarle nuovamente:[66] se le due chiavi fossero dello stesso colore lo Stregatto non avrebbe modo di convincere Alice se non con probabilità migliore di 1/2. Basta che Alice ripeta il procedimento più volte per essere sicura che lo Stregatto la stia imbrogliando solo con probabilità trascurabile!

[66] Stiamo assumendo qui che lo Stregatto non abbia problemi a vedere al buio, ma la vista dei gatti si sa è molto migliore di quella degli umani...

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_11, © Springer-Verlag Italia 2012

L'esempio di Alice e lo Stregatto ci fa capire che esistono contesti pratici in cui l'interazione tra le diverse parti può essere complessa e finalizzata agli scopi più svariati (vedremo diversi esempi in questo capitolo e nei successivi). In questi casi, solitamente, ci si affida ad un *protocollo crittografico*. Questo, a sua volta, potrebbe usare come "mattoni di base" le tecniche e/o le primitive che abbiamo studiato nella prima parte del testo. In astratto, un protocollo crittografico non è altro che una qualche forma di interazione (con una sua sintassi ed una semantica ben specifica) tra una o più parti di un sistema con lo scopo di raggiungere un certo obiettivo (i.e., soddisfare una certa proprietà) *minimizzando la quantità di fiducia a priori* che le entità coinvolte devono avere nelle rispettive controparti.

Sicurezza? Come vedremo i protocolli sono le costruzioni più complesse in crittografia, in quanto spesso intricati e difficili da analizzare. Una buona regola generale quando si progetta un protocollo crittografico è pensare che chi esegue il protocollo in modo onesto stia in realtà interagendo con l'attaccante ("modello paranoia"). In altri termini, non vogliamo basare un protocollo crittografico interamente sulla "fiducia".[67]

In ultima analisi quando progettiamo un protocollo crittografico dobbiamo sempre pensare che l'attaccante: (i) può intercettare tutti i messaggi inviati durante l'esecuzione del protocollo, (ii) può alterare, re-indirizzare o distruggere il flusso naturale del protocollo (questo include in generale la possibilità di modificare messaggi o inserirne di nuovi) e (iii) può essere una parte legittima oppure una parte esterna al sistema (o eventualmente una combinazione dei due). La Tab. 11.1 descrive alcuni attacchi tipici nel contesto dei protocolli crittografici.

- In un attacco di tipo "scambio della sessione" (*session hijacking*), la Regina lascia che il protocollo termini correttamente per poi sostituirsi ad Alice in un secondo momento (ad esempio dopo che Alice ed il Bianconiglio si siano mutuamente autenticati).

- In un attacco di tipo "spionaggio" (*eavesdropping* in inglese), la Regina si limita ad intercettare i messaggi scambiati per carpire qualche informazione segreta. (Più in generale si parla di attacchi *passivi*, cf. Paragrafo 11.2.)

[67] Ci sono casi in cui invece la fiducia ha un ruolo importante. A seconda del contesto, infatti, cambiamo gli incentivi che spingono le parti coinvolte ad avere fiducia. Tali incentivi si individuano nell'etica, nella reputazione (ad esempio nei sistemi di commercio elettronico), nella legge, nella minaccia fisica e nella distruzione mutua assicurata (come ad esempio è accaduto durante la Guerra Fredda).

Tab. 11.1. Esempi di attacchi tipici nel contesto dei protocolli crittografici

Attacco	Descrizione
Scambio della sessione	La Regina lascia che Alice ed il Bianconiglio terminino l'esecuzione del protocollo e poi si sostituisce ad una delle due parti
Spionaggio	La Regina intercetta i messaggi scambiati nel protocollo
Ri-uso	La Regina memorizza alcuni dei messaggi scambiati nel protocollo nel protocollo e li utilizza in esecuzioni future del protocollo
Modifica	La Regina altera i messaggi scambiati
Negazione del servizio	La Regina cerca di portare il funzionamento del sistema al limite delle prestazioni, lavorando su uno dei parametri d'ingresso, fino a renderlo non più in grado di erogare il servizio
Uomo-nel-mezzo	La Regina impersona Alice nei confronti del Bianconiglio
Inter-allacciamento	La Regina lancia diverse istanze del protocollo in parallelo ed usa i messaggi intercettati in una sessione per attaccarne un'altra
Riflessione	La Regina re-invia alla sorgente i messaggi intercettati

- In un attacco di tipo “ri-uso” (*re-play*), la Regina intercetta alcuni messaggi scambiati da Alice ed il Bianconiglio per poi utilizzarli altrove (con la speranza che essi siano ancora considerati come “validi”).

- In un attacco di tipo “modifica” (*modification*), la Regina modifica i messaggi scambiati nel protocollo con lo scopo, ad esempio, di far fallire l'interazione tra Alice ed il Bianconiglio. (Più in generale si parla di attacchi *attivi*, cf. Paragrafo 11.2.)

- In un attacco di tipo “negazione del servizio” (*denial of service*), la Regina tenta di portare il sistema al limite delle prestazioni, sperando così di metterlo fuori uso.

- In un attacco “uomo-nel-mezzo” (*man-in-the-middle*), la Regina interpreta il ruolo di Alice nei confronti del Bianconiglio e, viceversa, del Bianconiglio nei confronti di Alice.

- In un attacco di tipo “inter-allacciamento” (*interleaving*), la Regina lancia diverse istanze del protocollo in parallelo, inter-allacciando i messaggi di tali istanze con lo scopo di violare la sicurezza in una di esse.

- Gli attacchi di tipo “riflessione” (*reflection*) sono particolarmente efficaci nei protocolli con struttura simmetrica. Supponiamo che il protocollo preveda che il Bianconiglio debba rispondere ad una sfida inviata da Alice (e viceversa che Alice debba rispondere ad una sfida inviata dal Bianconiglio). (Vedremo più avanti che questi protocolli sono detti a “sfida e risposta”, cf.

Paragrafo 11.3.) La Regina vuole sostituirsi al Bianconiglio; non avendo le credenziali per rispondere correttamente alla sfida di Alice, “ri-lancia” la stessa sfida ad Alice, sperando di ottenere la risposta.

Guida per il lettore. Ciò premesso, la struttura di questo capitolo è la seguente. Nel Paragrafo 11.1 discuteremo il problema dello scambio delle chiavi. Passeremo quindi ad occuparci di *autenticazione*. Dopo aver definito (nel Paragrafo 11.2) i requisiti formali di sicurezza nel contesto dell'autenticazione, introdurremo (nel Paragrafo 11.3) il paradigma sfida e risposta e le tecniche (simmetriche ed asimmetriche) atte a realizzarlo. Analizzeremo quindi (nel Paragrafo 11.4) due protocolli molto efficienti dovuti ad Hopper e Blum; tali protocolli sono così efficienti che potrebbero essere usati da un essere umano senza l'uso di un calcolatore. Infine daremo qualche cenno relativo al concetto di autenticazione *mediata* (cf. Paragrafo 11.5).

11.1 Scambio di chiavi

La gestione delle chiavi è senz'altro uno dei punti più delicati in un sistema crittografico. Nel contesto della crittografia asimmetrica, il problema principale è quello dell'autenticità delle chiavi pubbliche degli utenti, indirizzato dalle infrastrutture a chiave pubblica (PKI, cf. Paragrafo 10.1). Nel contesto della crittografia simmetrica, d'altra parte, il problema principale è quello della distribuzione iniziale delle chiavi condivise. Infatti, per ovvi motivi, tali chiavi non possono essere trasmesse “in chiaro” agli utenti. Una possibile soluzione al problema potrebbe essere quella di organizzare un incontro tra gli utenti del sistema, generare la chiave segreta sul momento e dare una copia a ciascuna parte. Ad esempio il manager di una società può condividere una chiave segreta con ciascun impiegato il primo giorno di lavoro. Tale approccio non è scalabile, sopratutto se ogni impiegato deve anche condividere una chiave segreta con ogni altro impiegato, oppure se l'azienda è molto grande e ha diverse sedi dislocate in punti molto distanti tra loro. Inoltre tutte le chiavi condivise andrebbero memorizzate in modo sicuro: se ci sono U utenti nel sistema, il numero di chiavi segrete è $\binom{U}{2} = \Theta(U^2)$ e ciascun utente deve memorizzare $U - 1$ chiavi. (Notare che queste potrebbero non essere le uniche chiavi di cui si ha bisogno, in quanto potrebbero essere previste altre chiavi per altri scopi.)

Ovviamente più chiavi segrete si posseggono e più è complesso memorizzarle in modo sicuro. Ad esempio, i calcolatori su cui queste chiavi sono memorizzate potrebbero essere sotto il controllo di un virus il cui scopo è quello

di appropriarsene; pertanto memorizzare le chiavi su un computer non risolve necessariamente il problema. Si preferisce quindi l'uso delle *smartcard*: un hardware dedicato, delle dimensioni simili ad una carta di credito, che permette di memorizzazione dati in modo sicuro. È molto più raro che le smartcard siano affette da virus. D'altra parte esse offrono una memoria limitata, il che le rende inadatte a memorizzare un gran numero di chiavi segrete.

In questo paragrafo analizzeremo il problema da un punto di vista diverso, cercando di rispondere alla seguente domanda:

> *Supponiamo che Alice ed il Bianconiglio non si siano mai incontrati. Possono eseguire un un protocollo crittografico (senza contare su terze parti fidate) in modo che al termine del processo condividano un segreto?*

In una qualche misura la risposta a questa domanda è affermativa. Come vedremo nel seguito, per facilitare il problema, si è soliti distinguere tra *chiavi principali* e *chiavi di sessione*. L'idea è quella di distribuire in modo sicuro (e come visto molto costoso!) una chiave condivisa iniziale, detta chiave principale, che può poi essere utilizzata in diversi modi per condividere una o più chiavi di sessione, da utilizzare per uno scopo specifico.

Gestione delle chiavi crittografiche. In generale la gestione delle chiavi crittografiche riguarda: (i) la loro creazione (compito non sempre banale, come abbiamo visto ad esempio nel caso di RSA, cf. Crittosistema 6.1), (ii) la loro distribuzione ed autenticazione, (iii) il loro uso e (iv) la loro memorizzazione.

Quanto al punto (iii), la soluzione meno rischiosa è quella di prevedere che una chiave crittografica sia usata per un *unico* scopo. Idealmente però, vorremmo assicurare che una *stessa* chiave sia utilizzabile (in modo sicuro) per scopi *diversi*. Questo semplificherebbe la complessità associata alla distribuzione delle chiavi. Supponiamo di avere un insieme di primitive che rispetta un qualche requisito formale di sicurezza (e.g. un insieme di schemi di firma digitale). Cosa possiamo dire della loro sicurezza quando esse condividono tutte le stessa chiave? Recentemente questa domanda è stata indirizzata da Acar, Belenkiy, Bellare e Cash [Aca+10] i quali hanno definito il concetto di *agilità crittografica*. Un insieme di primitive crittografiche (dello stesso tipo) che soddisfa un certo requisito di sicurezza è detto *agile* se rimane sicuro anche quando tali primitive condividono la *stessa* chiave. Mentre alcune primitive (ad esempio le funzioni hash resistenti alle collisioni ed i cifrari asimmetrici CPA-sicuri) sono

agili, altre primitive (come i PRF, i MAC, i cifrari simmetrici CPA-sicuri, le firme digitali ed i cifrari asimmetrici CCA-sicuri) non lo sono.

Esistono poi questioni più pratiche. La prima è quella di prevedere chiavi di riserva, da utilizzare quando un utente smarrisce la propria chiave. È anche importante distinguere tra chiavi revocate (perché l'utente ne ha fatto un uso illecito) e chiavi che non sono più utilizzabili perché è scaduto il loro tempo di vita. In generale, infatti, è una buona regola associare ad una chiave un tempo di vita. Ciò ha diversi vantaggi. Prima di tutto limita il tempo in cui la chiave è esposta agli attacchi, rendendo il sistema più robusto. Inoltre, ciò inibisce il potere di un attaccante che è riuscito ad impadronirsi della chiave, in quanto essa sarà cambiata dopo un certo tempo; questo rende diverse sessioni indipendenti l'una dall'altra. D'altra parte rinnovare le chiavi di un intero sistema molto spesso ha un costo enorme.

Diffie-Hellman. Il primo protocollo che risolve il problema dello scambio delle chiavi è stato proposto da Diffie ed Hellman [DH76]. Sebbene tale soluzione soddisfi un requisito abbastanza “debole” di sicurezza, essa è stata fondamentale nello sviluppo di altri protocolli via via più evoluti. Inoltre il protocollo di Diffie-Hellman è molto noto per motivi storici, in quanto è stato il primo protocollo a fornire una soluzione al problema di scambio chiavi tra due parti che non condividono alcun segreto (ed infatti esso è nato insieme alla crittografia a chiave pubblica).

Per poter analizzare la sicurezza formale del protocollo bisogna introdurre un modello. In effetti la nostra analisi sarà per lo più euristica, quindi ci limiteremo a descrivere l'intuizione, rimandando alla letteratura per i dettagli. Supponiamo che Alice ed il Bianconiglio eseguano un protocollo per condividere una chiave segreta k elemento di un qualche insieme $\mathcal{K}$. Durante l'esecuzione del protocollo le due parti hanno accesso a sorgenti di randomicità indipendenti e scambiano alcuni messaggi. Al termine della procedura, Alice otterrà la chiave k_A ed il Bianconiglio la chiave k_B e richiederemo che $k_A = k_B = k \in \mathcal{K}$: in questo modo Alice ed il Bianconiglio hanno condiviso la chiave k. Il requisito di sicurezza più semplice che vogliamo da un protocollo di questo tipo è che nessun attaccante in grado di intercettare la comunicazione (in un numero arbitrario d'istanze del protocollo) sia in grado di distinguere le chiave k da un elemento casuale in $\mathcal{K}$.

La soluzione originale proposta da Diffie ed Hellman è mostrata in Fig. 11.1. Dato il parametro di sicurezza n, Alice genera un gruppo $\mathbb{G}$ con ordine un primo q (ad n bit) e con generatore g. Quindi estrae $a \xleftarrow{\$} \mathbb{Z}_q$ a caso, calcola $\alpha = g^a$

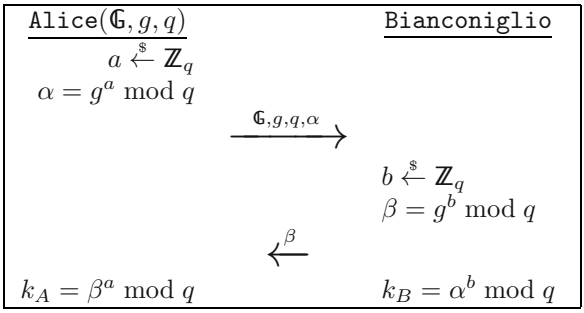


Fig. 11.1. L'idea di Diffie-Hellman

ed invia al Bianconiglio $(\mathbb{G}, g, q, \alpha)$. Il Bianconiglio a sua volta estrae $b \xleftarrow{\$} \mathbb{Z}_q$ e risponde ad Alice con $\beta = g^b$. Alice calcola infine $k_A = \beta^a$ ed il Bianconiglio $k_B = \alpha^b$, così che:

$$k_A = \beta^a = (g^b)^a = g^{ab} = (g^a)^b = \alpha^b = k_B,$$

e le due parti hanno condiviso $k = g^{ab}$.

Quanto alla sicurezza del protocollo, l'ipotesi minimale è che il problema del logaritmo discreto sia difficile in $\mathbb{G}$. Infatti se un attaccante può calcolare a (risp. b) a partire da $\alpha = g^a$ (risp. $\beta = g^b$) può anche calcolare k, violando la sicurezza del protocollo con un semplice attacco passivo (ovvero semplicemente osservando una singola interazione onesta tra Alice ed il Bianconiglio). In generale comunque ciò non è sufficiente, in quanto potrebbero esistere attacchi diversi, in cui non è necessario calcolare a e b per distinguere k da un elemento casuale in $\mathcal{K}$. Non è complesso verificare che la sicurezza passiva segue dall'ipotesi DDH (cf. Esercizio 11.6).

Considerazioni pratiche. Tutti gli elementi coinvolti nel protocollo fanno parte del gruppo finito $\mathbb{G}$. (Si veda la fine del Paragrafo 6.3 per alcune possibili scelte di $\mathbb{G}$; in generale il parametro q non deve essere troppo piccolo, per evitare l'attacco a forza bruta.[68]) Per usare il protocollo in pratica, sono necessarie stringa binarie, quindi si necessita di una mappa che trasformi k in una stringa

[68] Ad esempio un attaccante potrebbe sostituire α con 1 oppure con un elemento x appartenente ad un sottogruppo di $\mathbb{G}$ di ordine non troppo grande. Per questo motivo, quando un valore x è ricevuto nel protocollo, bisogna sempre controllare che $x^q \equiv 1$ in $\mathbb{G}$.

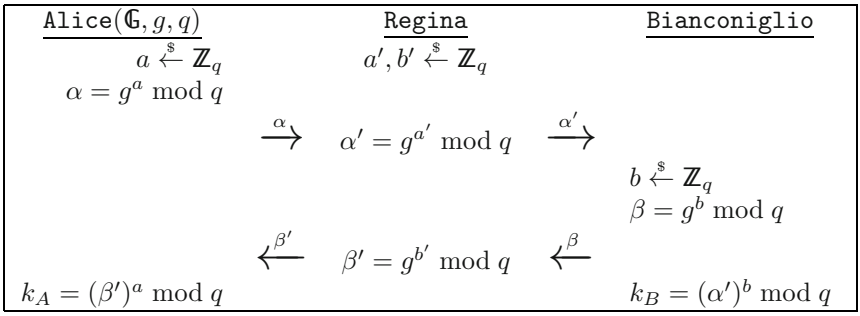


Fig. 11.2. Attacco MiM al protocollo Diffie-Hellman

mantenendo la proprietà di uniformità. Ciò può essere ottenuto, ad esempio, usando gli estrattori di randomicità (cf. Paragrafo 3.5).

Una seconda questione pratica riguarda la scelta e la validazione dei parametri. Una soluzione banale a questo problema è assumere che esista una sorta di *stringa comune di riferimento* (che tutti ritengono autentica) contenente (una descrizione di) $(\mathbb{G}, g, q)$.

Attacchi attivi. Non è complesso vedere che il protocollo di Diffie-Hellman è completamente insicuro in presenza di attacchi attivi. Un semplice attacco MiM (cf. Tab. 11.1) è mostrato in Fig. 11.2. La Regina, si dispone in mezzo tra Alice ed il Bianconiglio e gioca il ruolo del Bianconiglio nei confronti di Alice (e viceversa). Quando Alice invia $\alpha = g^a$, la Regina inoltra al Bianconiglio $\alpha' = g^{a'}$, avendo scelto a' a suo piacere. Allo stesso modo, quando il Bianconiglio invia $\beta = g^b$ ad Alice, la Regina lo rimpiazza con $\beta' = g^{b'}$ avendo scelto b' suo piacere. In questo modo la Regina ha condiviso la chiave $k_A = (\beta')^a = (\alpha)^{b'} = g^{ab'}$ con Alice e la chiave $k_B = (\alpha')^b = (\beta)^{a'} = g^{a'b}$ con il Bianconiglio, mentre Alice ed il Bianconiglio sono convinti di aver condiviso la stessa chiave k.

Diffie-Hellman autenticato? Dopo che il protocollo di Diffie-Hellman è stato proposto, sono stati compiuti numerosi sforzi per renderlo sicuro in presenza di attacchi attivi e per permettere inoltre alle due parti che lo eseguono di autenticarsi mutuamente. Il progetto di protocolli alla Diffie-Hellman autenticati si è rivelato complesso sia per quanto riguarda la costruzione che per quanto riguarda l'analisi di sicurezza, soprattutto se si vuole tenere alta l'effi-

cienza. Passi importanti nella definizione di un metodo rigoroso per analizzare questi protocolli, sono stati compiuti da Bellare e Rogaway [BR93] e da Bellare, Canetti e Krawczyk [BCK98].

Comunque, ottenere una qualunque forma di autenticazione, richiede che le due parti condividano qualche informazione iniziale che tipicamente prende la forma di una chiave crittografica con alto contenuto entropico: una chiave segreta principale (che può essere usata in un cifrario oppure in un codice autenticatore di messaggio), oppure una coppia di chiavi pubblica/privata (che possono essere usate in un cifrario a chiave pubblica o per produrre firme digitali). È naturale chiedersi che motivo c'è di eseguire il protocollo Diffie-Hellman se si condivide già un segreto. Il motivo, come anticipato all'inizio del capitolo, è che il segreto condiviso (distribuito in modo costoso) ha il ruolo di chiave principale che può essere utilizzata per condividere un numero arbitrario di chiavi di sessione in modo molto efficiente; tali chiavi potranno poi essere usate per altri scopi. In questo contesto esistono diverse soluzioni con sicurezza dimostrabile, si vedano [Bir+91; DOW92; BR93; BCK98; CK01; CK02].

Tra le versioni più efficienti, che utilizzano *solamente due messaggi* come nel protocollo Diffie-Hellman originale, ci sono il protocollo MQV [Law+03], la sua versione con sicurezza dimostrabile (nel modello dell'oracolo casuale) HMQV [Kra05] ed un protocollo recente dovuto a Gennaro, Krawczyk e Rabin [GKR10].

Per quanto ci riguarda, ci limiteremo a discutere alcuni tentativi *euristici* (tratti da [FS03, Capitolo 12]) per evitare l'attacco di Fig. 11.2 attraverso l'uso di un codice autenticatore di messaggio (cf. Capitolo 7).

Primo tentativo. La prima idea per evitare l'attacco MiM di Fig. 11.2 è quello di *autenticare* la chiave k negoziata, in modo da assicurarsi che Alice ed il Bianconiglio stiano in effetti condividendo la stessa chiave. Sia $\Pi_{\mathsf{MAC}} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ un MAC universalmente inforgiabile contro attacchi a messaggio scelto (cf. Definizione 7.2). Sia $k_M \leftarrow \mathsf{Gen}(1^n)$ la chiave segreta principale. Consideriamo il seguente protocollo:

1. Alice ed il Bianconiglio eseguono il protocollo di Fig. 11.1.
2. Dopo aver calcolato k, Alice calcola $\phi_A \leftarrow \mathsf{Tag}_{k_M}(k_A)$ e lo invia al Bianconiglio.
3. Il Bianconiglio verifica ϕ_A controllando che $\mathsf{Vrfy}_{k_M}(k_B, \phi_A) = 1$ ed invia a sua volta $\phi_B \leftarrow \mathsf{Tag}_{k_M}(k_B)$ ad Alice. Alice infine controlla che $\mathsf{Vrfy}_{k_M}(k_A, \phi_B)$ restituisca 1.

Una prima osservazione è che il protocollo potrebbe essere più compatto se il Bianconiglio inviasse il suo autenticatore insieme al valore β (nel secondo messaggio del protocollo in Fig. 11.1). Inoltre sarebbe bene non usare direttamente k come argomento dell'autenticatore (in quanto quest'ultimo potrebbe avere qualche debolezza e rivelare qualche informazione su k). Il problema principale, tuttavia, è che i due autenticatori sono relativi allo *stesso* messaggio: un attaccante può intercettare ϕ_A ed inviarlo successivamente ad Alice, facendosi autenticare come il Bianconiglio (attacco di tipo riflessione, cf. Tab. 11.1). Per evitare questo attacco, faremo in modo che un autenticatore inviato nel corso del protocollo autentichi anche *tutti* i messaggi scambiati in precedenza.

Secondo tentativo. Consideriamo il seguente scambio di messaggi:

1. Dato il parametro di sicurezza 1^n, Alice genera $(\mathbb{G}, g, q)$.
2. Alice estrae $a \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\alpha = g^a$. Invia quindi $m_A = (\alpha, \mathbb{G}, g, q)$ al Bianconiglio, insieme all'autenticatore $\phi_A \leftarrow \mathsf{Tag}_{k_M}(m_A)$.
3. Il Bianconiglio verifica ϕ_A, controlla i parametri $(\alpha, \mathbb{G}, g, q)$ in m_A, estrae $b \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\beta = g^b$. Infine invia (β, ϕ_B) ad Alice dove $\phi_B \leftarrow \mathsf{Tag}_{k_M}(m_A||\beta)$. Calcola poi la chiave $k = \alpha^b$.
4. Alice verifica ϕ_B ed usa β per calcolare la chiave $k = \beta^a$.

Come anticipato, in questo caso ϕ_A si riferisce a tutti i valori $(\alpha_A, \mathbb{G}, g, q)$ mentre ϕ_B si riferisce sia ad m_A che al valore β calcolato dal Bianconiglio. Controllare $(\mathbb{G}, g, q)$ al passo (2). significa accertarsi che $\#\mathbb{G}$ e q siano valori opportuni e che g sia effettivamente un generatore di $\mathbb{Z}_q$. Controllare α (risp. β) significa accertarsi che $\alpha^q = 1$ (risp. $\beta^q = 1$) in $\mathbb{G}$. L'attacco di tipo riflessione qui non funziona, in quanto stavolta ϕ_B si riferisce anche ai messaggi scambiati in precedenza e quindi l'attaccante non può farsi autenticare al posto del Bianconiglio intercettando ϕ_A e replicandolo verso Alice.

Restano comunque alcune debolezze. Prima di tutto i parametri non sono negoziati; sarebbe bene negoziare i parametri in modo che entrambe le parti siano d'accordo sul loro utilizzo. Inoltre si può attuare un attacco di tipo riuso (cf. Tab. 11.1): l'attaccante può registrare il primo messaggio destinato ad Alice ed inviarlo al Bianconiglio in un'altra istanza del protocollo. Sebbene ciò non implichi che l'attaccante venga a conoscenza della chiave segreta k, vorremmo comunque poter evitare una falsa "autenticazione".

Terzo tentativo. Possiamo risolvere le debolezze evidenziate come segue:

1. Alice sceglie il minimo valore per $\#\mathbb{G}$ (i.e., $s = \min \#\mathbb{G}$) ed un Nonce N_A. Invia quindi $m_A = (s, N_A)$ al Bianconiglio.

2. Dato il parametro di sicurezza 1^n ed il valore di s, il Bianconiglio genera $(\mathbb{G}, g, q)$, estrae $b \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\beta = g^b$. Invia quindi ad Alice $m_B = (\beta, \mathbb{G}, g, q)$ e l'autenticatore $\phi_B \leftarrow \mathsf{Tag}_{k_M}(m_A||m_B)$.

3. Alice verifica ϕ_B, ed i valori $(\beta, \mathbb{G}, g, q)$. Estrae $a \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\alpha = g^a$ e $k = \beta^a$. Invia (α, ϕ_A) al Bianconiglio, dove $\phi_A \leftarrow \mathsf{Tag}_{k_M}(m_A||m_B||\alpha)$.

4. Il Bianconiglio verifica ϕ_A, controlla α e calcola la chiave $k = \alpha^b$.

Stavolta Alice sceglie quello che secondo lei è un valore ragionevole per $\#\mathbb{G}$, proponendolo al Bianconiglio. Il problema però è che il Bianconiglio non negozia i parametri, ma semplicemente accetta quelli proposti da Alice. In questo modo è semplice realizzare un attacco negazione del servizio (cf. Tab. 11.1) facendo generare al Bianconiglio primi giganteschi (così da "sabotare" di fatto l'uso del protocollo). Il Nonce N_A è utilizzato per legare i vari messaggi ed evitare attacchi "ri-uso" (infatti ora tutti gli autenticatori dipendono da N_A).

Comunque, il primo messaggio non è autenticato. Inoltre, la chiave k potrebbe avere qualche struttura algebrica dovuta al modo in cui è stata generata che consente più facilmente di distinguerla da un elemento casuale in $\mathbb{G}$. Il modo tipico di risolvere quest'ultimo problema è attraverso una funzione hash crittograficamente robusta $H(\cdot)$ (cf. Capitolo 4).

Versione finale. Siamo pronti per definire la versione "finale" del protocollo:

1. Alice sceglie il minimo valore per $\#\mathbb{G}$ (i.e., $s_A = \min \#\mathbb{G}$) ed un Nonce N_A. Invia al Bianconiglio $m_A = (s_A, N_A)$.

2. Il Bianconiglio calcola $s = \max\{s_A, s_B\}$, essendo s_B il suo valore minimo per $\#\mathbb{G}$. Quindi verifica s e, dato il parametro di sicurezza 1^n, genera $(\mathbb{G}, g, q)$, estrae $b \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\beta = g^b$. Invia quindi ad Alice (m_B, ϕ_B) dove $m_B = (\beta, \mathbb{G}, g, q)$ e $\phi_B \leftarrow \mathsf{Tag}_{k_M}(m_A||m_B)$.

3. Alice verifica ϕ_B, ed i valori $(\beta, \mathbb{G}, g, q)$. Estrae $a \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\alpha = g^a$ e $k = H(\beta^a)$. Invia (α, ϕ_A) al Bianconiglio, dove $\phi_A \leftarrow \mathsf{Tag}_{k_M}(m_A||m_B||\alpha)$.

4. Il Bianconiglio verifica ϕ_A, controlla α e calcola $k = H(\alpha^b)$.

Una verifica con successo di ϕ_A assicura al Bianconiglio che Alice è una sorgente viva del messaggio (cioè che il messaggio non è replicato) in quanto ϕ_A autentica β che è stato generato casualmente dal Bianconiglio. Gli attacchi "ri-uso" sono evitati grazie alle sfide casuali costituite dai Nonce N_A e β. Inoltre, stavolta entrambe le parti negoziano la scelta dei parametri e verificano che i parametri generati rispettino i valori negoziati. La verifica di $\mathbb{G}$ da parte di Alice, in particolare, significa controllare che $s_A - 1 \leq \|\#\mathbb{G}\| \leq \varepsilon \cdot s_A$ per una piccola costante ε.

In ogni caso, il protocollo non ha sicurezza dimostrabile ed è facile trasformare la descrizione ad alto livello data sopra, in un'implementazione fallace. Ci sono comunque alcune caratteristiche interessanti. La prima è che se per caso l'attaccante riuscisse ad un certo punto ad impadronirsi di una delle chiavi di sessione k generate dal protocollo, non avrebbe nessun indizio sulla chiave segreta principale k_M (con cui sono generati gli autenticatori) e sulle altre chiavi di sessione negoziate in passato (a patto che gli esponenti casuali siano generati correttamente in ogni istanza del protocollo). Alice ed il Bianconiglio possono quindi semplicemente ignorare k ed eseguire nuovamente il protocollo. Se invece l'attaccante si impadronisce della chiave segreta principale, il protocollo non può più essere eseguito (senza aver negoziato prima un'altra chiave k_M per Π_{MAC}), ma tutte le chiavi di sessione generate fino a quel momento restano comunque sicure.[69] Infine, osserviamo che il carico computazionale è equi-partito tra Alice ed il Bianconiglio.

11.2 Nozioni di sicurezza per autenticazione

Nei paragrafi successivi ci occuperemo in modo più specifico di *autenticazione*. In particolare, analizzeremo "scambi di mano" via via più evoluti per ottenere diverse forme di autenticazione.

Cerchiamo innanzitutto di astrarre. Si definisce autenticazione (dal greco, "reale" o "genuino") il processo tramite il quale si verifica la genuinità di qualcosa (o qualcuno). Una prima classificazione è in base a *cosa* si vuole autenticare:

- *Autenticazione di persone.* Anche detta identificazione. Si tratta di assicurarsi dell'identità della persona con cui si sta comunicando. Tipicamente è eseguita in tempo reale e deve essere mutua. In generale, come vedremo, tale processo può avvenire in base: a ciò che si conosce (ad esempio una chiave segreta), a ciò che si possiede (ad esempio una chiave fisica o una carta

[69] Tale proprietà è detta sicurezza perfetta in avanti (*perfect forward secrecy* in inglese).

di credito) oppure ad una caratteristica personale (ad esempio un'impronta digitale o, più in generale, un tratto biometrico).

- *Autenticazione di messaggio.* Riguarda l'autenticazione della sorgente del messaggio o della sua integrità. Abbiamo già discusso questa problematica nei Capitoli 7 e 8.

È bene osservare che, tipicamente, la linea di demarcazione tra queste due categorie non è ben definita.

Un'altra classificazione può essere in base a *dove* l'autenticazione ha luogo. Essenzialmente essa può essere:

- *Locale.* Quando è eseguita da un dispositivo locale a cui una certa entità sta cercando di accedere (questo è il caso, ad esempio, di un utente che digita il codice segreto del suo cellulare).

- *Diretta.* Quando è eseguita da un sistema remoto a cui una certa entità sta tentando di accedere (ad esempio un utente registrato che effettua l'accesso (*login*) su un sito web).

- *Indiretta.* Come al punto precedente, con la differenza che l'autenticazione è effettuata coinvolgendo un sistema remoto dedicato, diverso dall'applicazione a cui si vuole accedere.

- *Non-in-linea.* Come sopra, ma il sistema di autenticazione non necessita di essere chiamato in causa ogni volta. (Questo è il caso della verifica dei certificati o degli schemi di firma a chiave pubblica.)

Osserviamo che, in ogni caso, l'autenticazione è un processo *istantaneo*: esso è valido nel momento in cui è eseguito, non c'è alcuna certezza per il futuro! Per rendere il processo duraturo ci sono due possibilità: (i) ripetere il processo nel tempo, (ii) stabilire un canale sicuro (ad esempio attraverso uno scambio chiavi autenticato). [70]

In generale, modelleremo un protocollo d'autenticazione come un protocollo interattivo eseguito da due parti (Alice e lo Stregatto), nel seguito indicate con $\mathscr{P}$ (colui che si autentica) e $\mathscr{V}$ (colui che verifica), entrambe algoritmi PPT.[71]

[70] Non bisogna confondere l'autenticazione con l'*autorizzazione*. Quest'ultima, infatti, definisce quale risorsa o servizio può essere acceduto e per quale scopo un'entità *già autenticata* può utilizzarlo. In altre parole l'autorizzazione ha senso solo *dopo* che una procedura di autenticazione è stata portata a termine. Proprio per questo motivo autorizzazione *non* implica affatto autenticazione.

[71] L'uso della lettera $\mathscr{P}$ deriva dal fatto che colui che si autentica è detto "dimostratore" (*prover* in inglese).

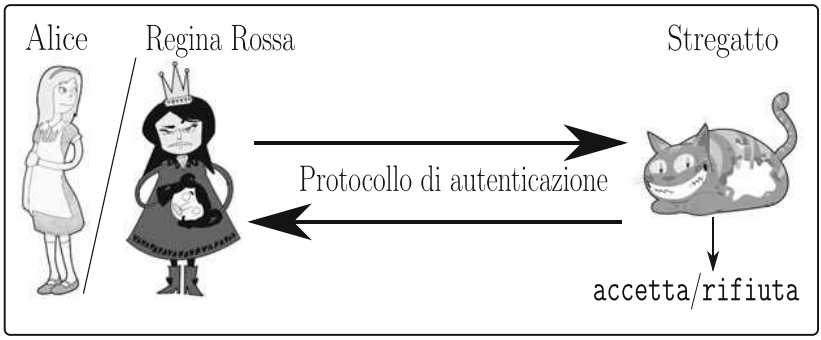


Fig. 11.3. Protocolli di autenticazione

Lo scenario di riferimento è mostrato in Fig. 11.3: intuitivamente un protocollo di autenticazione deve essere tale che la Regina Rossa non possa autenticarsi al posto di Alice!

Il protocollo di solito ha associato un algoritmo di generazione delle chiavi Gen, il quale prende come input il parametro di sicurezza n e restituisce una chiave condivisa (se si utilizzano tecniche simmetriche) oppure una coppia di chiavi pubblica/privata (se si utilizzano tecniche asimmetriche). Dopo l'esecuzione del protocollo $\mathscr{V}$ ritorna `accetta` oppure `rifiuta`, a seconda che l'autenticazione di $\mathscr{P}$ sia andata a buon fine oppure no. Diremo che il protocollo ha errore di correttezza α se per tutte le chiavi generate da $\mathsf{Gen}(1^n)$ un'esecuzione *onesta* del protocollo ritorna `accetta` con probabilità $1 - \alpha$.

Attacchi passivi. Diremo che un protocollo di autenticazione è sicuro contro attaccanti *passivi* se nessun attaccante PPT $\mathscr{A}$ può far si che $\mathscr{V}$ ritorni `accetta` con probabilità non trascurabile dopo aver osservato passivamente un certo numero di esecuzioni oneste tra $\mathscr{P}$ e $\mathscr{V}$. Più formalmente, diremo che il protocollo è (t, Q, ϵ)-sicuro contro attaccanti passivi se nessun attaccante PPT $\mathscr{A}$ eseguibile in tempo t e che osserva Q esecuzioni del protocollo, può far sì che $\mathscr{V}$ ritorni `accetta` con probabilità migliore di ϵ.

Attacchi attivi. In un attacco *attivo* $\mathscr{A}$ agisce in due fasi. Nella prima fase può interagire con $\mathscr{P}$ un numero polinomiale di volte impersonando $\mathscr{V}$ anche in modo concorrente.[72] Nella seconda fase $\mathscr{A}$ interagisce solo con $\mathscr{V}$ e ha un

[72] Si intende qui che $\mathscr{A}$ può lanciare più istanze di $\mathscr{P}$, anche in parallelo.

solo tentativo per impersonare $\mathscr{P}$. Diremo che un protocollo d'autenticazione è (t, Q, ϵ)-sicuro contro attaccanti attivi se nessun attaccante PPT $\mathscr{A}$ eseguibile in tempo t che effettua Q richieste a $\mathscr{P}$ nella prima fase, può far si che $\mathscr{V}$ ritorni `accetta` con probabilità migliore di ϵ.

Attacchi MiM. In un attacco MiM $\mathscr{A}$ può interagire in modo concorrente con un numero polinomiale di istanze sia di $\mathscr{P}$ che $\mathscr{V}$. In ciascuna istanza inoltre l'avversario viene a conoscenza della decisione di $\mathscr{V}$. Abbiamo già visto un esempio di tale attacco nel Paragrafo 11.1.

11.3 Il paradigma sfida e risposta

Un paradigma molto usato nei protocolli di autenticazione è il paradigma "sfida e risposta". L'interazione è composta da due messaggi: nel primo messaggio $\mathscr{V}$ invia una sfida a $\mathscr{P}$ e nel secondo messaggio $\mathscr{P}$ risponde alla sfida in modo che $\mathscr{V}$ sia in grado di capire se $\mathscr{P}$ possiede realmente le credenziali per essere autenticato. In questo paragrafo discuteremo le tecniche simmetriche ed asimmetriche che permettono di realizzare tale paradigma.

Tecniche simmetriche. Nel contesto simmetrico Alice ed il Bianconiglio condividono una chiave segreta k. Analizzeremo due protocolli basati su alcune primitive che abbiamo studiato nei capitoli precedenti: il primo attraverso un cifrario simmetrico (cf. Capitolo 5) ed il secondo attraverso un MAC (cf. Capitolo 7).

Cominciamo con il protocollo in Fig. 11.4. Sia $\Pi_E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un cifrario simmetrico con spazio dei messaggi $\mathcal{M}$, spazio dei testi cifrati $\mathcal{C}$ e

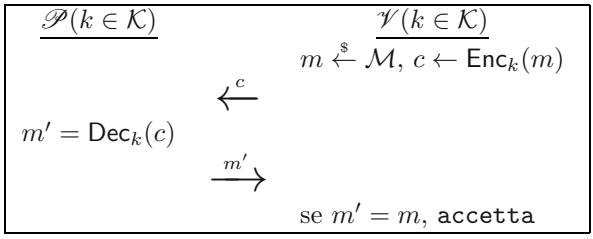


Fig. 11.4. Autenticazione attraverso un cifrario simmetrico

spazio delle chiavi $\mathcal{K}$. Il verificatore $\mathscr{V}$ sceglie un messaggio casuale $m \in \mathcal{M}$ ed invia a $\mathscr{P}$ la sfida $c \leftarrow \mathsf{Enc}_k(m)$. Quindi $\mathscr{P}$ decifra il crittotesto calcolando $m' = \mathsf{Dec}_k(c)$ ed invia m' a $\mathscr{V}$. Il verificatore ritorna `accetta` se e solo se m' coincide con la sfida iniziale m da lui scelta.

Teorema 11.1 (Sicurezza del protocollo di Fig. 11.4). *Se il cifrario* Π_E *è CCA-*$(t_{\mathsf{CCA}}, Q, \epsilon_{\mathsf{CCA}})$*-sicuro, il protocollo d'autenticazione in Fig. 11.4 è* (t, Q, ϵ)*-sicuro contro attaccanti attivi, dove*

$$t \approx t_{\mathsf{CCA}} \qquad \epsilon \leq \epsilon_{\mathsf{CCA}} + \frac{2Q+2}{\#\mathcal{C}}.$$

Dimostrazione. Dato un attaccante PPT $\mathscr{A}$ eseguibile in tempo t che attacca la sicurezza attiva del protocollo con vantaggio ϵ, possiamo costruire un avversario $\mathscr{B}$ che attacca la sicurezza CCA del cifrario Π_E. L'avversario $\mathscr{B}$ esegue l'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n)$, usando $\mathscr{A}$ come segue:

1. Lancia $\mathscr{A}(1^n)$.
2. Quando $\mathscr{A}$ si sostituisce a $\mathscr{V}$, ricevi la sfida c inviata da $\mathscr{A}$. Invoca l'oracolo di decifratura ottenendo $m = \mathsf{Dec}_{k_{\mathrm{AB}}}(c)$ ed invia m come risposta ad $\mathscr{A}$.
3. Quando $\mathscr{A}$ si sostituisce a $\mathscr{P}$ e prova ad autenticarsi, scegli i due messaggi $m_0, m_1 \xleftarrow{\$} \mathcal{M}$ ed inviali all'oracolo. Sia $c_b \leftarrow \mathsf{Enc}_k(m_b)$ il relativo crittotesto sfida. Sfida $\mathscr{A}$ usando c_b e ricevi la risposta m_b.
4. Se $m_b = m_0$ ritorna $b' = 0$, se $m_b = m_1$ ritorna $b' = 1$, altrimenti scegli b' a caso.

L'attaccante $\mathscr{B}$ non è in possesso della chiave di cifratura k, ma (a posteriori) usa il suo oracolo di decifratura per rispondere alle richieste di $\mathscr{A}$ nella prima fase. È facile vedere che la simulazione così ottenuta è perfetta e quindi $\mathscr{A}$ crede di attaccare un'istanza reale del protocollo. Nella seconda fase, allo stesso modo, il crittotesto c_b è calcolato cifrando un messaggio completamente casuale, come in una reale esecuzione del protocollo. In risposta all'ultima richiesta, $\mathscr{A}$ si aspetta di ricevere la decisione di $\mathscr{V}$; questa non può essere simulata da $\mathscr{B}$ (poiché $\mathscr{B}$ non conosce la chiave segreta). Tuttavia questo non è importante, perché $\mathscr{B}$ è in possesso della risposta m_b inviata da $\mathscr{A}$ e può utilizzare questa per scegliere b'. L'analisi che segue consente di esprimere la probabilità che $\mathscr{A}$ si autentichi (violando la sicurezza del protocollo) in funzione del vantaggio di $\mathscr{B}$ nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n)$.

Sia $\mathcal{E}_1$ l'evento in cui, nella prima fase dell'attacco, $\mathscr{A}$ ha inviato una sfida c uguale al crittotesto sfida c_b ricevuto nella seconda fase. Osserviamo che fintanto che $\mathcal{E}_1$ *non* si verifica, $\mathscr{B}$ non invoca mai l'oracolo di decifratura in corrispondenza del crittotesto sfida e quindi ha successo se $b' = b$. Pertanto, possiamo scrivere:

$$\begin{aligned} \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n) = 1\right] &= \mathbb{P}\left[b' = b \wedge \overline{\mathcal{E}_1}\right] \\ &\geq \mathbb{P}\left[b' = b\right] - \mathbb{P}\left[\mathcal{E}_1\right]. \end{aligned} \tag{11.1}$$

D'altra parte sia $\mathcal{E}_2$ l'evento che $\mathscr{A}$ si autentichi. Condizionando su tale evento e tenendo conto che per ipotesi $\mathbb{P}\left[\mathcal{E}_2\right] = \epsilon$, abbiamo dunque:

$$\begin{aligned} \mathbb{P}\left[b' = b\right] &= \mathbb{P}\left[b' = b \mid \mathcal{E}_2\right] \cdot \mathbb{P}\left[\mathcal{E}_2\right] \\ &\quad + \mathbb{P}\left[b' = b \mid \overline{\mathcal{E}_2}\right] \cdot (1 - \mathbb{P}\left[\mathcal{E}_2\right]) \\ &= 1 \cdot \epsilon + \left(\frac{1}{2} - \frac{1}{\#\mathcal{C} - 1}\right) \cdot (1 - \epsilon) \\ &= \frac{1}{2} - \frac{1}{\#\mathcal{C} - 1} + \left(\frac{1}{2} - \frac{1}{\#\mathcal{C} - 1}\right) \cdot (1 - \epsilon). \end{aligned}$$

Nel secondo passaggio abbiamo usato il fatto che quando $\mathscr{A}$ non si autentica $b' = b$ con probabilità $1/2$ meno la probabilità che casualmente la sfida di $\mathscr{A}$ sia proprio c_{1-b} (nel cui caso $b' = 1 - b \neq b$). Risolvendo infine per ϵ, troviamo:

$$\begin{aligned} \epsilon &= \frac{2(\#\mathcal{C} - 1)}{\#\mathcal{C} + 1} \cdot \left(\mathbb{P}\left[b' = b\right] - \frac{1}{2} + \frac{1}{\#\mathcal{C} - 1}\right) \\ &\leq 2 \cdot \mathbb{P}\left[b' = b\right] - 1 + \frac{2}{\#\mathcal{C} + 1} \\ &\leq [\text{usando l'Eq. (11.1)}] \\ &\leq 2 \cdot \left(\mathbb{P}\left[\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{SKE},\Pi_E}(\mathscr{B}, n) = 1\right] + \mathbb{P}\left[\mathcal{E}_1\right]\right) - 1 + \frac{2}{\#\mathcal{C} - 1} \\ &\leq [\text{siccome } \Pi_E \text{ è CCA-sicuro}] \\ &\leq 2 \cdot \left(\frac{1}{2} + \epsilon_{\mathsf{CCA}} + \mathbb{P}\left[\mathcal{E}_1\right]\right) - 1 + \frac{2}{\#\mathcal{C} - 1} \\ &\leq \epsilon_{\mathsf{CCA}} + 2 \cdot \mathbb{P}\left[\mathcal{E}_1\right] + \frac{2}{\#\mathcal{C} + 1} \\ &\leq [\text{ovviamente } \mathbb{P}\left[\mathcal{E}_1\right] = Q/(\#\mathcal{C})] \end{aligned}$$

$$\leq \epsilon_{\mathsf{CCA}} + 2 \cdot \frac{Q}{\#\mathcal{C}} + \frac{2}{\#\mathcal{C}}$$
$$= \epsilon_{\mathsf{CCA}} + \frac{2Q+2}{\#\mathcal{C}},$$

come volevasi dimostrare. □

È possibile rendere il protocollo sicuro contro attaccanti MiM aggiungendo un messaggio: prima $\mathscr{P}$ sceglie un Nonce N_P e lo invia a $\mathscr{V}$, quindi $\mathscr{V}$ costruisce la sfida cifrando la stringa $m||N_P$. Si veda [BCK98] per una dimostrazione.

Una seconda possibilità, sempre nel contesto simmetrico, è quella di usare un MAC. Sia $\Pi_M = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Vrfy})$ un codice autenticatore di messaggio con spazio dei testi in chiaro $\mathcal{M}$ e spazio delle chiavi $\mathcal{K}$. L'idea è che $\mathscr{V}$ sfidi $\mathscr{P}$ chiedendogli di produrre un autenticatore relativo ad un messaggio a caso scelto da $\mathscr{V}$. Siccome $\mathscr{V}$ conosce la chiave condivisa k, può verificare il MAC ed eventualmente autenticare $\mathscr{P}$. Il protocollo è mostrato in Fig. 11.5.

Teorema 11.2 (Sicurezza del protocollo di Fig. 11.5). *Se il MAC* Π_M *è* $(t_{\mathsf{MAC}}, Q, \epsilon_{\mathsf{MAC}})$*-ufcma, il protocollo in Fig. 11.5 è* (t, Q, ϵ)*-sicuro contro attaccanti attivi, dove*

$$t \approx t_{\mathsf{MAC}} \qquad \epsilon \leq \epsilon_{\mathsf{MAC}} + \frac{Q}{\#\mathcal{M}}.$$

Dimostrazione. Dato un attaccante PPT $\mathscr{A}$ eseguibile in tempo t che attacca la sicurezza attiva del protocollo con vantaggio ϵ, possiamo costruire un forgiatore PPT $\mathscr{F}$ che attacca la sicurezza del MAC Π_M. L'avversario $\mathscr{F}$ esegue l'esperimento $\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n)$, usando $\mathscr{A}$ come segue:

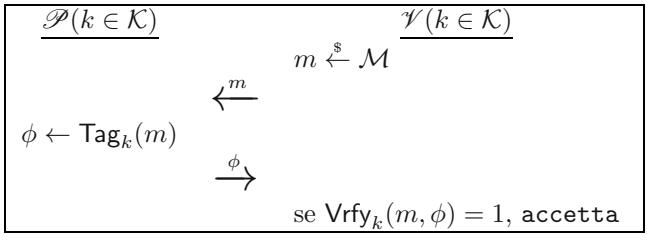


Fig. 11.5. Autenticazione attraverso un MAC

1. Lancia $\mathscr{A}(1^n)$.
2. Quando $\mathscr{A}$ si sostituisce a $\mathscr{V}$, ricevi la sfida m inviata da $\mathscr{A}$. Invoca l'oracolo per la generazione degli autenticatori ottenendo $\phi \leftarrow \mathsf{Tag}_k(m)$ ed invia ϕ come risposta ad $\mathscr{A}$.
3. Quando $\mathscr{A}$ si sostituisce a $\mathscr{P}$ e prova ad autenticarsi, scegli una sfida a caso $m^* \xleftarrow{\$} \mathcal{M}$ ed inviala ad $\mathscr{A}$. Sia ϕ^* la risposta di $\mathscr{A}$. Restituisci la forgiatura (m^*, ϕ^*).

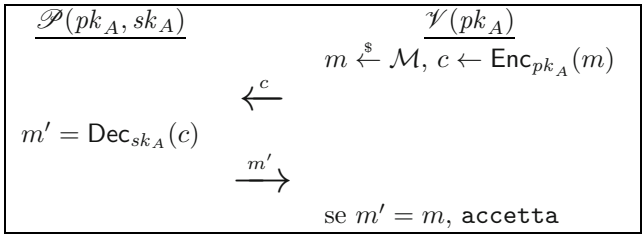
Non è difficile convincersi che la simulazione eseguita da $\mathscr{F}$ è perfetta, fintanto che il messaggio m^* scelto come sfida nella seconda fase dell'attacco non è uguale ad una dei messaggi m che $\mathscr{A}$ ha inviato nella prima fase dell'attacco (nel cui caso la forgiatura prodotta da $\mathscr{F}$ non è "fresca"). Indichiamo con $\mathcal{E}_1$ tale evento e sia $\mathcal{E}_2$ l'evento che $\mathscr{A}$ si autentichi. Abbiamo

$$\begin{aligned}
\epsilon &= \mathbb{P}[\mathcal{E}_2] = \mathbb{P}[\mathcal{E}_2 \wedge \mathcal{E}_1] + \mathbb{P}\left[\mathcal{E}_2 \wedge \overline{\mathcal{E}_1}\right] \\
&= \mathbb{P}[\mathcal{E}_2 \wedge \mathcal{E}_1] + \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n) = 1\right] \\
&\leq \mathbb{P}[\mathcal{E}_1] + \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n) = 1\right] \\
&\leq [\text{siccome } \mathscr{A} \text{ effettua } Q \text{ richieste}] \\
&\leq \frac{Q}{\#\mathcal{M}} + \mathbb{P}\left[\mathbf{Exp}^{\mathrm{ufcma}}_{\mathsf{MAC},\Pi_M}(\mathscr{F}, n) = 1\right] \\
&\leq [\text{siccome } \Pi_M \text{ è ufcma}] \\
&\leq \frac{Q}{\#\mathcal{M}} + \epsilon_{\mathsf{MAC}},
\end{aligned}$$

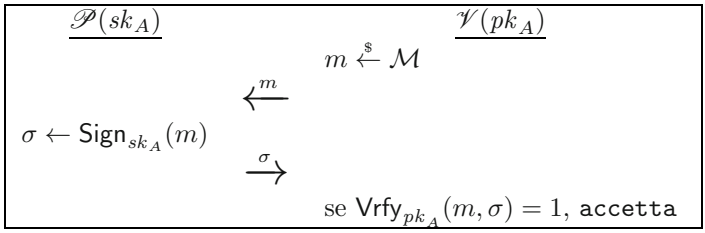
come volevasi dimostrare. □

Una semplice modifica del protocollo (sempre utilizzando due round) consente di ottenere sicurezza contro attaccanti MiM [BCK98] (cf. Esercizio 11.9).

Tecniche asimmetriche. Nel contesto asimmetrico Alice possiede una coppia di chiavi pubblica/privata (pk_A, sk_A) che utilizza per autenticarsi. Gli stessi protocolli definiti nel contesto simmetrico (cf. Fig. 11.6) possono essere realizzati usando (rispettivamente) un cifrario asimmetrico CCA-sicuro (cf. Definizione 6.4) ed uno schema di firma digitale universalmente inforgiabile contro attacchi a messaggio scelto (cf. Definizione 8.2). Gli schemi sono mostrati in Fig. 11.6, la prova di sicurezza è omessa.



(a) Autenticazione usando un cifrario asimmetrico



(b) Autenticazione usando una firma digitale

Fig. 11.6. Tecniche asimmetriche di autenticazione

Autenticazione mutua. Gli “scambi di mano” descritti finora sono unilaterali: mentre $\mathscr{P}$ si autentica, $\mathscr{V}$ non fornisce alcuna prova della sua identità. In alcuni contesti è importante che le due parti si autentichino a vicenda; in questo caso si parla di *autenticazione mutua*.

Cominciamo a discutere il contesto simmetrico. Un modo banale per ottenere autenticazione mutua è quello di ripetere i protocolli di Fig. 11.4 e 11.5 due volte, scambiando i ruoli di $\mathscr{P}$ e $\mathscr{V}$ nelle due esecuzioni. Cerchiamo di astrarre; sia $f_k(\cdot)$ una funzione che dipende dalla chiave segreta condivisa k (ad esempio, l’algoritmo di decifratura nel protocollo di Fig. 11.4 e l’algoritmo di generazione dei MAC nel protocollo di Fig. 11.5). In generale l’interazione tra le due parti avviene come di seguito: Il Bianconiglio sfida Alice con il messaggio m_B ed Alice risponde con $y_B = f_k(m_B)$ (il che richiede due messaggi). Quindi il Bianconiglio verifica la risposta ed eventualmente autentica Alice. A questo punto il processo si ripete a parti scambiate: Alice sfida il Bianconiglio con il messaggio m_A ed il Bianconiglio risponde con $y_A = f_k(m_A)$. Infine, Alice verifica la risposta ed eventualmente autentica il Bianconiglio. Ciò richiede dunque un totale di quattro messaggi.

Potremmo pensare di migliorare l'efficienza del protocollo richiedendo che Alice sfidi immediatamente il Bianconiglio[73] con m_A, il Bianconiglio quindi invia la risposta y_A insieme alla sua sfida m_B ed Alice risponde con y_B. Dunque, entrambi verificano la risposta della controparte. Questa soluzione, pur essendo più efficiente, è prona ad attacchi di tipo riflessione (cf. Tab. 11.1): si aprono diverse istanze del protocollo in parallelo e si usano le risposte di un'istanza per attaccarne un'altra. In particolare la Regina può sostituirsi inizialmente ad Alice ed inviare la sfida casuale m_A. Come indicato nel protocollo il Bianconiglio risponderà con $y_A = f_k(m_A)$ e sfiderà la Regina con m_B. Prima di inviare la risposta a questa sfida, la Regina apre una nuova istanza del protocollo (in parallelo) e nel primo messaggio stavolta sfida il Bianconiglio proprio con m_B. In questo modo, il Bianconiglio risponderà con un valore $y_B = f_k(m_B)$ (ed un'altra sfida casuale che può essere ignorata) che costituisce la risposta alla sfida m_B anche nella vecchia istanza del protocollo.

Esistono alcune contromisure per questo tipo di attacco. Essenzialmente, l'idea è quella di fare in modo che Alice ed il Bianconiglio facciano cose diverse. Ad esempio, nel calcolare le risposte alle sfide, il Bianconiglio potrebbe utilizzare una chiave derivata (diciamo la chiave $k + 1$). Un'altra possibilità è che ogni entità includa il proprio identificativo alle sfide, oppure Alice potrebbe scegliere sfide con caratteristiche diverse da quelle che sceglie il Bianconiglio. Notare che l'attacco di tipo riflessione non funziona nella versione del protocollo che utilizza quattro messaggi. (Da qui la regola che il primo ad autenticarsi deve essere colui che apre il protocollo.)

La stessa tecnica è utilizzabile nel contesto asimmetrico, dove stavolta ci sono due coppie di chiavi: la chiave pubblica/privata di Alice (pk_A, sk_A) e la chiave pubblica/privata del Bianconiglio (pk_B, sk_B). In questo caso la funzione $f(\cdot)$ può essere utilizzata in due modi: uno dipendente dalla chiave segreta (corrispondente agli algoritmi di decifratura e di generazione delle firme nei protocolli di Fig. 11.6) e l'altro dipendente dalla chiave pubblica (corrispondente agli algoritmi di cifratura e di verifica delle firme nei protocolli di Fig. 11.6). Osserviamo che in questo caso la versione efficiente del protocollo di autenticazione mutua non è soggetta ad attacchi di tipo riflessione, in quanto Alice ed il Bianconiglio fanno intrinsecamente cose diverse (in particolare, Alice usa sk_A mentre il Bianconiglio usa sk_B).

[73] Solitamente tutti i protocolli di autenticazione si aprono con un messaggio iniziale in cui in realtà si richiede di voler essere autenticati. In questo caso la sfida di Alice può essere inviata insieme a tale messaggio.

Il protocollo di Needham-Schröeder. Needham e Schröeder [NS78] hanno proposto due protocolli: il primo per ottenere autenticazione mutua usando tecniche asimmetriche, il secondo nel contesto dell'autenticazione mediata (cf. Paragrafo 11.5). Nel seguito, in accordo con la notazione standard nell'analisi dei protocolli attraverso i così detti metodi formali (di cui non ci occuperemo nel dettaglio), indicheremo con $\{m\}_k$ la cifratura del messaggio m con la chiave k (pubblica o privata che essa sia).

Il protocollo è mostrato in Fig. 11.7. Alice inizia l'interazione scegliendo un Nonce N_A ed inviando la cifratura con la chiave pubblica pk_B del Bianconiglio della stringa ottenuta concatenando il suo identificativo `Alice` seguito da N_A. Il Bianconiglio usa la sua chiave segreta per recuperare N_A, quindi estrae N_B a caso e cifra la stringa $N_A||N_B$ usando la chiave pubblica pk_A di Alice. Alice usa dunque la sua chiave segreta per recuperare N_B e risponde cifrando N_B con pk_B. Infine, il Bianconiglio recupera nuovamente N_B e controlla che questo sia coerente con il Nonce da lui scelto nel passo precedente del protocollo.

Qualche anno più tardi, nel 1995, Lowe [Low95] ha scoperto un attacco di tipo MiM. Supponiamo che la Regina $\mathscr{A}$ possegga una coppia di chiavi pubblica/privata $(pk_{\mathscr{A}}, sk_{\mathscr{A}})$ e che riesca a convincere Alice che la chiave pubblica del Bianconiglio è in realtà $pk_{\mathscr{A}}$. L'attacco avviene come segue:

1. Nella prima fase, $\mathscr{A}$ si sostituisce al Bianconiglio ricevendo il messaggio $\{\texttt{Alice}, N_A\}_{pk_{\mathscr{A}}}$ da Alice. La Regina usa $sk_{\mathscr{A}}$ per recuperare N_A.

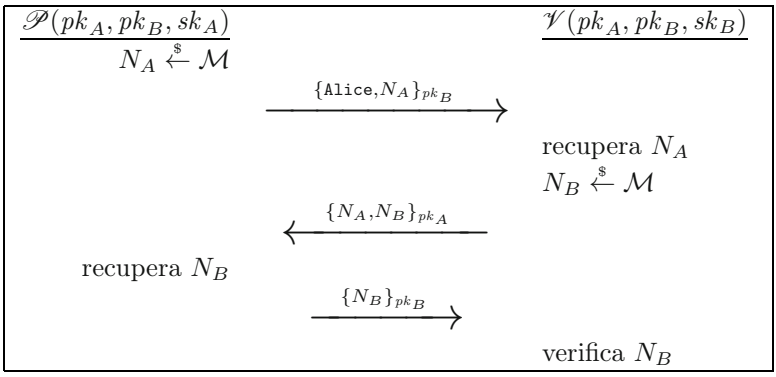


Fig. 11.7. Il protocollo di Needham-Schröeder per autenticazione mutua

2. Quindi $\mathscr{A}$ apre una nuova istanza del protocollo con il Bianconiglio, inviando $\{\texttt{Alice}, N_A\}_{pk_B}$. Il Bianconiglio decifra il messaggio recuperando N_A ed invia alla Regina (pensando si tratti di Alice) $\{N_A, N_B\}_{pk_A}$.

3. La Regina inoltra il messaggio ad Alice la quale lo decifra recuperando N_B ed inviando $\{N_B\}_{pk_{\mathscr{A}}}$ ad $\mathscr{A}$ (pensando si tratti del Bianconiglio).

4. Infine $\mathscr{A}$ può recuperare N_B ed inviare al Bianconiglio $\{N_B\}_{pk_B}$ facendogli credere di aver decifrato N_B ed autenticandosi come Alice.

Il protocollo di Needham-Schröeder-Lowe [Low96] evita questo attacco sostituendo il messaggio $\{N_A, N_B\}_{pk_A}$ con $\{\texttt{Bianconiglio}, N_A, N_B\}_{pk_A}$. Si veda [BP04] per un'analisi formale di sicurezza.

Stabilire un canale sicuro. Dopo che Alice ed il Bianconiglio si sono autenticati a vicenda possono scambiare una chiave di sessione (cf. Paragrafo 11.1). Le chiavi di sessione scambiate devono essere sempre diverse e non ricavabili da un attaccante che intercetti i messaggi necessari a condividerle.

Nel contesto a chiave pubblica, ad esempio, Alice può scegliere un valore k a caso, cifrarlo con la chiave pubblica del Bianconiglio pk_B ed aggiungere la stringa $c \leftarrow \mathsf{Enc}_{pk_B}(k)$ risultante ad uno dei messaggi usati durante il processo di autenticazione. Notare che, in questo modo, se la Regina riuscisse ad impersonare Alice potrebbe modificare arbitrariamente k, sostituendolo con un valore k' a sua scelta. Per evitare quest'attacco, Alice potrebbe ulteriormente firmare il messaggio $c \leftarrow \mathsf{Enc}_{pk_B}(k)$ con la sua chiave segreta, ottenendo $\sigma \leftarrow \mathsf{Sign}_{sk_A}(c)$, il che impedisce alla Regina di modificare k. (Questa non è altro che la versione asimmetrica del paradigma "prima cifra e poi autentica", che abbiamo incontrato nel Paragrafo 7.4.)

Osserviamo che se la Regina memorizza un'intera istanza del protocollo e poi si sostituisce al Bianconiglio può comunque recuperare k, compromettendo tutti i dati protetti attraverso tale chiave. Un modo per evitare anche quest'attacco è prevedere ad esempio che Alice invii $\mathsf{Enc}_{pk_B}(k_A)$ ed il Bianconiglio $\mathsf{Enc}_{pk_A}(k_B)$: la chiave condivisa sarà poi $k = k_A \oplus k_B$. Non c'è bisogno di sostituire entrambe le quantità, in quanto la Regina può rimpiazzarne solo una alla volta (ricordiamo che le due parti già si sono autenticate). Ora l'avversario deve impadronirsi di entrambi i nodi per ricavare la nuova chiave. La soluzione più robusta consiste nell'eseguire uno scambio di chiavi alla Diffie-Hellman autenticato (cf. Paragrafo 11.1).

11.4 I protocolli HB e HB$^+$

In alcuni contesti l'efficienza è un requisito fondamentale. Le soluzioni discusse nel paragrafo precedente hanno sicurezza dimostrabile, ma la loro efficienza in generale dipende dalla complessità delle primitive sottostanti. Nel 2001, Hopper e Blum [HB01] hanno proposto un protocollo d'autenticazione molto elegante basato sulla difficoltà del problema di imparare la parità in presenza di rumore (LPN, cf. la fine del Paragrafo 9.2). Il protocollo è così efficiente che, anche per valori realistici dei parametri in gioco, potrebbe essere eseguito da umani senza l'uso di un calcolatore. Più in generale, lo schema di Hopper e Blum è utile in tutti quei contesti in cui le risorse computazionali sono particolarmente limitate, come ad esempio l'autenticazione dei dispositivi a radio-frequenza (*Radio-Frequency IDentification*, RFID).

Il problema LPN. Siccome in precedenza non l'abbiamo fatto in modo esplicito, definiamo formalmente il problema LPN. Intuitivamente, nella sua versione computazionale, il problema LPN richiede di calcolare un vettore segreto $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n$ a partire da un certo numero di prodotti scalari (tra $\mathbf{s}$ ed alcuni vettori casuali di uguale dimensione) "sporcati" da rumore Bernoulliano.

Più precisamente, sia Ber_τ la distribuzione Bernoulliana con parametro $0 < \tau \leq 1/2$ (i.e., $\mathbb{P}[e = 1] = \tau$ se $e \xleftarrow{\$} \mathsf{Ber}_\tau$). Definiamo con $\Lambda_{\tau,n}(\mathbf{s})$ la distribuzione di probabilità su $\mathbb{Z}_2^n \times \mathbb{Z}_2$ ottenuta estraendo uniformemente $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n$ ed $e \xleftarrow{\$} \mathsf{Ber}_\tau$ e restituendo $(\mathbf{r}, \mathbf{r}^\mathsf{T} \cdot \mathbf{s} \oplus e)$.

Definizione 11.1 (Problema LPN computazionale). Diremo che il problema $\mathrm{LPN}_{\tau,n}$ computazionale è (t, Q, ϵ)-difficile se nessun attaccante PPT $\mathscr{A}$ eseguibile in tempo t, può determinare $\mathbf{s}$ con probabilità migliore di ϵ richiedendo Q campioni all'oracolo $\Lambda_{\tau,n}(\mathbf{s})$:

$$\mathbb{P}\left[\mathscr{A}^{\Lambda_{\tau,n}(\mathbf{s})}(1^n) = \mathbf{s} : \ \mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n\right] \leq \epsilon.$$

■

La complessità del problema $\mathrm{LPN}_{\tau,n}$ cresce al crescere di τ. Più in generale il problema è stato mostrato essere **NP**-completo [BMT78]. L'algoritmo più efficiente [BKW03; LF06] richiede t, Q dell'ordine $2^{\Theta(n \log n)}$.

Nella sua forma decisionale il problema LPN richiede di distinguere $\Lambda_{\tau,n}(\mathbf{s})$ dalla distribuzione uniforme su $\mathbb{Z}_2^{n+1}$ (indicata al solito con U_{n+1}).

Definizione 11.2 (Problema LPN decisionale). Diremo che il problema $\mathrm{LPN}_{\tau,n}$ decisionale è (t, Q, ϵ)-difficile se, per ogni attaccante PPT $\mathscr{D}$ che agisce in tempo t richiedendo Q campioni, si ha:

$$\left| \mathbb{P}\left[\mathscr{D}^{\Lambda_{\tau,n}(\mathbf{s})}(1^n) = 1 : \ \mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n \right] - \mathbb{P}\left[\mathscr{D}^{U_{n+1}}(1^n) = 1 \right] \right| \leq \epsilon.$$

■

Il Lemma 9.2 (cf. Paragrafo 9.2) mostra che risolvere la versione decisionale è equivalente a risolvere quella computazionale. Si veda anche [KS06a] per i parametri concreti della riduzione.

Il protocollo HB. La soluzione proposta da Hopper e Blum ricalca il paradigma sfida e risposta. Le due parti $\mathscr{P}$ e $\mathscr{V}$ condividono un segreto $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n$. Il verificatore $\mathscr{V}$ estrae un vettore $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^n$ e lo invia a $\mathscr{P}$, il quale risponde con $z = \mathbf{r}^\mathsf{T} \cdot \mathbf{s} \oplus e$ dove $e \xleftarrow{\$} \mathsf{Ber}_\tau$. Infine $\mathscr{V}$ ritorna `accetta` se e solo se $\mathbf{r}^\mathsf{T} \cdot \mathbf{s} = z$. Il protocollo, nella sua versione base, ha un errore di validità (i.e., la probabilità che $\mathscr{A}$ faccia sì che $\mathscr{V}$ ritorni `accetta` usando un valore di z a caso) pari ad $1/2$ ed errore di completezza (i.e., la probabilità che $\mathscr{V}$ ritorni `rifiuta` anche quando $\mathscr{P}$ è onesto) τ. Queste quantità possono essere ridotte ripetendo il protocollo in modo sequenziale (cioè eseguendo un passo base ℓ volte). In questo modo il protocollo ha anche sicurezza dimostrabile contro attaccanti passivi, come argomentato dagli stessi Hopper e Blum e mostrato più avanti in modo formale da Jules e Weis [JW05].

Per aumentare l'efficienza del protocollo vorremmo lanciare le diverse istanze in parallelo e non sequenzialmente. Il protocollo, nella sua versione parallela, è indicato con HB ed è rappresentato in Fig. 11.8. Ora $\mathscr{V}$ genera una matrice $\mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{n \times \ell}$ e la risposta prodotta da $\mathscr{P}$ è della forma $\mathbf{z} = \mathbf{R}^\mathsf{T} \cdot \mathbf{s} \oplus \mathbf{e}$ dove $\mathbf{e} \xleftarrow{\$} \mathsf{Ber}_\tau^\ell$. Il verificatore $\mathscr{V}$ accetta se e solo se al più una frazione τ' delle ℓ istanze parallele fallisce, ovvero se il peso di Hamming (cf. Definizione A.2) della stringa $\mathbf{z} \oplus \mathbf{R}^\mathsf{T} \cdot \mathbf{s}$ soddisfa $w_H\left(\mathbf{z} \oplus \mathbf{R}^\mathsf{T} \cdot \mathbf{s}\right) < \tau' \cdot \ell$. (Per essere concreti, nel seguito possiamo pensare $\tau' = \tau/2 + 1/4$.)

- `Parametri pubblici.` Il protocollo ha i seguenti parametri pubblici, tutti dipendenti dal parametro di sicurezza n.
  $0 < \tau < 1/2$: parametro di rumore
  $\tau < \tau' < 1/2$: soglia di accettazione
  ℓ: numero di ripetizioni parallele.

- `Generazione delle chiavi.` L'algoritmo di generazione delle chiavi restituisce la chiave condivisa $\mathbf{s} \leftarrow \mathsf{Gen}(1^n)$ con $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n$.
- `Protocollo.` Il protocollo è illustrato in Fig. 11.8.

Il protocollo HB ha sicurezza formale contro attacchi passivi, come mostrato per la prima volta da Katz e Shin [KS06a; KS06b; KSS10]. Mostriamo che gli errori di validità e completezza sono trascurabili. L'errore di completezza è la probabilità che un $\mathscr{P}$ onesto non venga autenticato. Questa non è altro che la probabilità $\alpha_{\tau,\ell}$ che un vettore $\mathbf{e} \xleftarrow{\$} \mathsf{Ber}_\tau^\ell$ (cioè ℓ campioni indipendenti prelevati dalla distribuzione Bernoulliana con parametro τ) contenga più di una frazione $\tau' \cdot \ell$ di valori 1. Invocando il limite di Chernoff (cf. Teorema A.5) è possibile mostrare (cf. Esercizio A.4) che esiste una costante c_τ (che dipende solo da τ) tale per cui:

$$\alpha_{\tau,\ell} = \mathbb{P}\left[w_H(\mathbf{e}) > \tau' \cdot \ell : \ \mathbf{e} \xleftarrow{\$} \mathsf{Ber}_\tau^\ell\right] < 2^{-c_\tau \cdot \ell}.$$

L'errore di validità è la probabilità che $\mathscr{A}$ faccia sì che $\mathscr{V}$ ritorni `accetta` inviando una risposta casuale. Questa non è altro che la probabilità $\alpha'_{\tau',\ell}$ che una stringa casuale $\mathbf{x} \xleftarrow{\$} \mathbb{Z}_2^\ell$ abbia peso di Hamming $\leq \tau' \cdot \ell$, dove $0 < \tau < \tau' < 1/2$. Ancora una volta, il limite di Chernoff implica (cf. Esercizio A.5) che esiste una costante c'_τ (che dipende solo da τ) tale per cui:

$$\alpha'_{\tau',\ell} = 2^{-\ell} \cdot \sum_{i=0}^{\lfloor \tau' \cdot \ell \rfloor} \binom{\ell}{i} < 2^{-c'_\tau \cdot \ell}.$$

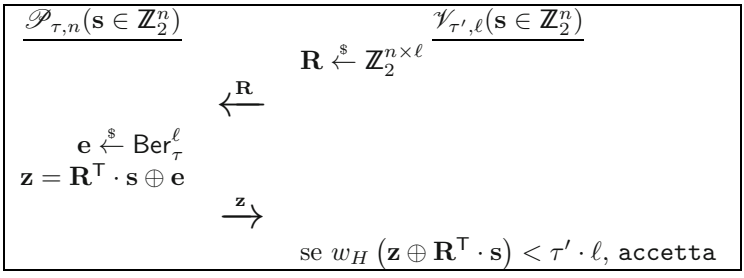


Fig. 11.8. Il protocollo HB, sicuro contro attaccanti passivi

Teorema 11.3 (Sicurezza del protocollo HB). *Se il problema* $\mathrm{LPN}_{\tau,n}$ *è* $(t_{\mathsf{LPN}}, \ell\cdot(Q+1), \epsilon_{\mathsf{LPN}})$*-difficile con* $0 < \tau \leq 1/2$*, il protocollo HB è* (t, Q, ϵ)*-sicuro contro attaccanti passivi, dove*

$$t = O(t_{\mathsf{LPN}}) \qquad \epsilon = \epsilon_{\mathsf{LPN}} + 2^{-\Theta(\ell)}.$$

Dimostrazione. Mostriamo che se esiste un attaccante PPT $\mathscr{A}$ in grado di violare la sicurezza del protocollo HB attraverso un attacco passivo con probabilità ϵ, allora possiamo usare $\mathscr{A}$ per costruire un attaccante PPT $\mathscr{D}$ in grado di risolvere la versione decisionale del problema $\mathrm{LPN}_{\tau,n}$ con probabilità ϵ_{LPN} come nell'enunciato del teorema. L'attaccante $\mathscr{D}$ ha accesso ad un oracolo che restituisce stringhe in $\mathbb{Z}_2^{n+1}$ e deve distinguere se tale oracolo è $\Lambda_{\tau,n}(\mathbf{s})$ (per un qualche $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^n$) oppure la distribuzione uniforme U_{n+1}. Per fare ciò, $\mathscr{D}$ usa $\mathscr{A}$ come segue:

1. Ogni volta in cui $\mathscr{A}$ richiede di osservare passivamente un'esecuzione onesta del protocollo HB, interroga ℓ volte l'oracolo ottenendo coppie di elementi $(\mathbf{R}[i], z_i)$ con $i = 1, \ldots, \ell$. Posto $\mathbf{R} = (\mathbf{R}[1], \ldots, \mathbf{R}[\ell])$ e $\mathbf{z} = (z_1, \ldots, z_\ell)$, ritorna ad $\mathscr{A}$ la coppia $(\mathbf{R}, \mathbf{z})$.

2. Quando $\mathscr{A}$ tenta di impersonare $\mathscr{P}$, interroga nuovamente l'oracolo per ℓ volte ottenendo coppie di elementi $(\mathbf{R}'[i], z'_i)$ con $i = 1, \ldots, \ell$. Sfida quindi $\mathscr{A}$ con $\mathbf{R}' = (\mathbf{R}'[1], \ldots, \mathbf{R}'[\ell])$ ed ottieni in cambio la risposta $\mathbf{z}''$.

3. Sia $\mathbf{z}' = (z'_1, \ldots, z'_\ell)$. Ritorna 1 se e solo se $w_H(\mathbf{z}' \oplus \mathbf{z}'') \leq 2\tau' \cdot \ell$.

Dobbiamo distinguere due casi.

L'oracolo di $\mathscr{D}$ è U_{n+1}. In questo caso $\mathscr{A}$ vede solamente stringhe uniformemente casuali. Quindi la stringa $\mathbf{z}' \oplus \mathbf{z}''$ è a distribuzione uniforme in $\mathbb{Z}_2^n$. Invocando il limite di Chernoff, ne segue che per una costante c''_τ (che dipende solo da τ) la probabilità con cui $\mathscr{D}$ restituisce 1 è esattamente $2^{-\ell} \cdot \sum_{i=0}^{2\lfloor \tau' \cdot \ell \rfloor} \binom{\ell}{i} < 2^{-c''_\tau \cdot \ell}$.

L'oracolo di $\mathscr{D}$ è $\Lambda_{\tau,n}(\mathbf{s})$. In questo caso la simulazione eseguita da $\mathscr{D}$ nella prima fase dell'attacco è perfetta. Indichiamo con $\mathbf{z}^* = (\mathbf{R}')^{\mathsf{T}} \cdot \mathbf{s}$ il vettore contenente i valori dei prodotti scalari (non affetti da rumore) $z_i^* = (\mathbf{R}'[i])^{\mathsf{T}} \cdot \mathbf{s}$. Siccome per ipotesi $\mathscr{A}$ impersona $\mathscr{P}$ con probabilità ϵ, ne segue che con probabilità almeno ϵ i vettori $\mathbf{z}''$ e $\mathbf{z}^*$ differiscono in al più $\tau' \cdot \ell$ valori. D'altra parte, siccome il vettore $\mathbf{z}'$ in questo caso soddisfa $\mathbf{z}' = (\mathbf{R}')^{\mathsf{T}} \cdot \mathbf{s} \oplus \mathbf{e}$ con $\mathbf{e} \xleftarrow{\$} \mathsf{Ber}_\tau^\ell$, il vettore $\mathbf{z}'$ è distribuito come in una risposta onesta di $\mathscr{P}$ nel

protocollo e quindi (poiché il protocollo ha errore di completezza $\alpha_{\tau,\ell}$) $\mathbf{z}'$ e $\mathbf{z}^*$ differiscono in al più $\tau' \cdot \ell$ valori con probabilità al più $\alpha_{\tau,\ell}$. Ma allora la probabilità che $\mathbf{z}'$ e $\mathbf{z}''$ differiscano in al più $2\tau' \cdot \ell$ valori è almeno $\epsilon - \alpha_{\tau,\ell}$.

Mettendo tutto insieme, abbiamo trovato:

$$\left| \mathbb{P}\left[\mathscr{D}^{\Lambda_{\tau,n}(\mathbf{s})}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{U_{n+1}}(1^n) = 1\right] \right| \geq \epsilon - \alpha_{\tau,\ell} - 2^{-c''_\tau \cdot \ell},$$

e quindi $\mathscr{D}$ può risolvere il problema $\mathrm{LPN}_{\tau,n}$ decisionale almeno con probabilità $\epsilon_{\mathsf{LPN}} = \epsilon - \alpha_{\tau,\ell} - 2^{-c''_\tau \cdot \ell} = \epsilon - 2^{-\Theta(\ell)}$, come richiesto. □

La dimostrazione porta ad un risultato utile solo quando $\tau < \tau' < 1/4$, poiché quando $\tau \geq 1/4$ risulta $2\tau' \cdot \ell \geq 2\tau \cdot \ell \geq \ell/2$ e quindi $2^{-\ell} \sum_{i=0}^{2\lfloor \tau' \cdot \ell \rfloor} \binom{\ell}{i} \geq 1/2$. In altri termini, in questo caso $\mathscr{D}$ ritorna 1 con alta probabilità anche quando l'oracolo è U_{n+1}. Si veda [KS06b; KSS10] per una dimostrazione nel caso in cui $\tau' \geq 1/4$.

Il protocollo HB$^+$. Il protocollo HB è insicuro contro attaccanti attivi. Ad esempio, $\mathscr{A}$ potrebbe sostituirsi a $\mathscr{V}$ e quindi sfidare $\mathscr{P}$ ripetutamente con lo stesso $\mathbf{R} = (\mathbf{R}[1], \ldots, \mathbf{R}[\ell])$. Prendendo la maggioranza delle risposte relative ad ogni valore $\mathbf{R}[i]^\mathsf{T} \cdot \mathbf{s}$, se i vettori in $\mathbf{R}$ sono linearmente indipendenti, è possibile recuperare $\mathbf{s}$ con probabilità alta. Per contrastare questo tipo di attacchi, Jules e Weis [JW05] hanno proposto la seguente estensione.

Le due parti condividono due segreti $\mathbf{s}_1, \mathbf{s}_2 \in \{0,1\}^n$. L'idea di base è che stavolta $\mathscr{P}$ cominci l'interazione inviando a $\mathscr{V}$ un vettore $\mathbf{r}_1 \xleftarrow{\$} \mathbb{Z}_2^n$; quindi $\mathscr{V}$ sceglie una sfida casuale $\mathbf{r}_2 \xleftarrow{\$} \mathbb{Z}_2^n$. Infine $\mathscr{P}$ risponde con $z = \mathbf{r}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{r}_2^\mathsf{T} \cdot \mathbf{s}_2 \oplus e$, dove $e \leftarrow \mathsf{Ber}_\tau$. In questo caso $\mathscr{V}$ ritorna `accetta` se e solo se $z = \mathbf{r}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{r}_2^\mathsf{T} \cdot \mathbf{r}_2$. Ancora una volta gli errori di validità e completezza possono essere ridotti ripetendo l'interazione base in modo sequenziale o parallelo. La versione sequenziale è stata analizzata da Juel e Weils [JW05] i quali hanno mostrato che il protocollo è sicuro contro attaccanti attivi. La versione parallela, indicata nel seguito con HB$^+$, è stata analizzata da Katz e Shin [KS06a] ed è descritta di seguito.

- `Parametri pubblici.` Il protocollo ha i seguenti parametri pubblici, tutti dipendenti dal parametro di sicurezza n.
  - $0 < \tau < 1/2$: parametro di rumore
  - $\tau < \tau' < 1/2$: soglia di accettazione
  - ℓ: numero di ripetizioni parallele.

- `Generazione delle chiavi.` L'algoritmo di generazione delle chiavi restituisce le chiavi condivise $\mathbf{s}_1, \mathbf{s}_2 \leftarrow \mathsf{Gen}(1^n)$ con $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathbb{Z}_2^n$.
- `Protocollo.` Il protocollo è illustrato in Fig. 11.9.

Gli errori di validità e completezza sono gli stessi del protocollo HB.

Teorema 11.4 (Sicurezza del protocollo HB$^+$). *Se il problema* $\mathrm{LPN}_{\tau,n}$ *è* $(t_{\mathsf{LPN}}, \ell \cdot Q, \epsilon_{\mathsf{LPN}})$*-difficile con* $\tau < \tau' < 1/4$*, il protocollo HB*$^+$ *è* (t, Q, ϵ)*-sicuro contro attaccanti passivi, dove*

$$t = O(t_{\mathsf{LPN}}) \qquad \epsilon^2 = \epsilon_{\mathsf{LPN}} + 2^{-\Theta(\ell)}.$$

Dimostrazione. Mostriamo che se esiste un attaccante PPT $\mathscr{A}$ in grado di violare la sicurezza del protocollo HB$^+$ con probabilità ϵ attraverso un attacco attivo, allora possiamo usare $\mathscr{A}$ per costruire un attaccante PPT $\mathscr{D}$ in grado di risolvere la versione decisionale del problema $\mathrm{LPN}_{\tau,n}$ con probabilità ϵ_{LPN} come nell'enunciato del teorema. L'attaccante $\mathscr{D}$ ha accesso ad un oracolo che restituisce stringhe in $\mathbb{Z}_2^{n+1}$ e deve distinguere se tale oracolo è $\Lambda_{\tau,n}(\mathbf{s}_1)$ (per un qualche $\mathbf{s}_1 \xleftarrow{\$} \mathbb{Z}_2^n$) oppure la distribuzione uniforme U_{n+1}. Per fare ciò, $\mathscr{D}$ usa $\mathscr{A}$ come segue:

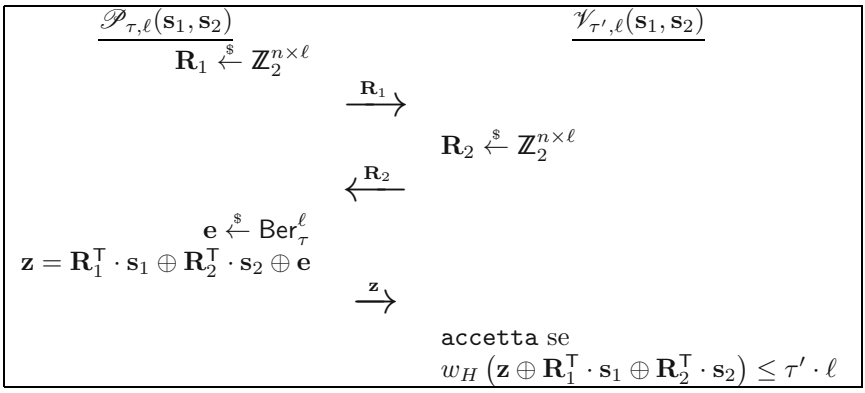


Fig. 11.9. Il protocollo HB$^+$, sicuro contro attaccanti attivi

1. Estrai $\mathbf{s}_2 \xleftarrow{\$} \mathbb{Z}_2^n$.
2. Nella prima fase dell'attacco, quando $\mathscr{A}$ (al posto di $\mathscr{V}$) vuole interagire con $\mathscr{P}$, interroga l'oracolo ℓ volte, ottenendo $(\mathbf{R}_1[i], z_i)$ con $i = 1, \ldots, \ell$. Invia quindi ad $\mathscr{A}$ la matrice $\mathbf{R}_1 = (\mathbf{R}_1[1], \ldots, \mathbf{R}_1[\ell])$. Quando $\mathscr{A}$ risponde con la matrice $\mathbf{R}_2$, calcola la risposta $\overline{\mathbf{z}} = \mathbf{R}_2^{\mathsf{T}} \cdot \mathbf{s}_2 \oplus \mathbf{z}$ avendo posto $\mathbf{z} = (z_1, \ldots, z_\ell)$.
3. Nella seconda fase dell'attacco, quando $\mathscr{A}$ (al posto di $\mathscr{P}$) tenta di autenticarsi, ricevi la sfida di $\mathscr{A}$, della forma $\mathbf{R}_1 = (\mathbf{R}_1[1], \ldots, \mathbf{R}_1[\ell])$. Estrai quindi $\mathbf{R}'_2 \xleftarrow{\$} \mathbb{Z}_2^{n \times \ell}$ e sfida $\mathscr{A}$ con $\mathbf{R}'_2 = (\mathbf{R}'_2[1], \ldots, \mathbf{R}'_2[\ell])$. Sia $\mathbf{z}' = (z'_1, \ldots, z'_\ell)$ la risposta di $\mathscr{A}$.
4. Riavvolgi $\mathscr{A}$ (al punto in cui ha già inviato $\mathbf{R}_1$) e sfidalo con $\mathbf{R}''_2 \xleftarrow{\$} \mathbb{Z}_2^{n \times \ell}$ della forma $\mathbf{R}''_2 = (\mathbf{R}''_2[1], \ldots, \mathbf{R}''_2[\ell])$. Sia $\mathbf{z}'' = (z''_1, \ldots, z''_\ell)$ la risposta di $\mathscr{A}$.
5. Poni $\mathbf{z}^{\oplus} = \mathbf{z}' \oplus \mathbf{z}''$. Siano $\mathbf{R}^*[i] = \mathbf{R}'_2[i] \oplus \mathbf{R}''_2[i]$ ed $\mathbf{R}^* = (\mathbf{R}^*[1], \ldots, \mathbf{R}^*[\ell])$. Calcola $\mathbf{z}^* = (\mathbf{R}^*)^{\mathsf{T}} \cdot \mathbf{s}_2$. Restituisci 1 se e solo se $\mathbf{z}^{\oplus}$ e $\mathbf{z}^*$ differiscono in al più $2\tau' \cdot \ell$ valori.

Dobbiamo distinguere due casi.

L'oracolo di $\mathscr{D}$ è U_{n+1}. Nella prima fase dell'attacco i valori z_i sono bit casuali. Ne segue che la risposta simulata $\overline{\mathbf{z}}$ è anch'essa casuale ed in particolare indipendente da $\mathbf{s}_2$.

Inoltre, anche il vettore $\mathbf{z}^{\oplus}$ è completamente casuale. Siccome poi i vettori $\mathbf{R}^*[i]$ (con $i = 1, \ldots, \ell$) sono indipendenti ed uniformemente distribuiti, essi sono anche linearmente indipendenti con probabilità $2^\ell / 2^n$.[74] Quando ciò accade, il vettore $\mathbf{z}^*$ è anch'esso a distribuzione uniforme e la probabilità che i due vettori $\mathbf{z}^*$ e $\mathbf{z}^{\oplus}$ differiscano in al più $2\tau' \cdot \ell$ valori è esattamente $2^{-\ell} \cdot \sum_{i=0}^{2\lfloor \tau' \cdot \ell \rfloor} \binom{\ell}{i} < 2^{-c''_\tau \cdot \ell}$ (usando il limite di Chernoff). Quindi, in questo caso, $\mathscr{D}$ restituisce 1 con probabilità al più $2^\ell / 2^n + 2^{-c''_\tau \cdot \ell}$.

[74] Ciò può essere mostrato come segue. Siano $\{\mathbf{r}_i\}_{i=1}^{\ell}$ vettori casuali in $\mathbb{Z}_2^n$. Indichiamo con $\mathcal{E}_i$ l'evento che il vettore $\mathbf{r}_i$ sia linearmente *dipendente* da uno dei vettori $\mathbf{r}_1, \ldots, \mathbf{r}_{i-1}$ (per $i = 0$ tale evento è l'evento in cui $\mathbf{r}_1$ sia il vettore tutto nullo). Siccome il sottospazio definito da $i - 1$ vettori ha dimensione al più 2^{i-1}, la probabilità di $\mathcal{E}_i$ è al più $2^{i-1}/2^n$. Per il limite dell'unione:

$$\mathbb{P}\left[\bigvee_{i=1}^{\ell} \mathcal{E}_i\right] \leq 2^{-n} \sum_{i=0}^{\ell-1} 2^i < \frac{2^\ell}{2^n}.$$

L'oracolo di $\mathscr{D}$ è $\Lambda_{\tau,n}(\mathbf{s}_1)$. In questo caso la simulazione della prima fase è perfetta. Sia ω la randomicità usata per simulare la prima fase dell'attacco (i.e., i vettori $\mathbf{s}_1, \mathbf{s}_2$, la randomicità di $\mathscr{A}$ e la randomicità usata per rispondere alle richieste di $\mathscr{A}$). Per un ω fissato sia ϵ_ω la probabilità, presa sulla scelta casuale dei vettori $\mathbf{R}_2[1], \ldots, \mathbf{R}_2[\ell]$ in $\mathbf{R}_2$, che $\mathscr{A}$ venga autenticato da $\mathscr{V}$. La probabilità che $\mathscr{A}$ risponda correttamente ad entrambe le sfide $\mathbf{R}'_2$ ed $\mathbf{R}''_2$ è quindi ϵ_ω^2. Mediando ed applicando la disuguaglianza di Jensen,[75] otteniamo:

$$\mathbb{E}_\omega\left[\epsilon_\omega^2\right] \geq \left(\mathbb{E}_\omega\left[\epsilon_\omega\right]\right)^2 = \epsilon^2.$$

Supponiamo dunque che $\mathscr{A}$ risponda correttamente ad entrambe le sfide $\mathbf{R}'_2$ ed $\mathbf{R}''_2$. Ciò significa che il vettore $\mathbf{z}'$ differisce in al più $\tau' \cdot \ell$ valori dal vettore delle risposte corrette:

$$\begin{aligned}\boldsymbol{\xi}' &= \mathbf{R}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus (\mathbf{R}'_2)^\mathsf{T} \cdot \mathbf{s}_2 \\ &= \left(\mathbf{R}_1[1]^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{R}'_2[1]^\mathsf{T} \cdot \mathbf{s}_2, \ldots, \mathbf{R}_1[\ell]^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{R}'_2[\ell]^\mathsf{T} \cdot \mathbf{s}_2\right);\end{aligned}$$

allo stesso modo, il vettore $\mathbf{z}''$ differisce in al più $\tau' \cdot \ell$ valori dal vettore delle risposte corrette:

$$\begin{aligned}\boldsymbol{\xi}'' &= \mathbf{R}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus (\mathbf{R}''_2)^\mathsf{T} \cdot \mathbf{s}_2 \\ &= \left(\mathbf{R}_1[1]^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{R}''_2[1]^\mathsf{T} \cdot \mathbf{s}_2, \ldots, \mathbf{R}_1[\ell]^\mathsf{T} \cdot \mathbf{s}_1 \oplus \mathbf{R}''_2[\ell]^\mathsf{T} \cdot \mathbf{s}_2\right).\end{aligned}$$

Ma allora il vettore $\mathbf{z}' \oplus \mathbf{z}'' = \mathbf{z}^\oplus$ differisce in al più $2\tau' \cdot \ell$ valori dal vettore:

$$\begin{aligned}\boldsymbol{\xi}' \oplus \boldsymbol{\xi}'' &= \mathbf{R}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus (\mathbf{R}'_2)^\mathsf{T} \cdot \mathbf{s}_2 \oplus \mathbf{R}_1^\mathsf{T} \cdot \mathbf{s}_1 \oplus (\mathbf{R}''_2)^\mathsf{T} \cdot \mathbf{s}_2 \\ &= \left(\mathbf{R}'_2[1]^\mathsf{T} \cdot \mathbf{s}_2 \oplus \mathbf{R}''_2[1]^\mathsf{T} \cdot \mathbf{s}_2, \ldots, \mathbf{R}'_2[\ell]^\mathsf{T} \cdot \mathbf{s}_2 \oplus \mathbf{R}''_2[\ell]^\mathsf{T} \cdot \mathbf{s}_2\right) \\ &= \left((\mathbf{R}'_2[1] \oplus \mathbf{R}''_2[1])^\mathsf{T} \cdot \mathbf{s}_2, \ldots, (\mathbf{R}'_2[\ell] \oplus \mathbf{R}''_2[\ell])^\mathsf{T} \cdot \mathbf{s}_2\right) \\ &= (\mathbf{R}^*)^\mathsf{T} \cdot \mathbf{s}_2 = \mathbf{z}^*.\end{aligned}$$

Dunque, in questo caso, $\mathscr{D}$ restituisce 1 con probabilità ϵ^2.

Mettendo tutto insieme, abbiamo trovato:

$$\left|\mathbb{P}\left[\mathscr{D}^{\Lambda_{\tau,n}(\mathbf{s}_1)}(1^n) = 1\right] - \mathbb{P}\left[\mathscr{D}^{U_{n+1}}(1^n) = 1\right]\right| \geq \epsilon^2 - \frac{2^\ell}{2^n} - 2^{-c''_\tau \cdot \ell},$$

e quindi $\mathscr{D}$ può risolvere il problema $\mathrm{LPN}_{\tau,n}$ decisionale almeno con probabilità $\epsilon_{\mathsf{LPN}} = \epsilon^2 + 2^{-\Theta(\ell)}$, come richiesto. □

[75] Nel contesto della teoria delle probabilità tale disuguaglianza, dimostrata da Jensen nel 1906, afferma che se X è una variabile aleatoria ed f una funzione convessa, allora

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$

Nel nostro caso la funzione f è il quadrato di una quantità in $[0, 1]$.

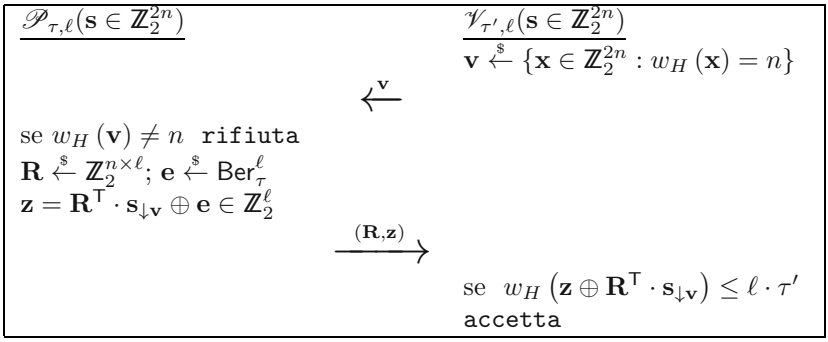


Fig. 11.10. Sicurezza attiva in due round usando il problema LPN

Attacchi MiM. Un attacco attivo non è tanto potente quanto un attacco MiM. In effetti il protocollo HB$^+$ non è resistente a tale tipo di attacchi [GRS05]. L'avversario si pone in mezzo tra $\mathscr{P}$ e $\mathscr{V}$, sceglie un vettore $\boldsymbol{\Delta} \in \mathbb{Z}_2^n$ e, ogni qualvolta $\mathscr{V}$ invia una sfida $\mathbf{r}_2$ la sostituisce con $\mathbf{r}_2 \oplus \boldsymbol{\Delta}$. Alla fine del round ottiene la risposta $z = \mathbf{r}_1^\mathsf{T} \cdot \mathbf{s}_2 \oplus (\mathbf{r}_2 \oplus \boldsymbol{\Delta})^\mathsf{T} \cdot \mathbf{s}_2 \oplus e$. La decisione di $\mathscr{V}$ può essere utilizzata per recuperare un singolo bit di $\mathbf{s}_2$. Infatti, se l'autenticazione (al termine degli ℓ round del protocollo) ha successo (risp. fallisce), con alta probabilità abbiamo $\boldsymbol{\Delta}^\mathsf{T} \cdot \mathbf{s}_2 = 0$ (risp. $\boldsymbol{\Delta}^\mathsf{T} \cdot \mathbf{s}_2 = 1$). In questo modo è possibile recuperare $\mathbf{s}_2$ bit per bit cambiando $\boldsymbol{\Delta}$ progressivamente. Una volta noto $\mathbf{s}_2$, è possibile recuperare $\mathbf{s}_1$ usando lo stesso attacco mostrato per HB.

Varie soluzioni, per lo più euristiche, sono state proposte per risolvere il problema, si vedano ad esempio [BCD06; MP07; BC08; GRS08b; GRS08a; GRS08c; Gol+08; LMM08; OOV08]. Purtroppo tutte le soluzioni proposte non hanno una prova formale di sicurezza o addirittura si sono rivelate essere insicure (ad esempio la soluzione proposta in [BC08], violata da Frumkin e Shamir [FS09] subito dopo essere stata pubblicata).

Rimangono aperti due problemi fondamentali. Il primo è quello di trovare un protocollo a due soli round che sia sicuro contro attaccanti attivi.[76] Il secondo problema è quello di costruire un protocollo sicuro rispetto attacchi MiM usando LPN. Recentemente queste domande hanno trovato una risposta [Kil+11].

[76] Potremmo pensare di modificare il protocollo HB$^+$ richiedendo che Alice invii la sfida $\mathbf{R}_1$ insieme al terzo messaggio. Purtroppo, non è noto se tale modifica mantenga la sicurezza del protocollo, in quanto la dimostrazione di Katz e Shin in questo caso non è più valida; ciò dipende dal fatto che non è più utile riavvolgere l'avversario nella riduzione.

$\mathscr{P}$ e $\mathscr{V}$ condividono un segreto $\mathbf{s} \in \mathbb{Z}_2^{2n}$. L'idea è che il Bianconiglio inizi l'interazione fissando un sottoinsieme (di dimensione n) del vettore segreto $\mathbf{s}$: il vettore $\mathbf{s}_{\downarrow \mathbf{v}} \in \mathbb{Z}_2^n$ (corrispondente agli elementi di $\mathbf{s}$ in cui $\mathbf{v}$ vale 1) sarà quindi usato come in una normale esecuzione del protocollo HB con la differenza che sarà ora Alice a scegliere la matrice $\mathbf{R}$. Il protocollo è mostrato in Fig. 11.10 e ha sicurezza attiva se il problema LPN è difficile.

Per ottenere sicurezza contro attaccanti MiM è necessario rendere il protocollo in Fig. 11.10 non-interattivo, ottenendo così un MAC e quindi un protocollo in due round con sicurezza contro attacchi MiM (cf. Paragrafo 11.3). Si rimanda direttamente a [Kil+11] per i dettagli.

11.5 Autenticazione mediata

Come abbiamo già discusso, ci sono due problemi fondamentali relativi all'utilizzo delle tecniche simmetriche in crittografia: la distribuzione iniziale dei segreti e la loro gestione e/o memorizzazione. Sebbene sia impossibile risolvere completamente il primo problema, esistono soluzioni intermedie che offrono diversi vantaggi. Una di queste prevede l'utilizzo di una terza parte fidata detta centro di distribuzione delle chiavi (*Key Distribution Center*, KDC).

Immaginiamo un'azienda che voglia consentire ad ogni coppia di impiegati di comunicare in modo sicuro. La soluzione banale è prevedere che ogni coppia di utenti condivida una chiave segreta, il che comporta un numero gigantesco di chiavi da condividere e memorizzare in modo sicuro. Una soluzione alternativa è prevedere che tutti gli impiegati si fidino di una terza entità, il KDC appunto, che si occuperà di far comunicare di volta in volta gli utenti in modo sicuro.

Un possibile meccanismo è il seguente. Innanzitutto è necessario che ciascun utente condivida una chiave con il KDC. Diciamo che Alice condivide la chiave k_A ed il Bianconiglio la chiave k_B. Se il Bianconiglio vuole comunicare con Alice, questi invia un messaggio al KDC esprimendo il suo volere. Il KDC sceglie a caso una chiave di sessione k e la invia alle due parti, cifrandola rispettivamente con le chiavi condivise k_A e k_B. Le due parti legittime possono così estrarre la chiave di sessione, da usare poi nella loro conversazione privata. Al termine della conversazione le due parti cancellano la chiave di sessione, in quanto possono sempre riferirsi al KDC per ottenerne una nuova. Questo è solo un esempio (tra l'altro non completamente sicuro) di *autenticazione mediata*, l'oggetto del presente paragrafo.

Prima di descrivere alcuni protocolli in questo contesto, discutiamo vantaggi e svantaggi relativi a tale approccio. I vantaggi sono che: (i) ogni utente deve

memorizzare una sola chiave segreta, vale a dire la chiave che condivide con il KDC,[77] (ii) un nuovo utente deve semplicemente condividere una chiave con il KDC, senza che gli altri utenti facciano niente. Dunque il problema della condivisione delle chiavi è quantomeno semplificato, mentre il problema della memorizzazione è praticamente risolto. D'altra parte: (i) il KDC è il punto debole del sistema, in quanto se viene attaccato con successo compromette la sicurezza dell'intero sistema, (ii) il KDC è un unico punto di fallimento, nel senso che se viene messo fuori uso il sistema diventa inutilizzabile. Una soluzione che mitiga questi problemi è l'uso di KDC distribuiti (anche se ciò crea qualche complicazione relativamente alla gestione di nuovi utenti).

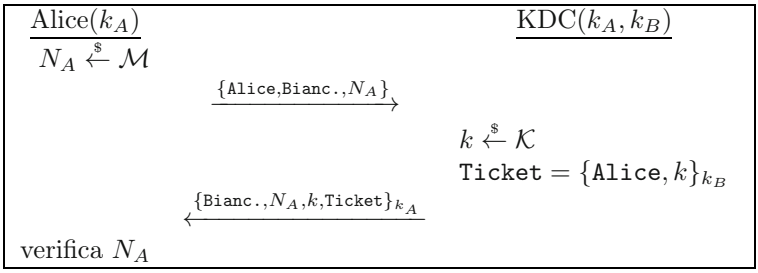
Il protocollo di Needham-Schröeder. Needham e Schröeder [NS78], hanno realizzato anche un protocollo per ottenere autenticazione mediata. Siano `Alice` e `Bianconiglio` le identità di Alice e del Bianconiglio e k_A (risp. k_B) la chiave segreta condivisa tra il KDC ed Alice (risp. Bianconiglio). Il protocollo è rappresentato in Fig. 11.11.

Alice inizia il processo d'autenticazione contattando il KDC, inviando il suo identificativo (`Alice`), l'identificativo della persona con cui vuole comunicare (`Bianconiglio`) ed un Nonce N_A. Il KDC genera la chiave di sessione k per le due parti ed un "biglietto da visita" ottenuto cifrando l'identificativo `Alice` e la chiave k con la chiave k_B condivisa da KDC e Bianconiglio (i.e., $\texttt{Ticket} = \{\texttt{Alice}, k\}_{k_B}$). Risponde quindi inviando una cifratura con chiave k_A della stringa $\texttt{Bianconiglio}, k, \texttt{Ticket}$. In questo modo Alice può decifrare e recuperare la chiave condivisa k_{AB} ed il "biglietto da visita" con cui ora può presentarsi al Bianconiglio (inviando anche un Nonce N_B). Quest'ultimo può decifrare la stringa `Ticket` e recuperare a sua volta k_{AB}. Infine il Bianconiglio si autentica usando k e il Nonce N_B inviato da Alice.

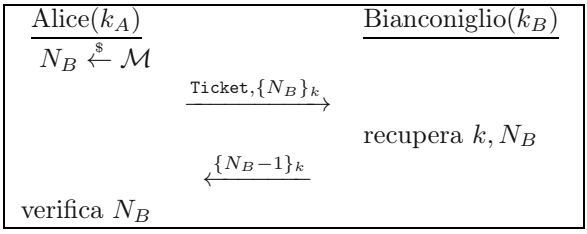
Il Nonce N_A dimostra in qualche modo la "vitalità" del KDC ad Alice. L'identificativo `Bianconiglio` nel secondo messaggio impedisce alla Regina di scambiare l'identità del Bianconiglio con la sua. Siccome solo il Bianconiglio può decifrare la stringa `Ticket` e recuperare k il quarto messaggio rassicura Alice sull'identità del Bianconiglio. Quest'ultimo a sua volta è sicuro dell'identità di Alice perché è contenuta nel "biglietto da visita".

Il protocollo è vulnerabile ad un attacco di tipo ri-uso (cf. Tab. 11.1), come evidenziato da Denninz e Sacco [DS81]. Supponiamo che la Regina intercetti una *vecchia* chiave di Alice j_A. (Alice potrebbe aver cambiato chia-

[77] Invece il KDC ne deve memorizzare tante quanti sono gli utenti del sistema, dunque tale nodo andrà adeguatamente protetto.



(a) L'interazione tra Alice ed il KDC

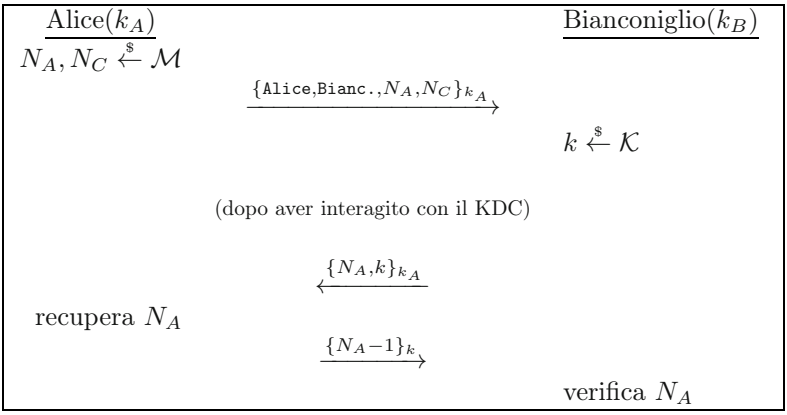


(b) L'interazione tra Alice ed il Bianconiglio

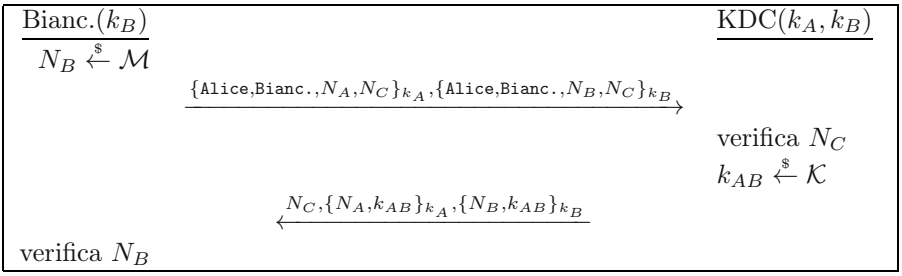
Fig. 11.11. Il protocollo di Needham-Schröeder per autenticazione mediata

ve, proprio perché pensa che j_A sia compromessa.) Supponiamo inoltre che la Regina abbia registrato una vecchia sessione del protocollo, eseguita con la chiave j_A. In particolare conosce $\{N_A, \texttt{Bianconiglio}, j_{AB}, \texttt{Ticket}\}_{j_A}$ essendo $\texttt{Ticket} = \{\texttt{Alice}, j_{AB}\}_{k_B}$. La Regina può allora *scavalcare il KDC* e presentarsi direttamente al Bianconiglio con un "biglietto da visita" valido.

È possibile evitare l'attacco modificando il protocollo [NS87]. Stavolta Alice comincia immediatamente a dialogare con il Bianconiglio, inviando il suo identificativo `Alice`. Il Bianconiglio la sfida con $\{N_0\}_{k_B}$ essendo N_0 un Nonce. Per poter recuperare il Nonce N_0 Alice dovrà rivolgersi al KDC. Invia quindi gli identificativi `Alice`, `Bianconiglio`, un Nonce N_A ed $\{N_0\}_{k_B}$ al KDC (in pratica come nel primo messaggio del protocollo di Fig. 11.11 ma con in più il messaggio $\{N_0\}_{k_B}$). A questo punto il KDC genera la chiave di sessione k insieme al "biglietto da visita" $\texttt{Ticket} = \{\texttt{Alice}, k, N_0\}_{k_B}$ (che ora include N_0) e risponde ad Alice con il messaggio $\{N_A, \texttt{Bianconiglio}, k, \texttt{Ticket}\}_{k_A}$. Ottenuto il "biglietto da visita", Alice può ricontattare il Bianconiglio presentandosi con la stringa `Ticket` ed una sfida $\{N_B\}_k$ (essendo N_B un Nonce). Il Bian-



(a) L'interazione tra Alice ed il Bianconiglio



(b) L'interazione tra Bianconiglio e KDC

Fig. 11.12. Il protocollo Otway-Rees per autenticazione mediata

coniglio infine può decifrare la stringa `Ticket` (controllando che contenga N_0) recuperare k_{AB} e rispondere alla sfida legando la risposta ad N_B.

Il protocollo Otway-Rees. Un altro esempio di autenticazione mediata è costituito dal protocollo di Otway-Rees [OR87], mostrato in Fig. 11.12.

Alice si presenta prima al Bianconiglio, estraendo due Nonce N_A, N_C e cifrando con chiave k_A il messaggio ottenuto concatenando le stringhe `Alice`, `Bianconiglio`, N_A ed N_C. Per poter rispondere alla sfida il Bianconiglio deve contattare il KDC. Estrae quindi un Nonce N_B, cifra le stringhe `Alice`, `Bianconiglio`, N_B ed N_C con la sua chiave k_B ed invia il crittotesto al KDC,

insieme al messaggio $\{N_A, N_C, \texttt{Alice}, \texttt{Bianconiglio}\}_{k_A}$ ricevuto da Alice al passo precedente. Il KDC controlla che il valore di N_C sia lo stesso nei due messaggi, quindi estrae la chiave di sessione k_{AB} ed invia al Bianconiglio N_C, $\{N_A, k\}_{k_A}$ ed $\{N_B, k\}_{k_B}$. Il Bianconiglio può quindi inoltrare $\{N_A, k\}_{k_A}$ ad Alice, la quale recupera N_A e si autentica legando la sua risposta al Nonce N_A ed alla chiave di sessione k_{AB}.

Il valore N_C deve essere impredicibile, altrimenti esiste il seguente attacco. Supponiamo che N_C sia l'output di un contatore e sia X l'ultimo valore usato. La Regina invia al Binaconiglio il messaggio $\{X + 1, \texttt{Alice}, \texttt{Bianconiglio}, \star\}$ essendo $\star$ una stringa casuale. Successivamente memorizza il messaggio $m_B = \{N_B, X + 1, \texttt{Alice}, \texttt{Bianconiglio}\}_{k_B}$ che il Bianconiglio invia al KDC. Quando Alice inizia la sua sessione, userà $N_C = X + 1$ inviando al Bianconiglio il messaggio $\{X + 1, \texttt{Alice}, \texttt{Bianconiglio}, m_A\}$, dove $m_A = \{N_A, X + 1, \texttt{Alice}, \texttt{Bianconiglio}\}_{k_A}$. La Regina può così intercettare m_A ed inviare al KDC i messaggi m_A, m_B. Il KDC risponderà con $\{N_A, k\}_{k_A}$ che la Regina può inoltrare ad Alice, autenticandosi al posto del Bianconiglio.

Esercizi

Esercizio 11.1. Dare una definizione di sicurezza passiva per un protocollo di scambio chiavi.
(Suggerimento: formalizzare il fatto che un avversario, dopo aver osservato un'esecuzione onesta del protocollo, non sia in grado di distinguere la chiave risultante da una chiave casuale.)

Esercizio 11.2. Consideriamo il seguente protocollo interattivo Π^* per cifrare un messaggio con un cifrario simmetrico $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Inizialmente Alice ed il Bianconiglio eseguono un protocollo di scambio chiavi Π' con sicurezza passiva (cf. Esercizio 11.1) per condividere una chiave k. Quindi Alice calcola $c \leftarrow \mathsf{Enc}_k(c)$ ed invia il risultato al Bianconiglio.

1. Formulare una definizione di sicurezza IND per il cifrario interattivo Π^*.
2. Mostrare che se Π è IND-sicuro e se $\mathsf{Gen}(1^n)$ restituisce una chiave casuale in $\mathcal{K} = \{0,1\}^n$, allora lo schema Π^* soddisfa la definizione data al punto precedente.

Esercizio 11.3. Supponiamo che Alice ed il Bianconiglio eseguano il protocollo di scambio chiavi di Diffie-Hellman, utilizzando il gruppo $\mathbb{G}$, con ordine 11, contenente i quadrati modulo 23.

1. Mostrare che $g = 4$ genera un gruppo di ordine 11.
2. Alice sceglie $a = 6$ ed il Bianconiglio $b = 9$. Determinare tutti i valori scambiati nell'esecuzione del protocollo e derivare la chiave condivisa.

Esercizio 11.4. Descrivere un'opportuna variante del protocollo di Diffie-Hellman sulle curve ellittiche.

Esercizio 11.5. Questo esercizio mostra come estendere il protocollo Diffie-Hellman al caso di tre parti. Supponiamo che Alice, il Bianconiglio ed il Cappellaio vogliano condividere un segreto. Siano $\mathbb{G}, \mathbb{G}_T$ gruppi tali che esiste una mappa bilineare $\Xi : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, con $|\mathbb{G}| = |\mathbb{G}_T| = p$ dove p è un primo e g è un generatore di $\mathbb{G}$. Alice, il Bianconiglio ed il Cappellaio, scelgono indipendentemente segreti $a, b, c \in \mathbb{Z}_p$ ed inviano agli altri i valori g^a, g^b, g^c. Alice, dunque, calcola $e(g^b, g^c)$ ed eleva il risultato all'esponente a. Il Bianconiglio ed il Cappellaio calcolano invece (rispettivamente)

$$\Xi(g^a, g^c)^b \qquad\qquad \Xi(g^a, g^b)^c.$$

1. Determinare la chiave k condivisa e spiegare perché, in effetti, k è la stessa per tutti.

2. Quale ipotesi computazionale è necessaria, affinché il protocollo abbia sicurezza passiva?

3. Descrivere una variante del protocollo sulle curve ellittiche.

Esercizio 11.6. Mostrare che se il problema DDH è difficile in $\mathbb{G}$, allora il protocollo di scambio chiavi alla Diffie-Hellman di Fig. 11.1 è sicuro contro attacchi passivi.

Esercizio 11.7. Consideriamo il seguente schema di identificazione a chiave pubblica. La chiave pubblica è $N = pq$ dove $p, q \equiv 3 \pmod 4$. Il dimostratore $\mathscr{P}$ conosce la fattorizzazione di N. Sia $\mathbb{J}_N^{+1} \subseteq \mathbb{Z}_N^*$ l'insieme di elementi in $\mathbb{Z}_N^*$ con simbolo di Jacobi $+1$.

Il verificatore $\mathscr{V}$ sceglie un elemento casuale $y \xleftarrow{\$} \mathbb{J}_N^{+1}$ (ricordiamo che ciò può essere fatto efficientemente, senza conoscere la fattorizzazione di N) ed invia y a $\mathscr{P}$. Quindi $\mathscr{P}$ controlla se y oppure $-y$ è un residuo quadratico modulo N (ricordiamo che, nel caso considerato, uno ed uno solo tra y e $-y$ è un residuo quadratico modulo N) e risponde inviando la radice quadrata x di tale valore. Il verificatore controlla che $x^2 = \pm y \bmod N$.

1. Dimostrare che se è difficile fattorizzare N, lo schema è sicuro contro attacchi passivi.

2. Mostrare come si può fattorizzare N tramite un attacco attivo.

Esercizio 11.8. Consideriamo la seguente variante del protocollo in Fig. 11.4. Le due parti condividono la chiave k del cifrario simmetrico $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Il verificatore $\mathscr{V}$ sceglie un messaggio $m \xleftarrow{\$} \mathcal{M}$, calcola $c \leftarrow \mathsf{Enc}_k(m)$ ed invia c come sfida. Quindi $\mathscr{P}$ calcola $m' = \mathsf{Dec}_k(c)$ e risponde con m. Infine, il verificatore controlla che $m' = m$.

1. Dire se il protocollo ha sicurezza passiva quando Π è CPA-sicuro.

2. Dire se il protocollo ha sicurezza attiva quando Π è CCA-sicuro.

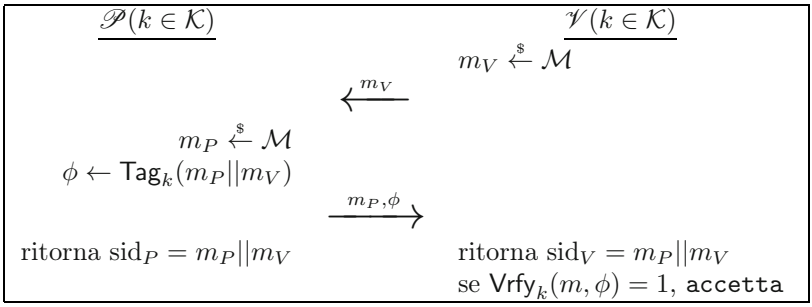
Stabilire in quali condizioni e contro quali attacchi il protocollo descritto è sicuro.

Esercizio 11.9. Consideriamo il protocollo di Fig. 11.5. Osserviamo che un attaccante MiM che attacca una data sessione del protocollo, può semplicemente inter-allacciare i messaggi scambiati da $\mathscr{P}$ e $\mathscr{V}$ fino a quando il verificatore non accetta. Sebbene questo non sia un vero e proprio attacco, in quanto di

fatto $\mathscr{P}$ sta interagendo con $\mathscr{V}$, bisogna in qualche modo considerare questo scenario quando si definisce la sicurezza MiM. Per evitare questo problema si usano *identificativi di sessione pubblici*: al termine del protocollo $\mathscr{P}$ e $\mathscr{V}$ ritornano un identificativo di sessione sid e l'autenticazione ha successo se e solo se $\mathrm{sid}_P = \mathrm{sid}_V$.

1. Dare una definizione formale di sicurezza MiM, richiedendo che l'attaccante abbia successo se riesce a convincere il verificatore in una sessione "fresca" (quindi per un valore di sid nuovo), oppure se riesce a confondere il verificatore facendo ritornare lo stesso identificativo di sessione a due diverse istanze di $\mathscr{P}$.

2. Considerare la versione seguente versione modificata del protocollo in Fig. 11.5:



Mostrare che se Π_M è $(t_{\mathsf{MAC}}, Q, \epsilon_{\mathsf{MAC}})$-ufcma, allora il protocollo è (t, Q, ϵ)-sicuro contro attacchi MiM, dove

$$t \approx t_{\mathsf{MAC}} \qquad \epsilon \leq \epsilon_{\mathsf{MAC}} + \frac{Q}{\#\mathcal{M}} + \frac{Q^2 - Q}{2 \cdot (\#\mathcal{M})}.$$

Letture consigliate

[Aca+10] Tolga Acar, Mira Belenkiy, Mihir Bellare e David Cash. "Cryptographic Agility and Its Relation to Circular Encryption". In: *EUROCRYPT*. 2010, pp. 403–422.

[BC08] Julien Bringer e Hervé Chabanne. "Trusted-HB: A Low-Cost Version of HB+ Secure Against Man-in-the-Middle Attacks". In: *IEEE Transactions on Information Theory* 54.9 (2008), pp. 4339–4342.

[BCD06] Julien Bringer, Hervé Chabanne e Emmanuelle Dottax. "HB++: A Lightweight Authentication Protocol Secure against Some Attacks". In: *SecPerU*. 2006, pp. 28–33.

[BCK98] Mihir Bellare, Ran Canetti e Hugo Krawczyk. "A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract)". In: *STOC*. 1998, pp. 419–428.

[Bir+91] Ray Bird, Inder S. Gopal, Amir Herzberg, Philippe A. Janson, Shay Kutten, Refik Molva e Moti Yung. "Systematic Design of Two-Party Authentication Protocols". In: *CRYPTO*. 1991, pp. 44–61.

[BKW03] Avrim Blum, Adam Kalai e Hal Wasserman. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *J. ACM* 50.4 (2003), pp. 506–519.

[BMT78] Elwyin R. Berlekamp, Robert J. McEliece e Henk C. A. van Tilborg. "On the Inherent Intractability of Certain Coding Problems". In: *IEEE Transactions on Information Theory* 24 (1978), pp. 384–386.

[BP04] Michael Backes e Birgit Pfitzmann. "A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol". In: *IEEE Journal on Selected Areas in Communications* 22.10 (2004).

[BR93] Mihir Bellare e Phillip Rogaway. "Entity Authentication and Key Distribution". In: *CRYPTO*. 1993, pp. 232–249.

[Car65] Lewis Carroll. *Alice's Adventures in Wonderland*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1865.

[CK01] Ran Canetti e Hugo Krawczyk. "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels". In: *EUROCRYPT*. 2001, pp. 453–474.

[CK02] Ran Canetti e Hugo Krawczyk. "Universally Composable Notions of Key Exchange and Secure Channels". In: *EUROCRYPT*. 2002, pp. 337–351.

[DH76] Whitfield Diffie e Martin E. Hellman. "New Directions in Cryptography". In: *IEEE Trans. Inform. Theory* 22 (nov. 1976), pp. 644–654.

[DOW92] Whitfield Diffie, Paul C. Van Oorschot e Michael J. Wiener. "Authentication and authenticated key exchanges". In: *Designs, Codes, and Cryptography* 2.2 (giu. 1992), pp. 107–125.

[DS81] Dorothy E. Denning e Giovanni Maria Sacco. “Timestamps in Key Distribution Protocols”. In: *Commun. ACM* 24.8 (1981), pp. 533–536.

[FS03] Niels Ferguson e Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, 2003.

[FS09] Dmitry Frumkin e Adi Shamir. *Un-Trusted-HB: Security Vulnerabilities of Trusted-HB*. Cryptology ePrint Archive, Report 2009/044. `http://eprint.iacr.org/`. 2009.

[GKR10] Rosario Gennaro, Hugo Krawczyk e Tal Rabin. “Okamoto-Tanaka Revisited: Fully Authenticated Diffie-Hellman with Minimal Overhead”. In: *ACNS*. 2010, pp. 309–328.

[Gol+08] Zbigniew Golebiewski, Krzysztof Majcher, Filip Zagorski e Marcin Zawada. *Practical Attacks on HB and HB+ Protocols*. Cryptology ePrint Archive, Report 2008/241. `http://eprint.iacr.org/`. 2008.

[GRS05] Henri Gilbert, Matt Robshaw e Herve Sibert. *An Active Attack Against HB+ - A Provably Secure Lightweight Authentication Protocol*. Cryptology ePrint Archive, Report 2005/237. `http://eprint.iacr.org/`. 2005.

[GRS08a] Henri Gilbert, Matthew J. B. Robshaw e Yannick Seurin. “Good Variants of HB+ Are Hard to Find”. In: *Financial Cryptography*. 2008, pp. 156–170.

[GRS08b] Henri Gilbert, Matthew J. B. Robshaw e Yannick Seurin. “HB# : Increasing the Security and Efficiency of HB+”. In: *EUROCRYPT*. 2008, pp. 361–378.

[GRS08c] Henri Gilbert, Matthew J. B. Robshaw e Yannick Seurin. “How to Encrypt with the LPN Problem”. In: *ICALP (2)*. 2008, pp. 679–690.

[HB01] Nicholas J. Hopper e Manuel Blum. “Secure Human Identification Protocols”. In: *ASIACRYPT*. 2001, pp. 52–66.

[JW05] Ari Juels e Stephen A. Weis. “Authenticating Pervasive Devices with Human Protocols”. In: *CRYPTO*. 2005, pp. 293–308.

[Kil+11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain e Daniele Venturi. “Efficient Authentication from Hard Learning Problems”. In: *EUROCRYPT*. 2011, pp. 7–26.

[Kra05] Hugo Krawczyk. “HMQV: A High-Performance Secure Diffie-Hellman Protocol”. In: *CRYPTO*. 2005, pp. 546–566.

[KS06a] Jonathan Katz e Ji Sun Shin. “Parallel and Concurrent Security of the HB and HB^+ Protocols”. In: *EUROCRYPT*. 2006, pp. 73–87.

[KS06b] Jonathan Katz e Adam Smith. *Analyzing the HB and HB+ Protocols in the “Large Error” Case*. Cryptology ePrint Archive, Report 2006/326. `http://eprint.iacr.org/`. 2006.

[KSS10] Jonathan Katz, Ji Sun Shin e Adam Smith. “Parallel and Concurrent Security of the HB and HB+ Protocols”. In: *J. Cryptology* 23.3 (2010), pp. 402–421.

[Law+03] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas e Scott A. Vanstone. “An Efficient Protocol for Authenticated Key Agreement”. In: *Des. Codes Cryptography* 28.2 (2003), pp. 119–134.

[LF06] Éric Levieil e Pierre-Alain Fouque. “An Improved LPN Algorithm”. In: *SCN*. 2006, pp. 348–359.

[LMM08] Xuefei Leng, Keith Mayes e Konstantinos Markantonakis. “HB-MP+ Protocol: An Improvement on the HB-MP Protocol”. In: *IEEE International Conference on RFID – IEEE RFID 2008* (2008), pp. 118–124.

[Low95] Gavin Lowe. “An Attack on the Needham-Schroeder Public-Key Authentication Protocol”. In: *Inf. Process. Lett.* 56.3 (1995), pp. 131–133.

[Low96] Gavin Lowe. “Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR”. In: *TACAS*. 1996, pp. 147–166.

[MP07] Jorge Munilla e Alberto Peinado. “HB-MP: A further step in the HB-family of lightweight authentication protocols”. In: *Computer Networks* 51.9 (2007), pp. 2262–2267.

[NS78] Roger M. Needham e Michael D. Schroeder. “Using Encryption for Authentication in Large Networks of Computers”. In: *Commun. ACM* 21.12 (1978), pp. 993–999.

[NS87] Roger M. Needham e Michael D. Schroeder. “Authentication Revisited”. In: *Operating Systems Review* 21.1 (1987), p. 7.

[OOV08] Khaled Ouafi, Raphael Overbeck e Serge Vaudenay. “On the Security of HB# against a Man-in-the-Middle Attack”. In: *ASIACRYPT*. 2008, pp. 108–124.

[OR87] David J. Otway e Owen Rees. “Efficient and Timely Mutual Authentication”. In: *Operating Systems Review* 21.1 (1987), pp. 8–10.

CAPITOLO

12

Password in crittografia

Urlò così forte che Alice non poté fare a meno di dire: «Zitto! Lo sveglierai con tutto questo rumore!». «Be', che parli a fare di svegliarlo» disse Dammelo, «se non sei altro che una delle cose del suo sogno». «Io sono vera!» disse Alice e iniziò a piangere. «Non diventerai neanche un pochino più vera piangendo» osservò Dimmelo; «non c'è niente da piangere».

Lewis Carroll, Attraverso lo Specchio e quel che Alice vi trovò [Car71]

Come abbiamo visto nel Capitolo 11 esistono diversi protocolli di autenticazione con sicurezza dimostrabile per scambio di chiavi e/o autenticazione (mutua), utilizzando informazioni condivise nella forma di chiavi crittografiche *ad alto contenuto entropico*. È naturale chiedersi cosa si può sperare di ottenere quando l'informazione condivisa tra le due parti ha *basso contenuto entropico*, ad esempio se essa è costituita da una parola scelta in un dizionario con "pochi" elementi.

Una *password* è una stringa di simboli appartenente ad un alfabeto finito, tipicamente usata nel contesto dell'*autenticazione di persone*. (In effetti due esseri umani non sono assolutamente in grado di memorizzare, ad esempio, una chiave RSA.) Siccome è semplice rappresentare una password come una stringa binaria o con un numero, in questo capitolo faremo riferimento al termine password intendendo indifferentemente una stringa di caratteri, di bit oppure un numero.

Password ed entropia. Come possiamo valutare la robustezza di una password? Un modo è quello di utilizzare il concetto di *entropia* (cf. Appendice A.3). Sia S una sorgente casuale di simboli appartenenti ad un alfabeto $\mathcal{S} = \{s_1, \ldots, s_n\}$ e supponiamo di aver definito una distribuzione di probabilità su $\mathcal{S}$. Ciò equivale a definire i valori $p_i = \mathbb{P}[S(t) = s_i]$ essendo $S(t)$ la variabile aleatoria corrispondente al carattere emesso all'istante t. L'entropia

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_12, © Springer-Verlag Italia 2012

media (misurata in bit/simbolo) emessa da S è allora:

$$H(S) = -\sum_{i=1}^{n} p_i \log p_i \text{ (bit/simbolo)}.$$

Quando tutti i simboli sono equiprobabili tale quantità è massima: vale a dire $H(S) \leq \log n$. Vogliamo modellare un'entità che sceglie una password attraverso la sorgente casuale $\mathcal{S}$.

Consideriamo, a titolo di esempio, password a 7-bit di caratteri ASCII (quelli presenti sulla tastiera di un calcolatore per intenderci). Si può verificare che ci sono 95 caratteri stampabili (avendo eliminato tutti i caratteri di controllo). Se le password sono considerate come sequenze di caratteri *casuali ed indipendenti* appartenenti a tale insieme, con $n = 95$ otteniamo $H(S) = \log 95 = 6.57$ (bit/carattere). Supponiamo di ritenere un attacco a forza bruta infattibile quando richiede più di 2^{128} passi. Ciò implica l'uso di password *completamente casuali* di lunghezza almeno $19.48 \approx 20$ caratteri![78]

La situazione in realtà è più complessa, in quanto tipicamente le password scelte dagli utenti *non sono affatto casuali*, ma sono parole appartenenti ad un certo dizionario. Dobbiamo allora considerare una sorgente con memoria. Sia $p(\mathbf{s})$ la distribuzione di probabilità *congiunta* relativa ad L caratteri consecutivi costituenti il vettore $\mathbf{s}$ scelto nell'insieme $\mathcal{S}^L$ delle parole ad L caratteri. Se definiamo la funzione entropia

$$H_L(S) = -\sum_{\mathbf{s}\in\mathcal{S}^L} p(\mathbf{s}) \log p(\mathbf{s}),$$

possiamo calcolare l'entropia media della sorgente come

$$H(S) = \lim_{L\to\infty} \frac{H_L(S)}{L}.$$

In generale ciò porta ad un abbassamento vertiginoso dell'entropia media. In particolare, nel caso della lingua italiana, si possono ottenere le seguenti stime:

$$\frac{H_3(S)}{3} \approx 3.15 \text{ (bit/carattere)} \quad \frac{H_4(S)}{4} \approx 2.67 \text{ (bit/carattere)}$$
$$\frac{H_5(S)}{5} \approx 2.22 \text{ (bit/carattere)} \quad \frac{H_6(S)}{6} \approx 1.87 \text{ (bit/carattere)}.$$

[78] Se consideriamo solo le lettere maiuscole, minuscole ed i numeri, sono disponibili solo 62 caratteri, il che abbassa l'entropia ad $H(S) = 5.95$ (bit/carattere), richiedendo così password ancora più lunghe.

Per questi motivi quindi, le soluzioni crittografiche adatte al caso in cui le chiavi hanno entropia elevata sono completamente inadeguate nel caso delle password.

Guida per il lettore. Nel Paragrafo 12.1 discuteremo alcune caratteristiche generali delle password; in particolare ci occuperemo della loro scelta e memorizzazione. Il Paragrafo 12.2 descrive uno schema di autenticazione molto elegante dovuto a Lamport [Lam81]. Nel Paragrafo 12.3 introdurremo due protocolli di scambio delle chiavi su base password: il protocollo EKE ed il protocollo SRP. Il Paragrafo 12.4, infine, introduce il modello formale per l'analisi di sicurezza dei protocolli di scambio delle chiavi su base password, esemplificandone l'uso attraverso uno schema proposto da Abdalla e Pointcheval [AP05].

12.1 Gestione delle password

In generale la scelta di password sicure è un problema spinoso [MT79; LR89; FK89] ed una questione sottovalutata dagli utenti. Esiste un modo per contrastare password deboli? La soluzione banale è quella di lasciare generare le password ad un computer e rinnovarle periodicamente. Lo svantaggio di questa soluzione è che le password così ottenute sono tipicamente difficili da memorizzare per un umano. Una buona soluzione a questo problema può essere quella di far scegliere la password all'utente e verificare poi che esse non siano troppo deboli, ad esempio controllando che non appartengano ad un dizionario di password deboli. Nel caso in cui il controllo non vada a buon fine si avvisa l'utente invitandolo a scegliere una password più sicura, magari suggerendo qualche linea guida su come generare una password sicura.

Filtri di Bloom. Un buon compromesso tra accuratezza e richiesta di tempo/memoria per il controllo su base dizionario di password proposte dagli utenti è costituito dai *filtri di Bloom* [Blo70]. Sia $\mathcal{D}$ un dizionario. Consideriamo N funzioni hash $H_j(\cdot)$ (per $j = 1, \ldots, N$) con valori in $0, \ldots, n-1$ ed una tabella $\mathcal{T}$ capace di memorizzare n valori. Per ogni parola $w \in \mathcal{D}$, si calcolano i valori di hash $v_j = H_j(w)$, con $j = 1, \ldots, N$ e si setta ad 1 la posizione della tabella corrispondente al valore calcolato, vale a dire $\mathcal{T}(v_j) = 1$. Quando l'utente propone una password π, questa è rifiutata se e solo se:

$$\mathcal{T}\left(H_j(\pi)\right) = 1 \qquad \forall\, j = 1, \ldots, N.$$

In altri termini, la password π è scartata se e solo se tutti gli elementi corrispondenti agli N valori hash $H_j(\pi)$ in $\mathcal{T}$ sono pari ad 1.

Osserviamo che due parole w *diverse* potrebbero portare ad 1 la stessa posizione in $\mathcal{T}$. Inoltre, molto più importante, potrebbe accadere che una password π sia scartata pur non appartenendo al dizionario $\mathcal{D}$ (si parla in questo caso di *falso positivo*). Calcoliamo la probabilità di tale evento, modellando le funzioni hash come un oracolo casuale. Sia ϵ la probabilità di un falso positivo. Calcoliamo

$$\begin{aligned}\mu &= \mathbb{P}\left[\mathcal{T}(i) = 0 \text{ per un dato } i\right] \\ &= \mathbb{P}\left[H_j(w) \neq i \text{ per } j = 1, \ldots, N \text{ e } \forall\ w \in \mathcal{D}\right] \\ &= \left(1 - \frac{1}{n}\right)^{N \cdot D},\end{aligned}$$

essendo $D = \#\mathcal{D}$. Possiamo concludere:

$$\begin{aligned}\epsilon &= \mathbb{P}\left[\mathcal{T}(H_j(\pi)) = 1 \text{ per } j = 1, \ldots, N\right] \\ &= (1-\mu)^N \approx \left(1 - e^{\frac{N \cdot D}{n}}\right)^N \leq \left(\frac{N \cdot D}{n}\right)^N,\end{aligned}$$

e tale valore può essere reso piccolo a piacere giocando su n, D ed N (cf. Esercizio 12.5).

Dalle password alle chiavi. Spesso le password sono usate per generare una chiave. Ovviamente la robustezza della chiave generata dipende fortemente dalla robustezza della password. Se la password è scelta in un dizionario $\mathcal{D}$ di dimensione $\#\mathcal{D}$, un attacco a forza bruta ha successo (mediamente) in $\#\mathcal{D}/2$ passi; la sicurezza non incrementa se si aumenta la lunghezza della chiave!

Dobbiamo distinguere il contesto simmetrico da quello asimmetrico. Nel primo caso, tipicamente, la chiave è generata a partire dalla password usando una funzione hash. Nel secondo caso, in genere, il processo di derivazione è più complesso, perché spesso nei crittosistemi asimmetrici le chiavi godono di proprietà particolari. Ad esempio è più difficile generare una chiave RSA (cf. Paragrafo 6.2) a partire da una password; una possibilità è quella di usare la password come seme iniziale di un PRG (cf. Paragrafo 3.2), a sua volta usato per generare i primi p e q necessari a costruire le chiavi RSA. Nella maggior parte dei contesti pratici questo processo non è eseguibile in tempo reale, perché molto costoso computazionalmente.

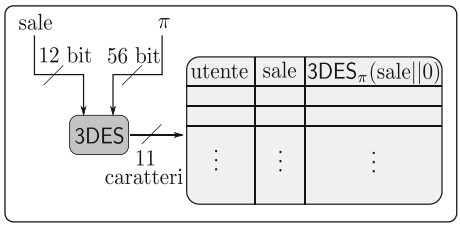


Fig. 12.1. Memorizzazione delle password in UNIX

"Sale" e "stiramento". Una metodologia generale per ricavare una chiave da una password è quella del "sale" e dello "stiramento" (*salt and stretching* in inglese): il "sale" è un qualche valore casuale usato nel processo di generazione della chiave, lo "stiramento" costituisce il calcolo necessario a derivare la chiave. Consideriamo il seguente esempio. Sia π la password ed s il "sale" (diciamo una stringa a 256 bit). Sia H una funzione hash crittograficamente sicura. Possiamo calcolare

$$X_i = H(X_{i-1} \,||\, \pi \,||\, s) \quad i = 1, 2, \ldots, \ell \quad \Rightarrow \quad k = X_\ell,$$

dove le stringhe X_i sono stringhe da 256 bit ed il valore X_0 è la stringa tutta nulla. Questa operazione deve essere eseguita ogni volta che la password è inserita. Ora l'attacco a forza bruta è più oneroso, in quanto l'attaccante che tenta di indovinare la password ha l'onere aggiuntivo di calcolare ripetutamente la funzione hash. Il valore di ℓ può essere scelto molto elevato, rendendo la vita difficile all'attaccante.

Memorizzazione delle password. La memorizzazione delle password è un problema non banale. Per ovvie ragioni esse non possono essere memorizzate in chiaro. D'altra parte memorizzarne solo l'hash non è una soluzione completamente soddisfacente, in quanto è sempre possibile tentare di indovinare la password una volta intercettato l'elenco contenente i valori hash relativi alle password memorizzate. Cifrare le password, d'altra parte, sposta solamente il problema, in quanto bisogna poi memorizzare in modo sicuro la chiave di cifratura.

Un esempio concreto di schema di memorizzazione delle password, è quello dei sistemi operativi UNIX [RT74]. Si crea un file contenente l'identità di ciascun utente con associato un valore di "sale" ed una stringa ottenuta dalla

password attraverso un cifrario a blocco. In concreto, per memorizzare password costituite da 7-8 caratteri ASCII (corrispondenti a 56 bit), si usano un "sale" a 12 bit ed il 3DES (cf. Paragrafo 5.3), come mostrato in Fig. 12.1. Siccome la password è lunga 56 bit, l'idea è quella di cifrare il "sale" usando la password π come chiave. Poiché il DES cifra blocchi a 64 bit, è necessario operare un "riempimento" (ad esempio aggiungendo tutti 0) dei 12 bit di "sale". Il risultato è una stringa di 64 bit (corrispondenti ad 11 caratteri ASCII). La verifica di una password avviene cifrando il "sale", usando la password dell'utente come chiave ed infine confrontando il valore ottenuto con il valore memorizzato.

Il ruolo del "sale" s è chiarito dal seguente esempio. Supponiamo che il "sale" non venga utilizzato e che la password π appartenga ad un certo dizionario $\mathcal{D}$. La probabilità che un attacco a forza bruta vada a buon fine è allora $p_0 = \frac{1}{D}$, essendo $D = \#\mathcal{D}$. L'attaccante potrebbe realizzare un attacco in parallelo semplicemente cifrando la stringa tutta nulla 0^{64}, e quindi tentare di indovinare la password cercando una collisione nel database intercettato. Se il file contiene N password e l'attacco in parallelo è lanciato ℓ volte, la probabilità di fallire è ovviamente $(1-\ell/D)^N$ e quindi l'attacco ha successo con probabilità $1-(1-\ell/D)^N$. Quando $N = 10^3$, $D = 10^6$ ed $\ell = 10^3$, tale valore è 0.63. Osserviamo che se invece utilizziamo il "sale" l'attacco in parallelo non è più realizzabile — o almeno il numero di utenti con stesso sale sarà molto basso — e l'attacco descritto ha successo solo con probabilità $\frac{\ell}{D} = 10^{-3}$ molto più bassa.

12.2 Lo schema di Lamport

Supponiamo che il Bianconiglio tenga traccia del numero ℓ di autenticazioni effettuate da Alice (da aggiornare ogni volta che Alice si autentica) e che utilizzi tale valore nel corso del protocollo di autenticazione. Lo schema di Lamport utilizza questo paradigma nel caso di autenticazione su base password. Sia x un valore ricavato da una password; poniamo $y_i = f^{\ell-i}(x)$ per ogni $i = 0, \ldots, \ell$, dove $f : \{0,1\}^n \to \{0,1\}^n$ è una permutazione unidirezionale (cf. Definizione 3.6). In altri termini il valore y_i è ottenuto applicando la funzione f al valore x esattamente $\ell - i$ volte. (Ciò è possibile proprio perché f è una permutazione.) Il Bianconiglio inizialmente memorizza il valore $y_0 = f^{\ell}(x)$; la funzione f è pubblica.

L'idea di base è la seguente. La prima volta che Alice vuole autenticarsi, il Bianconiglio chiede di esibire il valore y_1 che Alice può calcolare come $y_1 = f^{\ell-1}(x)$. Quindi il Bianconiglio controlla che $y_0 = f(y_1)$. Se la risposta di

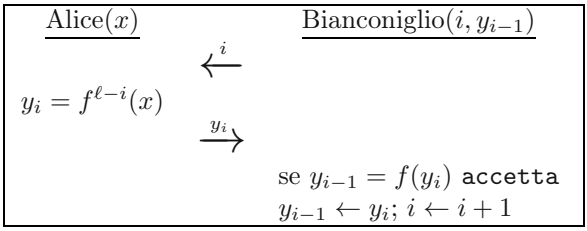


Fig. 12.2. L'interazione i-sima nel protocollo di Lamport

Alice è corretta — ovvero se $y_0 = f^\ell(x) = f(y_1) = f(f^{\ell-1}(x))$ — allora il Bianconiglio cancella y_0 e memorizza y_1. La seconda volta che Alice vuole autenticarsi, il Bianconiglio chiede di esibire y_2 e controlla che $y_1 = f(y_2)$ e così via. L'i-esima interazione è mostrata in Fig. 12.2.

Una volta raggiunta l'ℓ-sima autenticazione è necessario re-inizializzare il sistema (i.e., si sceglie un nuovo valore di x e si calcolano nuovamente i valori y_i). Per questo motivo si dice che il protocollo è pensato "per ℓ-usi". Intuitivamente la sicurezza (passiva) del protocollo di Lamport poggia sul fatto che (per ogni $i = 1, \ldots, \ell$) — quando il Bianconiglio memorizza y_{i-1} — Alice deve esibire una pre-immagine $y_i = f^{-1}(y_{i-1})$, il che è difficile senza conoscere x, poiché la funzione f è una permutazione unidirezionale. In effetti:

Teorema 12.1 (Sicurezza passiva del protocollo di Fig. 12.2). *Se la funzione f è una permutazione unidirezionale $(t_{\mathsf{OWP}}, \epsilon_{\mathsf{OWP}})$-sicura, il protocollo di Lamport è (t, ϵ)-sicuro (contro attaccanti passivi) per ℓ-usi, dove*

$$t \approx t_{\mathsf{OWP}} \qquad \epsilon \leq \epsilon_{\mathsf{OWP}}.$$

Dimostrazione. Supponiamo che il protocollo non sia (t, ϵ)-sicuro per ℓ-usi, vale a dire osservando $(i-1)$ autenticazioni oneste è possibile impersonare Alice al passo i con probabilità $\epsilon > \epsilon_{\mathsf{OWP}}$ (per un qualche $1 \leq i \leq \ell$). Ciò significa che, per un qualche i, abbiamo:

$$\mathbb{P}\left[\mathscr{A}(y_{i-1}, \ldots, y_1) = y_i : \; x \xleftarrow{\$} \{0,1\}^n, y_j = f^{\ell-j}(x) \; \forall \; 0 \leq j \leq \ell\right] > \epsilon_{\mathsf{OWP}}.$$

Mostriamo come usare $\mathscr{A}$ per costruire un attaccante PPT $\mathscr{B}$ in grado di invertire f con probabilità $> \epsilon_{\mathsf{OWP}}$, contro l'ipotesi che f sia $(t_{\mathsf{OWP}}, \epsilon_{\mathsf{OWP}})$-sicura. Dato un valore $\hat{y} = f(\hat{x})$, con $\hat{x} \xleftarrow{\$} \{0,1\}^n$, l'avversario $\mathscr{B}$ deve calcolare $\hat{x}$. Per fare ciò, $\mathscr{B}$ usa $\mathscr{A}$ come segue:

1. Calcola $f(\hat{y}), \ldots, f^{i-2}(\hat{y})$.

2. Lancia $\mathscr{A}(\hat{y}, f(\hat{y}), \ldots, f^{i-2}(\hat{y}))$. Sia z il valore restituito da $\mathscr{A}$.

3. Ritorna z.

Per prima cosa, siccome f è calcolabile in tempo polinomiale ed $\mathscr{A}$ è PPT, $\mathscr{B}$ è eseguibile in tempo $t_{\mathsf{OWP}} \approx t$. Inoltre, siccome i valori x ed $\hat{x}$ sono scelti a caso in $\{0,1\}^n$ (e poiché f è una permutazione), la distribuzione dei valori $\{y_{i-1}, \ldots, y_1\}$ nel protocollo è la stessa dei valori $\{\hat{y}, \ldots, f^{i-2}(\hat{y})\}$ simulati da $\mathscr{B}$. Ne segue che $z = f^{-1}(\hat{y}) = \hat{x}$ con probabilità almeno ϵ_{OWP}, contro l'ipotesi. □

Nella versione originale del protocollo di Lamport [Lam81], la funzione f è una funzione hash crittograficamente robusta. In particolare la stringa x è ottenuta concatenando la password π di Alice, un valore di "sale" s e l'identificativo del Bianconiglio. In simboli $y_i = H^{\ell-i}(\pi||s||\texttt{Bianconiglio})$.

Quando Alice vuole autenticarsi invia il suo identificativo `Alice`. Il Bianconiglio risponde inviando il valore del "sale" s, il suo identificativo `Bianconiglio` ed il valore i corrispondente al numero di autenticazioni già effettuate da Alice (aumentato di 1). Alice risponde con $y_i = H^{\ell-i}(\pi||s||\texttt{Bianconiglio})$ ed il Bianconiglio verifica che $H(y_i) = y_{i-1}$.[79]

Attacchi attivi. Siccome non c'è autenticazione mutua è difficile prevenire attacchi MiM e stabilire una chiave condivisa. Come anticipato, inoltre, la password deve essere re-inizializzata quando il valore i raggiunge ℓ. Esiste il seguente attacco (attivo) detto "del grande valore i". La Regina si sostituisce al Bianconiglio e, quando Alice si vuole autenticare, invia una sfida con un valore grande di i (diciamo $i = 100$ se il valore attualmente memorizzato dal Bianconiglio è $i = 20$). Il valore y_i ricevuto come risposta consentirà alla Regina di impersonare Alice per $100 - 20 = 80$ volte. Per evitare questo attacco Alice dovrebbe tenere memoria dell'ultima sfida ricevuta (ad esempio scrivendola su un foglietto di carta) in modo da non rispondere qualora la sfida sia diversa da quella attesa.

[79] Con un piccolo abuso di notazione, qui l'apice sul simbolo H indica il numero di volte che la funzione H è applicata al suo input, e non l'indice della funzione hash nell'insieme $\mathcal{H}$ (cf. Definizione 4.1).

12.3 Il protocollo EKE e le sue varianti

In un protocollo di scambio delle chiavi su base password, Alice ed il Bianconiglio usano una password per condividere una chiave di sessione k da utilizzare successivamente per altri scopi. Bellovin e Merritt [BM92] sono stati i primi a considerare lo scenario di scambio delle chiavi su base password, introducendo il protocollo dello scambio delle chiavi cifrato (*Encrypted Key Exchange*, EKE) che ha poi costituito la base per molti altri protocolli.

Il protocollo EKE è mostrato in Fig. 12.3. Alice ed il Bianconiglio condividono un segreto π a basso contenuto entropico ricavato da una password (ad esempio usando una funzione hash).

Sia p un primo e g un generatore di $\mathbb{Z}_p^*$. Alice sceglie un esponente casuale $x \xleftarrow{\$} \mathbb{Z}_p$ ed invia il suo identificativo `Alice` ed una cifratura con chiave π della quantità g^x. Il Bianconiglio recupera g^x (usando π), estrae $y \xleftarrow{\$} \mathbb{Z}_p$ e calcola la chiave $k = (g^x)^y = g^{xy}$. Sceglie quindi un Nonce N_B ed invia una cifratura con chiave π di g^y ed una cifratura con chiave k di N_B. In questo modo Alice può recuperare g^y (usando π) e calcolare la stessa chiave $k = (g^y)^x = g^{xy}$. Infine Alice sceglie un Nonce N_A ed invia una cifratura con chiave k della stringa $N_A||N_B$. Il Bianconiglio dimostra di possedere k decifrando il messaggio e legando la sua risposta a k ed N_A.

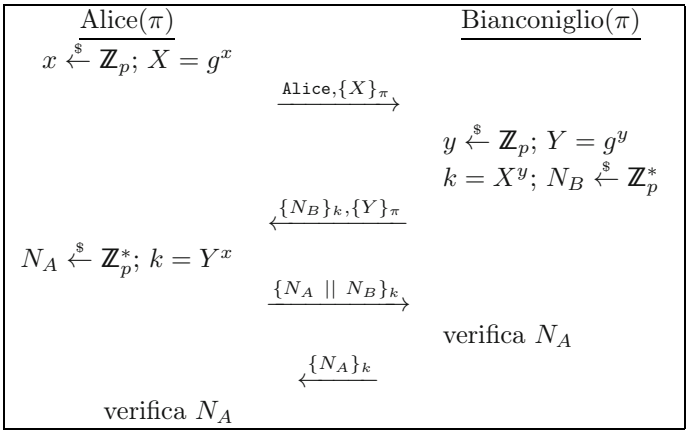


Fig. 12.3. Il protocollo EKE (tutte le operazioni sono intese modulo p)

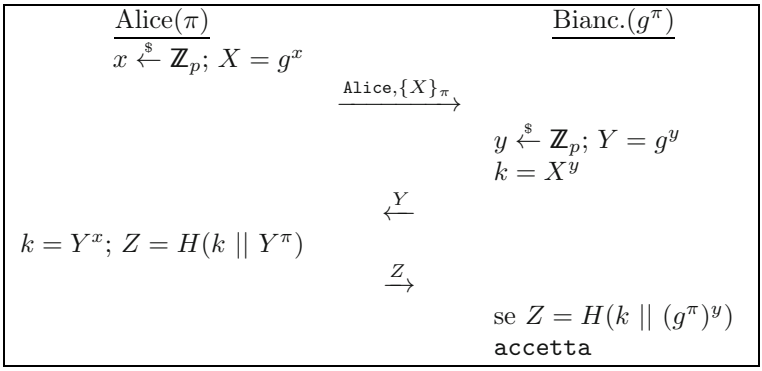


Fig. 12.4. Una miglioria del protocollo EKE

L'attacco su base dizionario è impraticabile, poiché tutte le quantità memorizzate sono casuali (non c'è modo di vedere se un tentativo di indovinare la password π è andato a buon fine o meno). Intuitivamente, violare la sicurezza del protocollo EKE equivale contemporaneamente ad indovinare la password e risolvere il problema CDH (cf. Definizione 6.7). Siccome $g^x \bmod p$ è compreso tra 0 e $p-1$, ogni volta che un tentativo di indovinare la password porta ad una quantità maggiore di p, la password può essere direttamente scartata. Se p è poco superiore ad una potenza di 2, praticamente metà delle password possono essere scartate ogni volta che si intercetta $\{g^x\}_\pi$. Una soluzione a questo problema è scegliere p poco inferiore ad una potenza di 2.

Osserviamo che se la Regina si impossessa dei dati memorizzati dal Bianconiglio, può impadronirsi di π e quindi impersonare successivamente Alice con successo. Esiste una versione successiva del protocollo [BM93] che evita questo problema. L'idea è di far sì che il Bianconiglio non memorizzi direttamente π, ma solamente qualcosa che gli permetta di verificare la password di Alice. Ciò è realizzato usando una funzione hash H, come mostrato in Fig. 12.4

SRP. Il protocollo sicuro remoto [Wu98] (*Secure Remote Protocol*, SRP) è ispirato al protocollo EKE.

Sia p un primo e g un generatore di $\mathbb{Z}_p^*$. Il Bianconiglio memorizza la quantità g^π, essendo π la stringa derivata dalla password. Alice inizia l'interazione inviando il suo identificativo `Alice` e la quantità g^x (con x scelto a caso). Il

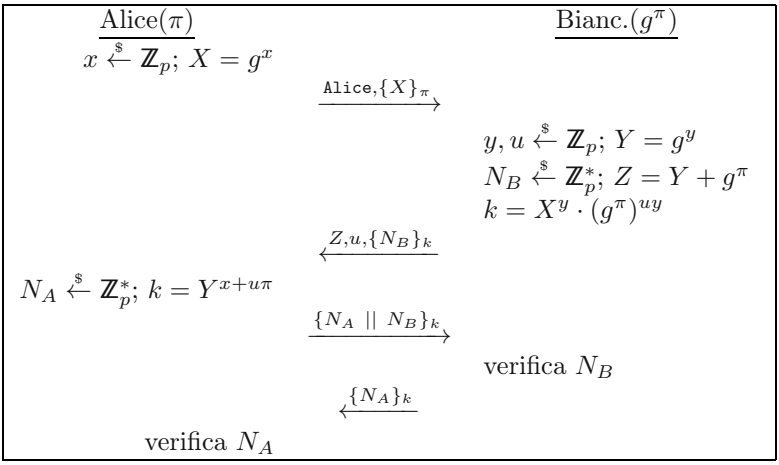


Fig. 12.5. Il Protocollo SRP

Bianconiglio sceglie y, u a caso e calcola la chiave $k = (g^x)^y \cdot (g^\pi)^{uy} = g^{y(x+u\pi)}$. Sceglie quindi un Nonce N_B ed invia la quantità $g^y + g^\pi$ seguita dalla stringa u e da una cifratura con chiave k del Nonce N_B. Alice può calcolare g^π, estrarre g^y a partire da $g^y + g^\pi$ e quindi ricavare $k = (g^y)^{x+u\pi} = g^{y(x+u\pi)}$. Infine Alice invia una cifratura con chiave k di un Nonce N_A ed il Bianconiglio lega la sua risposta a k ed N_A. L'interazione è mostrata in Fig. 12.5.

12.4 Lo schema di Abdalla e Pointcheval

Il progetto di protocolli di scambio delle chiavi basati sulle password, con una qualche forma di sicurezza dimostrabile, si è dimostrato complesso. Infatti, i primi protocolli avevano solo una prova euristica di sicurezza e molti sono stati attaccati con successo negli anni successivi [Pat97; MPS00]. I primi modelli formali (e relativi protocolli) sono stati proposti indipendentemente da Bellare, Pointcheval e Rogaway [BPR00] e da Boyko, MacKenzie, Patel e Swaminathan [MPS00; BMP00]. Questi schemi sono sicuri nel modello dell'oracolo casuale (cf. Paragrafo 4.4).

Il primo protocollo sicuro nel modello standard è dovuto a Goldreich e Lindell [GL06]. Sebbene la loro soluzione sia solo d'interesse teorico, in quanto

altamente inefficiente, ha avuto l'importanza di dimostrare che lo scambio delle chiavi su base password è realizzabile a partire da ipotesi molto deboli. Solo recentemente [KOY09; KV11] sono state trovate le prime soluzioni efficienti, sicure nel modello standard.

In questo paragrafo definiremo il modello formale per l'analisi dei protocolli di scambio delle chiavi su base password e lo applicheremo ad un semplice protocollo dovuto ad Abdalla e Pointcheval (sicuro nel modello dell'oracolo casuale), ispirato al protocollo EKE.

Nozioni di sicurezza per scambio delle chiavi su base password. Siccome le password sono usate principalmente nelle reti di calcolatori, nel contesto dei protocolli basati sulle password si fa riferimento, tipicamente, a clienti C (*client* in inglese) e serventi S (*server* in inglese). Ogni utente di un sistema di scambio delle chiavi su base password assume il ruolo di cliente o servente. Un cliente C possiede una password π_C; il servente memorizza le password relative a diversi clienti.

La sicurezza di un protocollo di scambio delle chiavi su base password è definita attraverso un insieme di oracoli (descritti di seguito) che astraggono le possibilità di un attaccante $\mathscr{A}$ in un attacco reale al protocollo. Notare che l'attaccante può lanciare diverse istanze di un dato utente U; indicheremo con U^i l'i-sima istanza dell'utente U. Si parla di modello *concorrente* quando $\mathscr{A}$ può attivare più istanze dell'utente U allo stesso tempo. Sia b un bit casuale, gli oracoli sono definiti di seguito:

$\mathsf{Esegui}(C^i, S^j)$. L'attaccante osserva passivamente un'interazione onesta tra il cliente C^i ed il servente S^j. L'oracolo restituisce una ricevuta che consiste in tutti i messaggi scambiati tra C^i ed S^j durante l'interazione.

$\mathsf{Invia}(U^i, m)$. L'oracolo restituisce la risposta dell'istanza U^i al messaggio m. Ciò modella un attaccante attivo che modifica i messaggi inviati sul canale dai partecipanti al protocollo.

$\mathsf{Rivela}(U^i)$. Restituisce la chiave di sessione dell'istanza U^i. Ciò modella il caso in cui l'attaccante riesce a compromettere il calcolatore controllato da U^i. Se nessuna chiave è definita per U^i oppure se l'oracolo Test (vedi sotto) è stato lanciato con input U^i, restituisci $\perp$.

$\mathsf{Test}(U^i)$. Se $b = 1$ ritorna la chiave di sessione di U^i, altrimenti restituisci una chiave casuale di uguale dimensione. Se nessuna chiave di sessione è definita per U^i, ritorna $\perp$.

Diremo che un'istanza U^i è stata aperta se è stato lanciato $\mathsf{Rivela}(U^i)$. Un istanza è detta accettare se ritorna `accetta` dopo aver ricevuto l'ultimo messaggio nel protocollo. Due istanze U_1^i ed U_2^j sono partner se (i) entrambe accettano, (ii) hanno uguale identificativo di sessione, (iii) l'identificativo del partner di U_1^i è U_2^j e viceversa e (iv) nessun altra istanza accetta con un partner che abbia identificativo U_1^i oppure U_2^j. (Possiamo pensare all'identificativo di sessione come ad una "fotografia" della conversazione tra cliente e servente prima che quest'ultimo accetti.) Infine diremo che U^i è un'istanza fresca se ha accettato e sia U^i che il suo partner non sono stati aperti.

L'avversario $\mathscr{A}$ ha accesso agli oracoli $\mathsf{Esegui}, \mathsf{Invia}, \mathsf{Rivela}, \mathsf{Test}$; quindi esegue la procedura Test su un'istanza fresca e ritorna un bit b'. Diremo che $\mathscr{A}$ viola la sicurezza se $b' = b$, ovvero se l'avversario è in grado di indovinare il bit b usato dall'oracolo Test. Il protocollo è (t, ϵ)-sicuro se $|\mathbb{P}[\mathscr{A} \text{ indovina } b] - 1/2| \leq \epsilon$, per ogni attaccante PPT $\mathscr{A}$ eseguibile in tempo t. Intuitivamente, questa definizione cattura il fatto che l'attaccante, nonostante interferisca con il protocollo attraverso gli oracoli Esegui, Invia e Rivela, non è in grado di distinguere la chiave di sessione, condivisa in un'istanza fresca, da una chiave casuale.

Lo schema di Abdalla e Pointcheval. Nel 2005, Abdalla e Pointcheval [AP05] hanno proposto due protocolli di scambio delle chiavi su base password e ne hanno dimostrato la sicurezza nel modello dell'oracolo casuale. Presentiamo qui il primo dei due protocolli, che è sicuro nel modello non-concorrente. Entrambi gli schemi seguono l'approccio del protocollo EKE, in cui la funzione di cifratura è implementata mascherando l'elemento X (risp. Y) attraverso il valore M^π (risp. N^π), dove M (risp. N) è un elemento a caso in $\mathbb{G}$ e π è la stringa derivata dalla password.

- `Parametri pubblici.` Sia $\mathbb{G}$ un gruppo finito con generatore g ed ordine un primo p; siano inoltre M, N elementi casuali in $\mathbb{G}$ ed H una funzione hash. I parametri $(\mathbb{G}, g, p, M, N, H)$ sono pubblici e noti a tutti.

- `Parametri segreti.` Il cliente ed il servente (indicati con utente A e utente B) condividono una password segreta $\pi \in \mathbb{Z}_p$.

- `Protocollo.` Il protocollo è illustrato in Fig. 12.6.

La correttezza del protocollo segue dal fatto che $k_A = k_B = g^{xy}$.

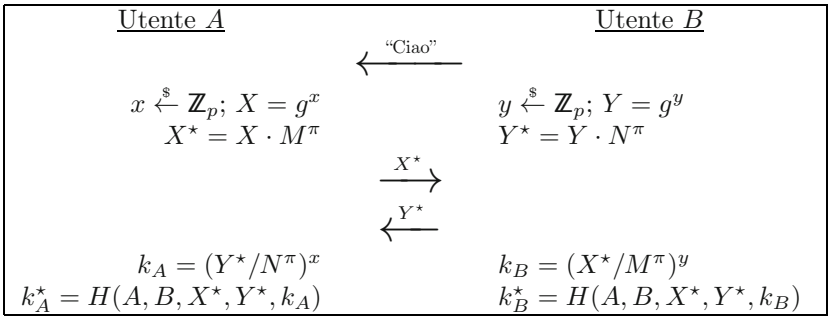


Fig. 12.6. Il protocollo di Abdalla e Pointcheval

Varianti del problema CDH. Sia $\mathbb{G}$ un gruppo finito con generatore g ed ordine un primo p. Nel problema CDH (cf. Definizione 6.7) si richiede, dati due elementi g^x, g^y, di calcolare $g^{xy} = \mathrm{CDH}(g^x, g^y)$. Consideriamo la seguente variante del problema CDH, detta problema CDH su base password a basi scelte (*Password-based Chosen-basis CDH*, PCCDH), più adatta al caso delle password. Sia $\mathcal{D} = \{1, \ldots, \ell\}$ un dizionario contenente ℓ password ed f una mappa iniettiva da $\mathcal{D}$ a $\mathbb{Z}_p$. Consideriamo un avversario $\mathscr{A}$ come segue. Nella prima fase $\mathscr{A}$ vede tre elementi casuali M, N, X' in $\mathbb{G}$ ed una descrizione della mappa f; al termine di questa fase deve restituire un valore $Y' \in \mathbb{G}$. Quindi, si sceglie una password $\mathrm{pwd} \in \{1, \ldots, \ell\}$ e la si inoltra ad $\mathscr{A}$. Dato $\pi = f(\mathrm{pwd})$, l'obiettivo dell'avversario è ritornare un valore k tale che $k = \mathrm{CDH}(X, Y)$ dove $X = X'/M^\pi$ ed $Y = Y'/N^\pi$. Notare che un avversario in grado di indovinare la password pwd nella prima fase, può calcolare $\pi = f(\mathrm{pwd})$ e quindi scegliere $Y' = g \cdot N^\pi$ e $k = X'/M^\pi$. Se le password sono a distribuzione uniforme in $\mathcal{D}$, la probabilità di successo di tale strategia è al più $1/\ell$. Diremo che un'istanza del problema PCCDH è (t, ϵ)-difficile se nessun attaccante PPT eseguibile in tempo t ha successo con probabilità migliore di $1/\ell + \epsilon$. (Essenzialmente assumere che il problema sia difficile equivale a dire che nessun avversario può far meglio che indovinare la password.) Più in generale, si può permettere all'avversario di restituire un insieme $\mathcal{K}$ contenente un certo numero di valori k (al termine della seconda fase): $\mathscr{A}$ ha successo se uno dei valori k soddisfa $k = \mathrm{CDH}(X, Y)$. In questo caso si parla di problema S-PCCDH (*Set Password-based Chosen-basis CDH*). Sorprendentemente, è possibile dimostrare che il problema S-PCCDH è equivalente al problema CDH classico [AP05].

Lemma 12.2 (S-PCCDH ⇒ CDH). *Sia $s = \#\mathcal{K}$. Se il problema CDH è $(t_{\mathsf{CDH}}, \epsilon_{\mathsf{CDH}})$-difficile, allora il problema S-PCCDH è (t, ϵ)-difficile, dove*

$$t_{\mathsf{CDH}} \approx t \qquad \epsilon \leq \sqrt[6]{\frac{2^{14}}{\ell^2}\left(\epsilon_{\mathsf{CDH}} + \frac{2s^4}{p}\right)}.$$

Quanto alla sicurezza del protocollo, abbiamo il seguente:

Teorema 12.3 (Sicurezza del protocollo in Fig. 12.6). *Sia $\mathbb{G}$ un gruppo finito con generatore g ed ordine un primo p. Indichiamo con $Q_{\mathsf{Invia}} = Q^A_{\mathsf{Invia}} + Q^B_{\mathsf{Invia}} + Q_{\mathsf{Ciao}}$ il numero di richieste effettuate all'oracolo Invia, dove Q^A_{Invia} (risp. Q^B_{Invia}) sono le richieste della forma $\mathsf{Invia}(A, \cdot)$ (risp. $\mathsf{Invia}(B, \cdot)$) e Q_{Ciao} sono le richieste di tipo $\mathsf{Invia}(A, \text{"Ciao"})$. Se il problema CDH è $(t_{\mathsf{CDH}}, \epsilon_{\mathsf{CDH}})$-difficile e se il problema S-PCCDH è (t, ϵ)-difficile, allora il protocollo di Abdalla e Pointcheval è $(t^\star, \epsilon^\star)$-sicuro nel modello dell'oracolo casuale, dove*

$$t^\star \approx t_{\mathsf{CDH}} \qquad \epsilon^\star \leq \frac{(Q_{\mathsf{Esegui}} + Q_{\mathsf{Invia}})^2}{2p} + Q_H \cdot \epsilon_{\mathsf{CDH}} + (Q^A_{\mathsf{Invia}} + Q^B_{\mathsf{Invia}}) \cdot \epsilon$$

essendo Q_H il numero di richieste effettuate all'oracolo H.

Dimostrazione. Osserviamo che, applicando il Lemma 12.2, la sicurezza segue direttamente dall'ipotesi CDH. Nella dimostrazione definiremo una serie di giochi intermedi, indistinguibili l'uno dall'altro (se non con probabilità trascurabile). Il primo gioco sarà del tutto equivalente ad un attacco reale al protocollo mentre nell'ultimo gioco la chiave $k^\star$ sarà calcolata *indipendentemente* dalla password, stabilendo così la sicurezza dello schema. Nel seguito, indicheremo con X_i la probabilità che l'attaccante $\mathscr{A}$ indovini il bit b usato nella procedura Test dell'i-simo gioco.

Gioco G_0. Questo esperimento corrisponde ad un attacco reale al protocollo, quindi per definizione abbiamo $\mathbb{P}[X_0] = \epsilon^\star$.

Gioco G_1. In questo esperimento scegliamo una password π a caso in $\mathbb{Z}_p$ e simuliamo gli oracoli $\mathsf{Esegui}, \mathsf{Rivela}, \mathsf{Test}$ e Invia come specificato di seguito.

In risposta ad una richiesta $\mathsf{Esegui}(A^i, B^j)$*:*

- $\theta \xleftarrow{\$} \mathbb{Z}_p$, $\Theta = g^\theta$, $\Theta^\star = \Theta \cdot M^\pi$;
- $\phi \xleftarrow{\$} \mathbb{Z}_p$, $\Phi = g^\phi$, $\Phi^\star = \Phi \cdot N^\pi$;

- $k = \Theta^{\phi}$, $k_A^\star = H(A, B, \Theta^\star, \Phi^\star, k)$, $k_B^\star = k_A^\star$;
- ritorna $((A, \Theta^\star), (B, \Phi^\star))$.

In risposta ad una richiesta $\mathsf{Rivela}(U^i)$*:*

- se la chiave di sessione $k^\star$ per U^i non è definita ritorna $\perp$;
- altrimenti restituisci $k^\star$.

In risposta ad una richiesta $\mathsf{Test}(U^i)$*:*

- $k^\star \leftarrow \mathsf{Rivela}(U^i)$ con $|k^\star| = n$; se $k^\star$ non è definito, ritorna $\perp$;
- $b \xleftarrow{\$} \{0,1\}$; se $b = 0$ poni $(k^\star)' = k^\star$, altrimenti $(k^\star)' \xleftarrow{\$} \{0,1\}^n$;
- ritorna $(k^\star)'$.

Simulazione di $\mathsf{Invia}(\cdot,\cdot)$. Procedi come segue:

- Per ogni richiesta di tipo $\mathsf{Invia}(A^i, \text{“Ciao”})$:
  - se un'istanza di A è già attiva ritorna $\perp$;
  - $\theta \xleftarrow{\$} \mathbb{Z}_p$, $\Theta = g^\theta$, $\Theta^\star = \Theta \cdot M^\pi$;
  - ritorna $(A, \Theta^\star)$.
- Per ogni richiesta di tipo $\mathsf{Invia}(B^i, (A, \Theta^\star))$:
  - $\phi \xleftarrow{\$} \mathbb{Z}_p$, $\Phi = g^\phi$, $\Phi^\star = \Phi \cdot N^\pi$;
  - $k = (\Theta^\star / M^\pi)^\phi$;
  - $k^\star = H(A, B, \Theta^\star, \Phi^\star, k)$;
  - ritorna $(B, \Phi^\star)$.
- Per ogni richiesta di tipo $\mathsf{Invia}(A^i, (B, \Phi^\star))$:
  - $k = (\Phi^\star / N^\pi)^\theta$;
  - $k^\star = H(A, B, \Theta^\star, \Phi^\star, k)$.

Ogni volta che si deve invocare l'oracolo casuale H con input x, si controlla se la coppia (x, ω) è presente nella lista $\mathcal{L}_H$ (inizialmente vuota). In caso negativo, si estrae $\omega \xleftarrow{\$} \{0,1\}^n$, si aggiunge (x, ω) alla lista $\mathcal{L}_H$ e si restituisce la coppia (x, ω). È immediato accorgersi che tale simulazione è perfetta, per cui $\mathbb{P}[X_1] = \mathbb{P}[X_0]$.

Gioco G_2. Questo gioco è identico al precedente, ma ora fermiamo la simulazione quando si trova una collisione nelle ricevute $((A, X^\star), (B, Y^\star))$. Siccome l'avversario invia Q_{Esegui} richieste all'oracolo Esegui e Q_{Invia} richieste all'oracolo Invia, il paradosso del compleanno (cf. Teorema 4.1) implica:

$$|\mathbb{P}[X_2] - \mathbb{P}[X_1]| \leq \frac{(Q_{\mathsf{Invia}} + Q_{\mathsf{Esegui}})^2}{2p}.$$

Gioco G_3. In questo esperimento rimpiazziamo l'oracolo H con un oracolo segreto H' e calcoliamo la chiave finale nella simulazione dell'oracolo Esegui come $k_A^\star, k_B^\star = H'(A, B, \Theta^\star, \Phi^\star)$. (Osserviamo in particolare che la funzione H' non dipende dalla password π.) La simulazione dell'oracolo H' è identica a quella dell'oracolo H, usando la relativa lista $\mathcal{L}_{H'}$.

Mostriamo che, se il problema CDH è difficile, è impossibile distinguere questo gioco dal gioco G_2. Notare che l'unico modo per distinguere i due esperimenti è interrogare direttamente l'oracolo in corrispondenza di valori $(A, B, \Theta^\star, \Phi^\star, k)$ tali che $H(A, B, \Theta^\star, \Phi^\star, k) \neq H'(A, B, \Theta^\star, \Phi^\star)$. Quando ciò avviene, si può risolvere un un'istanza (A, B) del problema CDH, simulando l'oracolo Esegui come nel gioco G_2, ma usando $\Theta = A \cdot g^\theta$ e $\Phi = B \cdot g^\phi$. In questo caso, infatti, $k = \mathrm{CDH}(\Theta, \Phi) = \mathrm{CDH}(A, B) \cdot A^\phi \cdot B^\theta \cdot g^{\theta\phi}$. Siccome $\mathscr{A}$ invia Q_H richieste all'oracolo e poiché il problema CDH è difficile:

$$|\mathbb{P}[X_3] - \mathbb{P}[X_2]| \leq Q_H \cdot \epsilon_{\mathsf{CDH}}.$$

Gioco G_4. Lo scopo di questo esperimento è quello di limitare il vantaggio di $\mathscr{A}$ durante un attacco attivo, in cui l'avversario ha modificato il flusso naturale del protocollo attraverso l'oracolo Invia. Pertanto nella simulazione dell'oracolo Invia sceglieremo l'output uniformemente a caso ed indipendentemente dalla password π, come segue:

Simulazione dell'oracolo Invia*:*

- per ogni richiesta di tipo $\mathsf{Invia}(A^i, \text{“Ciao”})$ rispondi con $(A, X^\star = g^{x^\star})$ per $x^\star \xleftarrow{\$} \mathbb{Z}_p$; se esiste un'altra istanza concorrente attiva, termina la sessione;
- per ogni richiesta di tipo $\mathsf{Invia}(B^i, (A, X^\star))$ rispondi con $(B, Y^\star = g^{y^\star})$ per $y^\star \xleftarrow{\$} \mathbb{Z}_p$; poni $k_B^\star = H'(A, B, X^\star, Y^\star)$;
- per ogni richiesta di tipo $\mathsf{Invia}(A^i, (B, Y^\star))$ calcola $k_A^\star = H'(A, B, X^\star, Y^\star)$.

Mostreremo che

$$|\mathbb{P}[X_4] - \mathbb{P}[X_3]| \leq (Q^A_{\mathsf{Invia}} + Q^B_{\mathsf{Invia}}) \cdot \epsilon,$$

concludendo così la dimostrazione. Introduciamo una sequenza di $Q^A_{\mathsf{Invia}} + Q^B_{\mathsf{Invia}} = Q_{AB}$ esperimenti ibridi $\mathcal{H}^j_3$, dove $j = 0, \ldots, Q_{AB}$. Sia i un contatore di tutte le richieste all'oracolo Invia del tipo $(B, (A, X^\star))$ oppure $(A, (B, Y^\star))$, ovvero tutte le richieste tranne quelle di tipo $(A, \text{“Ciao”})$. Nell'esperimento $\mathcal{H}^j_3$ procediamo come segue: (i) se $i \leq j$ elabora le richieste di tipo Invia come in G_4, altrimenti (ii) se $i > j$ elabora le richieste di tipo Invia come in G_3. Osserviamo che $G_4 \equiv \mathcal{H}^0_3$ e $G_3 \equiv \mathcal{H}^{Q_{AB}}_3$. Sia p_j la probabilità che $\mathscr{A}$ indovini il bit b nell'esperimento $\mathcal{H}^j_3$; abbiamo:

$$|\mathbb{P}[X_4] - \mathbb{P}[X_3]| \leq \sum_{j=1}^{Q_{AB}} |p_j - p_{j-1}|.$$

Pertanto resta da limitare $|p_j - p_{j-1}|$. Per fare ciò, consideriamo il seguente attaccante $\mathscr{B}$ che risolve il problema S-PCCDH. L'avversario $\mathscr{B}$ riceve come input tre elementi casuali in $\mathbb{G}$ — indicati nel seguito con $U = g^u$, $V = g^v$ e $W = g^w$ — e la descrizione della mappa f. Ricordiamo che $\mathscr{B}$ agisce in due fasi: al termine della prima fase deve scegliere un valore Y' in $\mathbb{G}$; nella seconda fase riceve pwd $\xleftarrow{\$} \{1, \ldots, \ell\}$ e deve ritornare $k = \mathrm{CDH}(W/U^\pi, Y'/V^\pi)$ per $\pi = f(\text{pwd})$. Quindi $\mathscr{B}$ simula l'ambiente per $\mathscr{A}$ come segue.

- Simula tutte le richieste agli oracoli Rivela, Esegui e Test come in G_3.
- In risposta ad una richiesta $\mathsf{Invia}(A, \text{“Ciao”})$:
  - se $i \leq j$ rispondi con $(A, X^\star = W \cdot g^{x^\star})$, per $x^\star \xleftarrow{\$} \mathbb{Z}_p$ (se esiste un'altra sessione concorrente per A, termina la sessione);
  - se $i > j$ elabora la richiesta come in G_3.
- In risposta ad una richiesta $\mathsf{Invia}(B, (A, X^\star))$:
  - se $i < j$ elabora la richiesta come in G_4;
  - se $i = j$ rispondi con $(B, Y^\star = W)$ e ritorna $Y' = Y^\star$ come output della prima fase; ricevi l'input pwd $\xleftarrow{\$} \{1, \ldots, \ell\}$ della seconda fase e, posto $\pi = f(\text{pwd})$, calcola $k^\star_B = H(A, B, X^\star, Y^\star, k_B)$ con $k_B = (X^\star/V^\pi)^{w - u \cdot \pi}$ (*caso (I)*);

  - se $i > j$ elabora la richiesta come in G_3.

- In risposta ad una richiesta $\mathsf{Invia}(A, (B, Y^\star))$:
  - se $i < j$ elabora la richiesta come in G_4;
  - se $i = j$, ritorna $Y' = Y^\star$ come output della prima fase; ricevi l'input pwd $\stackrel{\$}{\leftarrow} \{1, \ldots, \ell\}$ della seconda fase e, posto $\pi = f(\text{pwd})$, calcola $k_A^\star = H(A, B, X^\star, Y^\star, k_A)$, con $k_A = (Y^\star / V^\pi)^{w + x^\star - u \cdot \pi}$ (*caso (II)*);
  - se $i > j$ elabora la richiesta come in G_3.

Indichiamo con k la parte dell'input di H che non è presente in H' e siano $k_1, \ldots, k_{Q_H}$ la lista di tutti questi valori k (uno per ogni richiesta all'oracolo casuale). Nel caso (I) $\mathscr{B}$ pone $k'_i = k_i / (Y'/V^\pi)^{x^\star}$ per ogni $i = 1, \ldots, Q_H$. Nel caso (II) $\mathscr{B}$ pone $k'_i = k_i$. Quindi restituisce $\mathcal{K} = \{k'_1, \ldots, k'_{Q_H}\}$.

Notare che, se i valori u, v e w fossero noti, la simulazione di una richiesta di tipo Invia sarebbe identica ad un'esecuzione dell'esperimento $\mathcal{H}_3^{j-1}$. D'altra parte $\mathscr{B}$ non conosce u, v e w, quindi, invece di usare l'oracolo H come descritto sopra, usa l'oracolo segreto H', calcolando $k_A^\star, k_B^\star = H'(A, B, X^\star, Y^\star)$. In questo caso la simulazione di una richiesta di tipo Invia è identica all'ibrido $\mathcal{H}_3^j$. Sia $\mathcal{E}$ l'evento in cui l'avversario richiede il valore $H(A, B, X^\star, Y^\star, k)$ per $k = \mathrm{CDH}(X^\star / U^\pi, Y^\star / V^\pi)$ ed uno tra $X^\star$ oppure $Y^\star$ è l'elemento definiti nell'elaborazione della richiesta j-sima. Osserviamo che gli esperimenti $\mathcal{H}_3^j$ e $\mathcal{H}_3^{j-1}$ sono identici a patto che $\mathcal{E}$ *non* si verifichi. Ciò implica $|p_j - p_{j-1}| \leq \mathbb{P}[\mathcal{E}]$. D'altra parte quando $\mathcal{E}$ si verifica, la lista $\mathcal{K}$ restituita da $\mathscr{B}$ contiene il valore $k = \mathrm{CDH}(X^\star / U^\pi, Y^\star / V^\pi)$. In questo caso $\mathscr{B}$ può risolvere il problema S-PCCDH in quanto k_A (nel caso (I)) oppure k_B (nel caso (II)) possono essere usati per calcolare $\mathrm{CDH}(W/U^\pi, Y'/V^\pi)$ in accordo a

$$k_A = \mathrm{CDH}(W \cdot g^{x^\star} / U^\pi, Y^\star / V^\pi) = \mathrm{CDH}(W/U^\pi, Y'/V^\pi) \cdot \mathrm{CDH}(Y'/V^\pi)^{x^\star}$$
$$k_B = \mathrm{CDH}(X^\star / V^\pi, W/U^\pi) = \mathrm{CDH}(W/U^\pi, Y'/V^\pi).$$

Siccome il problema S-PCCDH è (t, ϵ)-difficile per ipotesi, segue $\mathbb{P}[\mathcal{E}] \leq Q_{AB} \cdot \epsilon$, come desiderato. □

Esercizi

Esercizio 12.1. Un sistema informatico consente ad Alice di scegliere una password con lunghezza da 1 ad 8 caratteri. Ipotizzando che un attaccante possa provare 10^4 password al secondo, l'amministratore del sistema decide di far scadere le password degli utenti non appena queste possano essere indovinate con probabilità 0.1. Determinare il tempo di validità della password scelta da Alice, nei seguenti casi:

1. I caratteri della password sono scelti tra tutti i caratteri ASCII compresi tra 1 a 127.

2. I caratteri della password sono solamente alfa-numerici (quindi tutte le lettere dalla "A" alla "Z" e tutti i numeri dallo "0" al "9").

3. I caratteri della password sono numeri.

Esercizio 12.2. Supponiamo che alcune password siano selezionate prelevando combinazioni a 4 caratteri delle 21 lettere dell'alfabeto italiano. Consideriamo un avversario in grado di provare una password al secondo.

1. Assumendo che l'avversario riceva una notifica alla fine di ciascun tentativo, quanto tempo impiega a trovare la password?

2. Assumendo che l'avversario riceva una notifica all'inserimento di ciascun carattere, quanto tempo impiega a trovare la password?

Esercizio 12.3. Un generatore di password seleziona 2 campioni di 3 lettere scelte a caso in un alfabeto $\mathcal{A}$ (diciamo l'alfabeto italiano), per formare password di 6 caratteri. Ciascun campione è della forma *cvc* (ovvero consonante, vocale, consonante), in cui $v \in \mathcal{V} = \{\text{a,e,i,o,u}\}$ e $c \in \mathcal{C} = \mathcal{A} \setminus \mathcal{V}$.

1. Determinare il numero totale di password possibili.

2. Qual è la probabilità di indovinare una password?

Esercizio 12.4. Dato un dizionario $\mathcal{D}$, si utilizza un filtro di Bloom con $N = 2$ funzioni hash ed un tabella che memorizza fino ad $n = 5$ valori. Supponiamo di definire

$$H_1(x) = x \bmod 5 \qquad\qquad H_2(x) = 2 \cdot x + 3 \bmod 5.$$

(Notare che ciò prevede che le parole $w \in \mathcal{D}$ siano prima rappresentate come valori interi.) Ipotizzando che il dizionario contenga solo le parole $\{9, 11\}$:

1. Completare la tabella $\mathcal{T}$ relativa al filtro di Bloom descritto.

2. Usare la tabella per verificare se le parole $\{15, 16\}$ appartengono al dizionario. Si genera un falso positivo?

Esercizio 12.5. La tecnica di filtraggio di Bloom è applicata ad un dizionario $\mathcal{D}$ contenente $\#\mathcal{D} = D$ parole, utilizzando N funzioni hash ed una tabella a dimensione n.

1. Supponiamo che sia noto il valore del rapporto n/D. Determinare il valore ottimo di N che minimizza la probabilità di un falso positivo, ovvero $\epsilon = (1 - e^{-N \cdot D/n})^N$. *(Suggerimento: osservare che basta minimizzare* $\ln \epsilon$*.)*

2. Esprimere la probabilità di falso positivo in funzione del valore ottimo trovato al punto precedente. A quanto si deve fissare il rapporto n/D affinché tale probabilità sia inferiore al 2%? E se volessimo probabilità di falso positivo inferiore all'1%?

Esercizio 12.6. Consideriamo il seguente protocollo, che utilizza i filtri di Bloom per stimare la differenza di insiemi. Alice possiede un insieme $\mathcal{X}$ ed il Bianconiglio un insieme $\mathcal{Y}$, entrambi con $\#\mathcal{X} = \#\mathcal{Y} = D$ elementi. Entrambi creano un filtro di Bloom relativamente ai loro insiemi, usando lo stesso numero n di bit e lo stesso numero N di funzioni hash. Determinare il numero atteso di bit in cui i filtri di Bloom differiscono, in funzione di D, N, n e del valore $\#(\mathcal{X} \cap \mathcal{Y})$.
(Suggerimento: provare prima i casi $\#(\mathcal{X} \cap \mathcal{Y}) = 0, 1, 2, \ldots$ *e quindi derivare una formula generale.)*

Esercizio 12.7. Consideriamo la seguente variante del filtraggio di Bloom. Anziché condividere lo spazio n tra N funzioni hash, ciascuna funzione hash ha assegnato uno slot contenente n/N valori.

1. Analizzare le prestazioni del filtro modificato per un generico dizionario $\mathcal{D}$ contenente $\#\mathcal{D} = D$ elementi. In particolare, mostrare che in questo caso si ottiene:
$$\epsilon = \left(1 - \frac{N}{n}\right)^D \approx e^{-N \cdot D/n}.$$

2. Paragonare il risultato ottenuto con un filtro di Bloom standard. Quale soluzione genera un numero maggiore di falsi positivi?

Esercizio 12.8. Istanziare il protocollo di Lamport con una funzione concreta, assumendo che il problema del logaritmo discreto sia difficile in $\mathbb{Z}_p^*$.

Esercizio 12.9. Consideriamo password composte da 10 dei 95 caratteri dei caratteri ASCII stampabili. Supponiamo di disporre di un sistema in grado di cifrare 6.4 milioni di password al secondo. Quanto tempo è necessario per provare tutte le possibili password su un sistema UNIX, avendo accesso al file contenente le password in forma cifrata?

Letture consigliate

[AP05] Michel Abdalla e David Pointcheval. “Simple Password-Based Encrypted Key Exchange Protocols”. In: *CT-RSA*. 2005, pp. 191–208.

[Blo70] Burton H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors”. In: *Commun. ACM* 13.7 (1970), pp. 422–426.

[BM92] Steven M. Bellovin e Michael Merritt. “Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks”. In: *IEEE Symposium on Security and Privacy*. 1992, pp. 72–84.

[BM93] Steven M. Bellovin e Michael Merritt. “Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise”. In: *ACM Conference on Computer and Communications Security*. 1993, pp. 244–250.

[BMP00] Victor Boyko, Philip D. MacKenzie e Sarvar Patel. “Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman”. In: *EUROCRYPT*. 2000, pp. 156–171.

[BPR00] Mihir Bellare, David Pointcheval e Phillip Rogaway. “Authenticated Key Exchange Secure against Dictionary Attacks”. In: *EUROCRYPT*. 2000, pp. 139–155.

[Car71] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1871.

[FK89] David C. Feldmeier e Philip R. Karn. “UNIX Password Security - Ten Years Later”. In: *CRYPTO*. 1989, pp. 44–63.

[GL06] Oded Goldreich e Yehuda Lindell. “Session-Key Generation Using Human Passwords Only”. In: *J. Cryptology* 19.3 (2006), pp. 241–340.

[KOY09] Jonathan Katz, Rafail Ostrovsky e Moti Yung. “Efficient and secure authenticated key exchange using weak passwords”. In: *J. ACM* 57.1 (2009).

[KV11] Jonathan Katz e Vinod Vaikuntanathan. “Round-Optimal Password-Based Authenticated Key Exchange”. In: *TCC*. 2011, pp. 293–310.

[Lam81] Leslie Lamport. “Password Authentification with Insecure Communication”. In: *Commun. ACM* 24.11 (1981), pp. 770–772.

[LR89] Michael Luby e Charles Rackoff. “A Study of Password Security”. In: *J. Cryptology* 1.3 (1989), pp. 151–158.

[MPS00] Philip D. MacKenzie, Sarvar Patel e Ram Swaminathan. “Password-Authenticated Key Exchange Based on RSA”. In: *ASIACRYPT*. 2000, pp. 599–613.

[MT79] Robert Morris e Ken Thompson. “Password Security - A Case History”. In: *Commun. ACM* 22.11 (1979), pp. 594–597.

[Pat97] Sarvar Patel. “Number Theoretic Attacks on Secure Password Schemes”. In: *IEEE Symposium on Security and Privacy*. 1997, pp. 236–247.

[RT74] Dennis Ritchie e Ken Thompson. “The UNIX Time-Sharing System”. In: *Commun. ACM* 17.7 (1974), pp. 365–375.

[Wu98] Thomas D. Wu. “The Secure Remote Password Protocol”. In: *NDSS*. 1998.

CAPITOLO 13

Conoscenza nulla

«E adesso vediamo se tu credi a me. Io, per la precisione, ho cent'un anni, cinque mesi e un giorno». «Non posso crederci!» disse Alice. «Non puoi?» disse la Regina in tono compassionevole. «Prova di nuovo: fai un bel respiro e chiudi gli occhi». Alice rise. «Non serve provare» disse; «alle cose impossibili non si può credere!».

Lewis Carroll, Attraverso lo Specchio e quel che Alice vi trovò [Car71]

Nei capitoli precedenti ci siamo principalmente occupati di quali requisiti deve soddisfare un protocollo di autenticazione sicuro e di come realizzare tali protocolli. In questo capitolo, ci occuperemo di uno scenario più generale. Immaginiamo che Alice voglia convincere lo Stregatto della veridicità di una certa affermazione.

> *È possibile per Alice convincere lo Stregatto della veridicità dell'affermazione, senza rivelare null'altro se non il fatto che l'affermazione è appunto veritiera?*

Ciò è in effetti possibile, come vedremo, per ogni "linguaggio" in **NP** e la risposta a questa domanda risiede nei protocolli a conoscenza nulla (*Zero Knowledge*, ZK), oggetto del presente capitolo.

Per avere un'intuizione più chiara del concetto di conoscenza nulla, consideriamo il seguente esempio. Alice dichiara di poter contare le foglie di un grande acero in pochissimo tempo, e vuole convincere lo Stregatto della sua abilità senza però rivelargli il suo metodo. Mentre Alice chiude gli occhi, lo Stregatto sceglie una delle seguenti azioni: (i) stacca una foglia dall'acero oppure (ii) non stacca alcuna foglia. A questo punto Alice può aprire gli occhi e deve capire (in una frazione di secondo) cosa ha fatto lo Stregatto. Ora, se Alice sbaglia a dare la risposta è chiaro che non è affatto in grado di contare le foglie dell'acero. Ma cosa si può dedurre se la risposta data da Alice è corretta? Ovviamente, Alice potrebbe semplicemente essere stata fortunata ed

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_13,

aver indovinato la risposta (il che accade con probabilità 1/2); ma cosa dire se l'interazione tra Alice e lo Stregatto venisse ripetuta diciamo 100 volte? Ora la probabilità che Alice sia fortunata è solo 2^{-100}, così che lo Stregatto deve convincersi che Alice conosce effettivamente un metodo rapidissimo per contare le foglie dell'acero! (Qualora il lettore avesse voglia di arricchire la conoscenza dei propri figli spiegando loro il concetto di conoscenza nulla sin dalla tenera età, si consiglia di consultare [Qui+89].)

Definizioni su base simulazione. Per formalizzare la proprietà di conoscenza nulla, useremo una definizione su base "simulazione". Richiederemo che tutto ciò che la Regina Rossa è in grado di imparare osservando un'interazione tra Alice e lo Stregatto, possa essere perfettamente "simulato" dalla Regina Bianca (detta anche il "simulatore") *senza eseguire alcun protocollo*. Ciò intuitivamente significa che l'interazione tra Alice e lo Stregatto non ha rilasciato alcuna informazione se non la veridicità o la falsità dell'affermazione stessa.

Guida per il lettore. La struttura del capitolo è la seguente. Nel Paragrafo 13.1, daremo la definizione di sistema di prova interattivo a conoscenza nulla e studieremo le sue proprietà. Vedremo quindi alcuni risultati negativi che si applicano ai sistemi di prova interattivi nel Paragrafo 13.2. Studieremo quindi (nel Paragrafo 13.3) una classe molto importante di protocolli interattivi (detti protocolli-Σ) e vedremo diversi esempi concreti di tali schemi per alcuni linguaggi legati alla teoria dei numeri. Infine, nel Paragrafo 13.4, definiremo i protocolli a conoscenza nulla non-interattivi e vedremo come essi possano essere utilizzati per costruire cifrari a chiave pubblica CCA-sicuri nel modello standard. Uno studio completo dei protocolli a conoscenza nulla esula dagli scopi del testo. Per approfondimenti si rimanda alla letteratura oppure a testi più specialistici, ad esempio [Gol01, Capitolo 4]. Una panoramica si trova in [Gol02b].

13.1 Sistemi di prova a conoscenza nulla

In un sistema di prova interattivo [GMR85], un "dimostratore" $\mathscr{P}$ (i.e., Alice) tenta di convincere un verificatore $\mathscr{V}$ (i.e., lo Stregatto) della veridicità di una certa affermazione. Formalmente, la terminologia "veridicità di una certa affermazione" è espressa attraverso il concetto di *relazione*.

Definizione 13.1 (Relazione a tempo polinomiale). Una relazione a tempo polinomiale è una funzione $\varrho : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, tale che: (i)

$\varrho(x,w) = 1$ implica $|w| \leq \text{poly}(|x|)$ e (ii) data una coppia (x,w) possiamo verificare in tempo polinomiale (in $|x|$ e $|w|$) se $\varrho(x,w) = 1$. ■

A partire da ogni relazione ϱ, è possibile definire un linguaggio

$$\mathcal{L} = \{x : \exists\, w \text{ tale che } \varrho(x,w) = 1\}\,.$$

Notare che se ϱ è una relazione a tempo polinomiale, allora $\mathcal{L} \in \mathbf{NP}$ (in quanto w è in questo caso un indizio per x). In altri termini, ad ogni linguaggio in **NP** è associata una relazione

$$\varrho(x,w) = 1 \quad \Leftrightarrow \quad x \in \mathcal{L} \text{ e } w \text{ è un indizio per } x.$$

Informalmente, un sistema di prova interattivo è uno scambio di messaggi tra $\mathscr{P}$ e $\mathscr{V}$ in cui, fissato un linguaggio $\mathcal{L}$, il dimostratore tenta di convincere il verificatore che un dato elemento x appartiene ad $\mathcal{L}$. Indicheremo tale interazione con $\mathscr{P}(\cdot) \leftrightarrows \mathscr{V}(\cdot)$ e diremo che l'output è `accetta` quando $\mathscr{V}$ è convinto che $x \in \mathcal{L}$ dopo aver interagito con $\mathscr{P}$. A livello intuitivo vorremmo che un sistema di prova interattivo sia (i) *completo*, nel senso che quando $x \in \mathcal{L}$ il dimostratore $\mathscr{P}$ riesca sempre a convincere $\mathscr{V}$ e (ii) *valido*, nel senso che quando $x \notin \mathcal{L}$ non deve essere possibile (in nessun modo!) convincere $\mathscr{V}$ del contrario. Ciò motiva la seguente definizione:

Definizione 13.2 (Sistema di prova interattivo). Sia $n \in \mathbb{N}$ un parametro statistico di sicurezza. Una coppia di algoritmi PPT $(\mathscr{P}, \mathscr{V})$ è detta un sistema di prova interattivo per un linguaggio $\mathcal{L}$, se soddisfa quanto segue:

- *Completezza.* Per ogni $x \in \mathcal{L}$,

$$\mathbb{P}\left[\mathscr{P}(1^n, x) \leftrightarrows \mathscr{V}(1^n, x) = \texttt{accetta}\right] \geq 1 - \text{negl}(n).$$

- *Validità.* Per ogni $x \notin \mathcal{L}$ e per ogni algoritmo $\mathscr{P}^*$ (anche computazionalmente illimitato):

$$\mathbb{P}\left[\mathscr{P}^*(1^n, x) \leftrightarrows \mathscr{V}(1^n, x) = \texttt{accetta}\right] \leq \text{negl}(n).$$

(Quando si richiede che $\mathscr{P}^*$ sia PPT si parla di *argomenti* [BCC88].)

■

Notare che ogni linguaggio $\mathcal{L} \in \mathbf{NP}$, ammette un sistema di prova interattivo: dato $x \in \mathcal{L}$ il dimostratore invia un indizio w relativo ad x ed il verificatore ritorna `accetta` se e solo se $\varrho(x,w) = 1$.

Conoscenza nulla. Definiremo ora la proprietà di conoscenza nulla, introdotta per la prima volta da Goldwasser, Micali e Rackoff [GMR85]. Intuitivamente, diremo che un sistema di prova interattivo (per un linguaggio $\mathcal{L}$) è ZK se tutto ciò che può essere ricavato dopo aver interagito con $\mathscr{P}$ per un input x, può essere calcolato, a partire da x, senza interagire con $\mathscr{P}$. È importante sottolineare che ciò deve valere per ogni modo efficiente di interagire con $\mathscr{P}$, non necessariamente quello prescritto da un'esecuzione onesta del protocollo. Formalmente:

Definizione 13.3 (Conoscenza nulla computazionale). Sia $(\mathscr{P}, \mathscr{V})$ un sistema di prova interattivo per un linguaggio $\mathcal{L}$. Diremo che $(\mathscr{P}, \mathscr{V})$ è a conoscenza nulla se, per ogni algoritmo PPT $\mathscr{V}^*$, esiste un simulatore PPT $\mathscr{Z}$ (detto simulatore di ZK) tale che i seguenti insiemi di distribuzioni sono computazionalmente indistinguibili:

$$\{\mathscr{P}(1^n, x) \leftrightarrows \mathscr{V}^*(1^n, x)\}_{x \in \mathcal{L}} \overset{c}{\approx} \{\mathscr{Z}(1^n, x)\}_{x \in \mathcal{L}} .$$

■

Una variante della definizione precedente richiede che le due distribuzioni siano identiche anziché computazionalmente indistinguibili; si parla in questo caso di conoscenza nulla *perfetta*. Quando le due distribuzioni sono vicine in distanza statistica (cf. Definizione A.3), infine, si parla di conoscenza nulla *statistica*.

Diremo che un sistema di prova è banale se $\mathcal{L} \in \mathbf{BPP}$ (oppure in $\mathcal{L} \in \mathbf{P}$), poiché in questo caso il verificatore può decidere da solo se $x \in \mathcal{L}$ (quindi è banale esibire un simulatore). Il nostro obiettivo sarà costruire sistemi di prova a conoscenza nulla per affermazioni che il verificatore non può verificare da solo.

Isomorfismo di grafi. Presentiamo un sistema di prova a conoscenza nulla per il seguente problema di isomorfismo di grafi. Un grafo $G = (V, E)$ è specificato dall'insieme dei vertici V e dall'insieme E di lati tra elementi di V. Due grafi $G_0 = (V_0, E_0)$ e $G_1 = (V_1, E_1)$ sono isomorfi — indicato con $G_0 \simeq G_1$ — se esiste una permutazione dei vertici $\varphi : V_0 \to V_1$ tale che $(u, v) \in E_0 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_1$. La mappa φ è detta isomorfismo da G_0 a G_1. Con un piccolo abuso di notazione indicheremo con $\varphi(G)$ il grafo ottenuto permutando i vertici di G secondo φ. Osserviamo che il linguaggio $\mathcal{L} = \{(G_0, G_1) : G_0 \simeq G_1\}$ è ovviamente in **NP**. (L'indizio è rappresentato da una descrizione della mappa φ.)

Consideriamo il protocollo in Fig. 13.1 per provare che G_0 e G_1 sono isomorfi secondo la mappa φ. (Abbiamo indicato con $(\varphi_1 \circ \varphi_2)(G)$ la permutazione

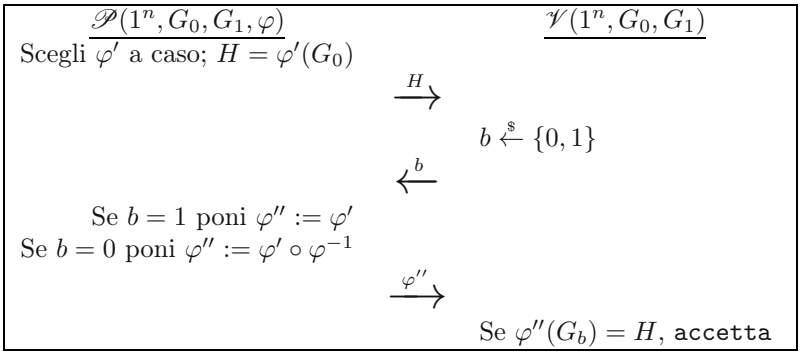


Fig. 13.1. Sistema di prova interattivo per il linguaggio dei grafi isomorfi

composta delle permutazioni φ_1 e φ_2 con input G, ovvero la permutazione ottenuta applicando φ_1 al grafo permutato $\varphi_2(G)$.) Abbiamo il seguente:

Teorema 13.1 (Conoscenza nulla per il linguaggio dei grafi isomorfi). *Il protocollo in Fig. 13.1 è un sistema di prova ZK (con errore di validità* $1/2$*) per il linguaggio* $\mathcal{L}$ *di isomorfismo di grafi.*

Dimostrazione. Ovviamente $\mathscr{P}$ e $\mathscr{V}$ sono eseguibili in tempo polinomiale. La proprietà di completezza è immediata da verificare. Infatti se $b = 0$ e $\mathscr{P}$ è onesto abbiamo $H = \varphi'(G_0)$, quindi $\mathscr{V}$ accetta con probabilità 1. Lo stesso vale per $b = 1$, perché $\varphi''(G_1) = \varphi' \circ \varphi^{-1}(G_1) = \varphi'(G_0) = H$.

Mostriamo ora che la proprietà di validità è verificata con errore $1/2$. Si tratta di mostrare che quando G_0 e G_1 non sono isomorfi, non è possibile far accettare $\mathscr{V}$ con probabilità migliore di $1/2$. L'osservazione chiave qui è che uno solo tra G_0 e G_1 può essere isomorfo ad H; se infatti entrambi i grafi lo fossero si dovrebbe avere $G_0 \simeq G_1$ per transitività. Ciò implica che $\mathscr{P}$ può ingannare $\mathscr{V}$ solo con probabilità $1/2$. Tale valore non è trascurabile (ma può essere reso tale, come discuteremo al termine della dimostrazione).

Resta da mostrare che il protocollo è a conoscenza nulla. Per ogni $\mathscr{V}^*$ scorretto, costruiamo il seguente simulatore $\mathscr{Z}(1^n, (G_0, G_1))$:

1. Fissa la randomicità ω per $\mathscr{V}^*$.

2. Ripeti quanto segue κ volte:

(a) scegli una permutazione casuale φ', estrai $b \xleftarrow{\$} \{0,1\}$ e poni $H = \varphi'(G_b)$;

(b) lancia $\mathscr{V}^*(1^n, (G_0, G_1), H; \omega)$ ed ottieni in cambio la risposta b';

(c) se $b = b'$ termina e restituisci (ω, H, b, φ').

3. Se per κ volte non si verifica mai $b = b'$, restituisci $\perp$.

Siano $(G_0, G_1) \in \mathcal{L}$, dobbiamo mostrare che la distribuzione dell'output del simulatore è indistinguibile dalla distribuzione dell'output di $\mathscr{V}^*$ in una normale esecuzione del protocollo. Per fare ciò, mostriamo prima che la probabilità che $\mathscr{Z}$ restituisca $\perp$ è trascurabile in κ. Osserviamo che, per ogni valore di ω, il valore b scelto (in ognuna delle κ interazioni) da $\mathscr{Z}$ è indipendente da H: il grafo H è una copia isomorfa di G_b che è distribuita esattamente come una copia isomorfa a caso di G_{1-b}, perché $G_0 \simeq G_1$. Ma allora, il valore di b è completamente indipendente dalla vista di $\mathscr{V}^*$ quando questi è lanciato con input $(1^n, (G_0, G_1), H; \omega)$. Ne segue che $b = b'$ con probabilità $1/2$ e quindi la probabilità che b sia sempre diverso da b' è $2^{-\kappa}$, che è trascurabile. Pertanto, $\mathscr{Z}$ restituisce $\perp$ con probabilità trascurabile.

Mostriamo infine che, condizionando sull'evento che $\mathscr{Z}$ non restituisca $\perp$, l'output di $\mathscr{Z}$ è indistinguibile dall'output di $\mathscr{V}^*$ in una normale esecuzione del protocollo. Consideriamo tutti gli elementi dell'output presi singolarmente, vale a dire (ω, H, b, φ'). La stringa ω è casuale, quindi è distribuita come in una normale interazione tra $\mathscr{P}$ e $\mathscr{V}$. Il grafo H è una copia isomorfa a caso di G_b per un qualche b, ma abbiamo già argomentato che questa è distribuita esattamente come una copia isomorfa casuale di G_0, come nel protocollo. Segue che il bit sfida b è una funzione deterministica di ω ed H (perché la randomicità di $\mathscr{V}^*$ è fissa) e quindi la sua distribuzione nell'esperimento reale ed in quello simulato è identica . Infine la mappa φ' è a distribuzione uniforme sugli isomorfismi da G_b in H, esattamente come nel protocollo reale. Concludiamo che il protocollo è a conoscenza nulla. □

Come osservato, il protocollo soddisfa la proprietà di validità con probabilità $1/2$ (non trascurabile). Per rendere l'errore di validità trascurabile possiamo pensare di ripetere il protocollo r volte, ottenendo così un fattore 2^{-r} che diventa trascurabile al crescere di r. Bisogna però stare attenti a controllare che, anche in questa variante, la proprietà di conoscenza nulla sia verificata! Esistono essenzialmente due modi per ripetere il protocollo: in parallelo oppure sequenzialmente. Eseguire il protocollo in parallelo significa che nel primo messaggio $\mathscr{P}$ invia r grafi $H_1, \ldots, H_r$, nel secondo messaggio $\mathscr{V}$ invia r sfide

$b_1, \ldots, b_r$, e nel terzo messaggio $\mathscr{P}$ risponde a ciascuna sfida come nello schema di Fig. 13.1. Sfortunatamente, non è noto se questa versione del protocollo è a conoscenza nulla, seppur non esista una prova del contrario.[80] La seconda possibilità è quella di lanciare il protocollo r volte in sequenza. Lo svantaggio è che la complessità cresce come $O(r)$. Tuttavia, la buona notizia è che si può dimostrare che la versione sequenziale conserva la proprietà di conoscenza nulla. (Facendo un salto in avanti, vedremo nel Paragrafo 13.2 che questo problema della ripetizione parallela è intrinseco di ogni sistemi di prova interattivo.)

Un sistema di prova ZK per ogni linguaggio in NP. Più in generale è possibile mostrare che ogni linguaggio in **NP** ammette un sistema di prova ZK. Per fare ciò, è sufficiente costruire un sistema di prova ZK per un linguaggio **NP**-completo. (Il linguaggio dei grafi isomorfi non è **NP**-completo.) In [GMW91; GK96a] si costruisce un tale sistema di prova.

Conoscenza nulla con verificatore onesto. Un modo per rilassare la Definizione 13.3 è supporre che $\mathscr{V}$ sia sempre onesto nell'esecuzione del protocollo e tenti di imparare qualcosa di non lecito solamente a partire da ciò che ha imparato al termine dell'interazione con $\mathscr{P}$. Ciò modella avversari *"onesti ma curiosi"* che intercettano un'esecuzione del protocollo tra $\mathscr{P}$ e $\mathscr{V}$ (in effetti questa nozione ricorda il caso di sicurezza passiva per i protocolli di autenticazione, cf. Paragrafo 11.2):

Definizione 13.4 (Conoscenza nulla con verificatore onesto). Sia $(\mathscr{P}, \mathscr{V})$ un sistema di prova interattivo per un linguaggio $\mathcal{L}$. Diremo che $(\mathscr{P}, \mathscr{V})$ è a conoscenza nulla con verificatore onesto (*Honest Verifier Zero Knowledge*, HVZK) se esiste un simulatore PPT $\mathscr{Z}$ tale che i seguenti insiemi di distribuzioni sono computazionalmente indistinguibili:

$$\{\mathscr{P}(1^n, x) \leftrightarrows \mathscr{V}(1^n, x)\} \overset{c}{\approx} \mathscr{Z}(1^n, x).$$

■

Non è difficile mostrare che, nel caso di verificatore onesto, la versione parallela del protocollo in Fig. 13.1 è anch'essa HVZK (cf. Esercizio 13.1).

[80] In realtà si crede che costruire il simulatore in questo caso sia difficile [GK96b].

Conoscenza nulla per input ausiliario. La definizione di conoscenza nulla che abbiamo discusso finora, richiede essenzialmente che tutto ciò che può essere ricavato interagendo con $\mathscr{P}$ *con input condiviso* x può essere simulato a partire solo dalla conoscenza di x. Si potrebbe obiettare che questa definizione non è abbastanza generale per applicazioni pratiche. Se, infatti, un protocollo a conoscenza nulla è usato come sottoparte di un protocollo più complesso, potrebbe essere che il verificatore possegga un qualche input ausiliario (possibilmente dipendente da x) da usare per estrarre informazione da $\mathscr{P}$. Per questo motivo si è soliti estendere la Definizione 13.3 considerando il caso in cui $\mathscr{V}^*$ ottenga un arbitrario input ausiliario $z \in \{0,1\}^*$. Si parla in questo caso [GO94; GK96b] di conoscenza nulla con input ausiliario (*Auxiliary Input Zero Knowledge*, AIZK).

Definizione 13.5 (Conoscenza nulla con input ausiliario). Sia $(\mathscr{P}, \mathscr{V})$ un sistema di prova interattivo per un linguaggio $\mathcal{L}$. Diremo che $(\mathscr{P}, \mathscr{V})$ è a conoscenza nulla con input ausiliario se per ogni algoritmo PPT $\mathscr{V}^*$ esiste un simulatore PPT $\mathscr{Z}$ tale che i seguenti insiemi di distribuzioni sono computazionalmente indistinguibili:

$$\{\mathscr{P}(1^n, x) \leftrightarrows \mathscr{V}^*(1^n, x, z)\}_{x \in \mathcal{L}, z \in \{0,1\}^*} \overset{c}{\approx} \{\mathscr{Z}(1^n, x, z)\}_{x \in \mathcal{L}, z \in \{0,1\}^*} .$$

■

Tutti i protocolli a conoscenza nulla che introdurremo in questo capitolo (ed anche quelli già discussi) soddisfano questa versione della definizione di conoscenza nulla. (Più in generale si può dimostrare che ciò vale sempre quando il simulatore utilizza l'algoritmo di $\mathscr{V}$ come scatola nera [GK96b; Bar01].) Il motivo per cui questa estensione è importante è che la versione sequenziale di un qualsiasi protocollo a conoscenza nulla con input ausiliario è anch'essa a conoscenza nulla con input ausiliario [GO94], mentre la stessa proprietà non vale per la nozione di conoscenza nulla della Definizione 13.3 (si veda [GK96b]).

Indistinguibilità d'indizi. In generale un problema in **NP** può avere più di un indizio. Ha senso chiedersi se un sistema di prova per un linguaggio $\mathcal{L}$ in **NP** lascia trapelare informazione su quale indizio è stato usato da $\mathscr{P}$ nell'interazione. Da qui la seguente definizione, dovuta a Fiege e Shamir [FS90]:

Definizione 13.6 (Indistinguibilità d'indizi). Sia $\mathcal{L}$ un linguaggio in **NP** e $(\mathscr{P}, \mathscr{V})$ un sistema di prova interattivo con completezza perfetta. Diremo che $(\mathscr{P}, \mathscr{V})$ ha indistinguibilità d'indizi (*Witness Indistinguishability*, WI) se, per

ogni algoritmo PPT $\mathscr{V}^*$ e per ogni insieme $\{w\}_{x\in\mathcal{L}}$ e $\{w'\}_{x\in\mathcal{L}}$ tale che w e w' sono indizi per x, i seguenti insiemi di distribuzioni sono computazionalmente indistinguibili:

$$\{\mathscr{P}(x,w) \leftrightarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathscr{P}(x,w') \rightleftarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*}.$$

■

È chiaro che la nozione d'indistinguibilità d'indizi è un requisito più debole della nozione di conoscenza nulla. In particolare, esistono protocolli che sono WI ma non AIZK. Ad esempio, quando esiste un solo indizio per un dato x, il protocollo che trasmette tale indizio è (banalmente) WI, ma non è ZK. D'altra parte si può mostrare che un protocollo AIZK è sempre WI:

Lemma 13.2 (AIZK ⇒ WI). *Se un sistema di prova interattivo $(\mathscr{P},\mathscr{V})$ per un linguaggio $\mathcal{L}$ in* **NP** *è AIZK esso è anche WI.*

Dimostrazione. Sia $\mathscr{Z}$ il simulatore di AIZK del sistema di prova. Abbiamo:

$$\{\mathscr{Z}(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathscr{P}(x,w) \rightleftarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*}$$
$$\{\mathscr{Z}(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathscr{P}(x,w') \rightleftarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*}.$$

Per transitività allora:

$$\{\mathscr{P}(x,w) \rightleftarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*} \overset{c}{\approx} \{\mathscr{P}(x,w') \rightleftarrows \mathscr{V}^*(x,z)\}_{x\in\mathcal{L},z\in\{0,1\}^*}.$$

□

La proprietà WI è nota essere invariante per composizione parallela [FS90] (i.e., ogni protocollo WI rimane WI anche quando eseguito in parallelo).

13.2 Risultati negativi

In questo paragrafo discutiamo alcuni risultati negativi relativi ai sistemi di prova interattivi a conoscenza nulla.

Sul ruolo dell'interazione. Un sistema di prova non-interattivo (detto anche unidirezionale) è un sistema di prova in cui $\mathscr{P}$ invia un unico messaggio. Mostreremo che un tale sistema di prova non può essere a conoscenza nulla. Più precisamente, gli unici linguaggi per cui è possibile costruire sistemi di prova non-interattivi a conoscenza nulla (senza fare ipotesi aggiuntive) sono i linguaggi banali.

Lemma 13.3 (L'interazione è necessaria). *Se $\mathcal{L}$ ammette un sistema di prova non-interattivo a conoscenza nulla, allora $\mathcal{L} \in$ **BPP**.*

Dimostrazione. Dato il simulatore $\mathscr{Z}$ del sistema di prova a conoscenza nulla non-interattivo per $\mathcal{L}$, definiremo un algoritmo PPT $\mathscr{M}$ in grado di decidere $\mathcal{L}$. Dato un input x, un'invocazione del simulatore $\mathscr{Z}$ genera una coppia (π, ω) in cui π è il (singolo) messaggio inviato da $\mathscr{P}$ ed $\omega \in \Omega$ è la randomicità di $\mathscr{V}$. Quindi, $\mathscr{M}$ fissa $\omega' \xleftarrow{\$} \Omega$ lancia $\mathscr{V}(1^n, x, \pi; \omega')$ e ritorna 1 (ovvero decide che $x \in \mathcal{L}$) se e solo se il verificatore ritorna `accetta`.

Ovviamente quando $x \in \mathcal{L}$, la proprietà di conoscenza nulla implica che la decisione di $\mathscr{M}$ è corretta (in quanto la distribuzione di (π, ω') è computazionalmente indistinguibile da quella in una reale esecuzione del protocollo). D'altra parte quando $x \notin \mathcal{L}$, la proprietà di validità implica che $\mathscr{M}$ ritorna 0 con probabilità trascurabilmente vicina ad 1, decidendo quindi correttamente anche in questo caso. Siccome $\mathscr{M}$ è PPT, concludiamo che $\mathcal{L} \in$ **BPP**. □

Nel Paragrafo 13.4 vedremo che esistono sistemi di prova non-interattivi a conoscenza nulla per linguaggi non banali, quando si fanno ipotesi ulteriori come l'esistenza di una stringa comune di riferimento oppure di un oracolo casuale.

Sul ruolo della randomicità di $\mathscr{P}$ e $\mathscr{V}$. Mostriamo ora che il fatto che dimostratore e verificatore di un sistema di prova a conoscenza nulla siano probabilistici è necessario, se il linguaggio $\mathcal{L}$ non è banale.

Teorema 13.4 (La randomicità del verificatore è necessaria). *Se il linguaggio $\mathcal{L}$ ammette un sistema di prova a conoscenza nulla con verificatore deterministico, allora $\mathcal{L} \in$ **BPP**.*

Dimostrazione. L'osservazione chiave è che, siccome il verificatore è deterministico, il dimostratore può determinare da solo il prossimo messaggio di $\mathscr{V}$.

Pertanto il sistema di prova può essere trasformato in un sistema di prova non-interattivo per lo stesso linguaggio; banalmente tale sistema di prova rimane a conoscenza nulla. L'asserto segue quindi dal Lemma 13.3. □

Un corollario di questo teorema è che non esiste un sistema di prova a conoscenza nulla con errore di validità nullo (per linguaggi non banali), perché altrimenti potremmo sempre richiedere che il verificatore usi la stringa tutta nulla come randomicità (ovvero il verificatore potrebbe essere rimpiazzato da uno deterministico).

Un risultato simile si ha per il dimostratore, solo che ora dobbiamo assumere che il sistema sia AIZK:

Teorema 13.5 (La randomicità del dimostratore è necessaria). *Se il linguaggio $\mathcal{L}$ ammette un sistema di prova a conoscenza nulla con input ausiliario e con dimostratore onesto deterministico, allora $\mathcal{L} \in \mathbf{BPP}$.*

Dimostrazione. Supponiamo che nel sistema di prova, sia $\mathscr{V}$ ad iniziare l'interazione. Consideriamo un verificatore scorretto $\mathscr{V}^*$ con input ausiliario $z_1, z_2, \ldots$ che invia z_i come messaggio i-simo. Gli altri messaggi sono scelti arbitrariamente. Poiché il dimostratore è deterministico, segue che i primi i messaggi inviati da $\mathscr{P}$ sono univocamente determinati da z_i (per ogni i). La stessa cosa, quindi, deve valere nella simulazione.

Costruiamo un algoritmo PPT $\mathscr{M}$ in grado di decidere $\mathcal{L}$. Dato x come input, $\mathscr{M}$ simula un'interazione tra $\mathscr{P}$ e $\mathscr{V}$ con input x. Per prima cosa $\mathscr{M}$ fissa la randomicità ω per $\mathscr{V}$. Dunque $\mathscr{M}$ determina il prossimo messaggio z_i di $\mathscr{V}$, lanciando $\mathscr{V}$ con input $x, \omega, (z_1, \ldots, z_{i-1})$. Quindi per determinare l'i-simo messaggio di $\mathscr{P}$, lancia $\mathscr{L}(1^n, x, (z_1, \ldots, z_i))$. Infine, $\mathscr{M}$ ritorna 1 se e solo se alla fine $\mathscr{V}$ ritorna `accetta`. □

Sulla composizione di sistemi di prova ZK. Essenzialmente esistono tre modi differenti di comporre un protocollo: in modo sequenziale, parallelo o concorrente. Osserviamo che quando si parla di composizione di un sistema di prova, si assume che le parti siano oneste in ogni singola interazione; in altri termini i messaggi scelti dalle parti oneste in una fissata interazione sono indipendenti dai messaggi ricevuti nelle altre iterazioni. Nel caso di parti disoneste avviene esattamente il contrario: nell'attacco ad un sistema composto, un avversario cercherà di legare i messaggi ricevuti in diverse istanze.

Nella versione sequenziale, il protocollo è ripetuto per r volte in sequenza. Ciò ha lo svantaggio di richiedere un numero sempre maggiore di turni. Come

abbiamo discusso nel paragrafo precedente [GK96b], i sistemi di prova AIZK mantengono le loro proprietà nel caso di composizione sequenziale. (Ricordiamo che ciò non si verifica, in generale, per i sistemi di prova a conoscenza nulla della Definizione 13.3.)

Nella versione parallela, vengono invocate un numero r di istanze allo stesso tempo. Ciò equivale ad assumere un modello di comunicazione completamente sincrono. Si può vedere che i sistemi di prova a conoscenza nulla, in generale, non sono invarianti per composizione parallela. In particolare possiamo costruire un protocollo a conoscenza nulla, la cui versione parallela non è a conoscenza nulla. Il protocollo è il seguente.

1. Supponiamo che un dimostratore $\mathscr{P}$ possegga un segreto w (ad esempio, l'indizio relativo ad una stringa x) ed una descrizione di una funzione casuale $f : \{0,1\}^{2n} \to \{0,1\}^n$.

2. Il dimostratore interagisce con un avversario $\mathscr{A}$ che specifica un valore $b \in \{0,1\}$.

   - Se $b = 1$, il dimostratore sceglie $\alpha \stackrel{\$}{\leftarrow} \{0,1\}^n$ e lo inoltra ad $\mathscr{A}$. Quindi $\mathscr{A}$ deve rispondere con una coppia $(\beta, \gamma) \in \{0,1\}^{2n}$. Se $f(\alpha, \beta) = \gamma$, il dimostratore invia ad $\mathscr{A}$ il segreto w.
   - Se $b = 1$, l'avversario $\mathscr{A}$ sceglie $\alpha \stackrel{\$}{\leftarrow} \{0,1\}^n$ e lo invia a $\mathscr{P}$. Quindi il dimostratore calcola $\gamma = f(\alpha, \beta)$ per un valore $\beta \stackrel{\$}{\leftarrow} \{0,1\}^n$ ed inoltra (β, γ).

Non è complesso mostrare che la strategia di $\mathscr{P}$ è a conoscenza nulla (anche con input ausiliario). Infatti nel caso $b = 0$, l'avversario può scegliere una coppia (β, γ) corretta solo con probabilità trascurabile. Quindi, in questo caso, $\mathscr{A}$ non impara nulla dall'interazione. D'altra parte se $b = 1$, l'avversario vede una stringa uniforme in $\{0,1\}^{2n}$.

La situazione è diversa se $\mathscr{A}$ può lanciare due istanze in parallelo. Nella prima sessione $\mathscr{A}$ sceglie $b = 0$, mentre nella seconda sceglie $b = 1$. Sia α messaggio che $\mathscr{P}$ invia nella prima istanza. L'avversario registra α e lo inoltra a $\mathscr{P}$ nella seconda istanza, ottenendo la coppia $(\alpha, f(\alpha, \beta))$. Quindi, $\mathscr{A}$ può inoltrare tale coppia a $\mathscr{P}$ nella prima istanza, ottenendo così il segreto di $\mathscr{P}$ con probabilità 1.

È importante osservare che, anche se la proprietà di conoscenza nulla non è invariante per composizione parallela, esistono protocolli a conoscenza nulla

specifici, per linguaggi in **NP**, che sono invarianti per composizione parallela. Tali protocolli [Gol02a; Gol06; DNS98] hanno inoltre un numero di round costante.

La modalità concorrente, infine, generalizza le due precedenti ammettendo che il modello di comunicazione sia asincrono. (Ovvero istanze arbitrarie del protocollo possono essere lanciate in istanti arbitrari.) Anche in questo caso, esistono esempi specifici di sistemi di prova a conoscenza nulla che sono invarianti per composizione concorrente [RK99; Can00; Bar01; Can+01; KP01; Gol02a; PRS02; Gol06].

Oltre la barriera della simulazione a scatola nera. La proprietà di conoscenza nulla richiede che tutto ciò che un avversario $\mathscr{V}^*$ può ottenere interagendo con il dimostratore, possa essere prodotto da un simulatore $\mathscr{Z}$ che non interagisce affatto con il dimostratore stesso. In un certo senso, ciò significa richiedere che esista una sorta di simulatore universale che prende come input ulteriore la descrizione della strategia di $\mathscr{V}^*$ (ovvero il suo algoritmo). La domanda è *come* il simulatore usa questo algoritmo. Fino al 2001, tutti gli esempi di sistemi a conoscenza nulla vedevano $\mathscr{Z}$ usare l'algoritmo di $\mathscr{V}^*$ come se fosse una scatola nera. Sembrava dunque che i risultati negativi inerenti i simulatori a scatola nera fossero intrinseci dei sistemi a conoscenza nulla.

Ad esempio Goldreich e Krawczyk [GK96b], hanno mostrato che la composizione parallela di nessun protocollo con numero di round costante ed errore di validità trascurabile può soddisfare la proprietà di conoscenza nulla usando un simulatore a scatola nera (se non nel caso banale di linguaggio in **BPP**). Ciò aveva portato a formulare la congettura che i sistemi di prova con numero di round costante ed errore di validità trascurabile non possono essere a conoscenza nulla. Un risultato simile era noto per il caso di composizione concorrente [Can+01].

Barak [Bar01] ha risolto la congettura in negativo, mostrando che esistono altri modi di usare $\mathscr{V}^*$. Ciò ha consentito di ottenere diversi risultati, non ottenibili con simulatori a scatola nera.

13.3 Protocolli-Σ

In questo paragrafo studieremo una particolare classe di sistemi di prova interattivi con la seguente struttura: il protocollo è costituito da tre messaggi (α, β, γ) e l'interazione è iniziata dal dimostratore. (Da qui la lettera greca Σ per indicare tali protocolli, come introdotto per la prima volta da Ronald

Cramer nella sua tesi di dottorato [Cra96].) Formalmente abbiamo la seguente definizione:

Definizione 13.7 (Protocolli-Σ). Sia $\mathcal{L} = \{x : \exists w \text{ per cui } \varrho(x, w) = 1\}$ un linguaggio per una relazione a tempo polinomiale ϱ. Un sistema di prova $(\mathscr{P}, \mathscr{V})$ a tre round — del tipo (α, β, γ) — è un protocollo-Σ, se è un sistema di prova HVZK (cf. Definizione 13.4) che soddisfa ulteriormente la seguente proprietà:

- *Validità Speciale (Special Soundness, SS).* Per ogni terna valida (α, β, γ), $(\alpha, \beta', \gamma')$ esiste $\mathscr{K}^*$ tale che, per $\beta \neq \beta'$, risulta:

$$w' \leftarrow \mathscr{K}^*(1^n, \alpha, \beta, \beta', \gamma, \gamma') \qquad \Rightarrow \qquad \varrho(x, w') = 1.$$

Nel seguito, indicheremo lo spazio delle sfide β con $\mathcal{B} = \{0,1\}^{\ell}$. ■

A partire dalla Definizione 13.7, si possono ottenere alcune varianti discusse brevemente di seguito. In primo luogo, possiamo considerare due generalizzazioni della proprietà di HVZK: (i) la proprietà di HVZK speciale (*Special Honest Verifier Zero Knowledge*, S-HVZK) e (ii) la proprietà HVZK speciale in senso forte (*Strong Special Honest Verifier Zero Knowledge*, SS-HVZK). Nel caso di S-HVZK, si richiede che, per ogni β nello spazio delle sfide $\mathcal{B}$, esista un simulatore $\mathscr{Z}^*$ tale che la coppia $(\alpha^*, \gamma^*) \leftarrow \mathscr{Z}^*(1^n, \beta)$ è (computazionalmente) indistinguibile da una coppia (α, γ) risultante da un'esecuzione di $\mathscr{P}(1^n, x)$ con sfida β.[81] Nel caso di SS-HVZK, si richiede ulteriormente che il simulatore possa scegliere γ a caso: esiste $\mathscr{Z}^{**}$ tale che, per ogni $\beta \in \mathcal{B}$, la coppia $(\alpha^{**}, \gamma^{**}) \leftarrow \mathscr{Z}^{**}(1^n, \beta)$ — dove ora γ^{**} è scelto uniformemente a caso — è (computazionalmente) indistinguibile da una coppia (α, γ) risultante da un'esecuzione di $\mathscr{P}(x)$ con sfida β.

In alcuni casi, come vedremo, si assume anche che la risposta γ ad una data sfida β sia unica, e che la stringa α abbia sufficiente entropia minima [Fis05].

La proprietà principale dei protocolli-Σ è che essi costituiscono automaticamente una "prova di conoscenza" (*Proof of Knowledge*, PoK). Per fare un paragone, un sistema di prova interattivo può essere visto come un modo per dimostrare che un data affermazione è vera, mentre una prova di conoscenza può essere interpretata come un modo per dimostrare *perché* quella stessa affermazione è vera. Ma cosa significa il fatto che una macchina di Turing conosce

[81] A questo proposito è possibile dimostrare [Fis05] che ogni protocollo-Σ che soddisfa la HVZK, soddisfa anche la S-HVZK se $\ell = O(\log(n))$. Si veda anche l'Esercizio 13.6.

qualcosa? Significa che $\mathscr{P}$ non solo è in grado di provare a $\mathscr{V}$ che x è in un qualche linguaggio $\mathcal{L}$, ma che in realtà possiede un indizio w relativamente al fatto che $x \in \mathcal{L}$. (Notare che esistono linguaggi per cui una prova che $x \in \mathcal{L}$ è banale, ad esempio se $\mathcal{L}$ è l'insieme di elementi della forma g^w per un generatore g di un gruppo $\mathbb{G}$; una prova di conoscenza è qualcosa di più, permette di mostrare che $\mathscr{P}$ conosce w tale che $g^w \in \mathbb{G}$.)

Per formalizzare questo concetto, iniziamo a considerare il caso semplice in cui esiste un dimostratore (eventualmente scorretto) $\mathscr{P}^*$ che è sempre in grado di convincere $\mathscr{V}$. Richiederemo che esista un algoritmo $\mathscr{K}$, che è in grado di estrarre un indizio w da $\mathscr{P}^*$. Si intende qui che $\mathscr{K}$ può interagire con $\mathscr{P}^*$ arbitrariamente (ad esempio riavvolgendolo) per ottenere w. (Se così non fosse sarebbe infatti impossibile ricavare l'indizio, in quanto per definizione $\mathscr{V}$ non impara niente dall'interazione con $\mathscr{P}^*$.) In questo senso, possiamo concludere che $\mathscr{P}^*$ deve conoscere w. Più in generale, se $\mathscr{P}^*$ è in grado di convincere $\mathscr{V}$ solo con una certa probabilità, richiederemo che $\mathscr{K}$ sia in grado di ottenere un indizio da $\mathscr{P}^*$ con circa la stessa probabilità [BG92].

Definizione 13.8 (Prova di conoscenza). Sia $(\mathscr{P}, \mathscr{V})$ un sistema di prova interattivo. Diremo che $(\mathscr{P}, \mathscr{V})$ è una prova di conoscenza se soddisfa quanto segue. Se esiste $\mathscr{P}^*$ tale che $\mathscr{P}^*(1^n, x) \leftrightarrows \mathscr{V}(1^n, x)$ restituisce `accetta` con probabilità almeno ϵ, allora esiste $\mathscr{K}$ tale che:

$$w' \leftarrow \mathscr{K}^{\mathscr{P}^*}(1^n, x) \qquad \Rightarrow \qquad \mathbb{P}\left[\varrho(x, w') = 1\right] \geq \text{poly}(\epsilon).$$

■

In effetti:

Lemma 13.6 (SS ⇒ PoK). *Ogni protocollo-Σ con spazio delle sfide* $\mathcal{B} = \{0,1\}^\ell$ *tale che* $\ell = \omega(\log n)$,[82] *è anche una prova di conoscenza.*

Dimostrazione. Supponiamo che esista $\mathscr{P}^*$ tale che:

$$\mathbb{P}\left[\mathscr{P}^*(1^n, x) \leftrightarrows \mathscr{V}(1^n, x) = \texttt{accetta}\right] \geq \epsilon.$$

Costruiamo $\mathscr{K}(1^n, x)$ in grado di calcolare (interagendo con $\mathscr{P}^*$) un indizio w tale che $\varrho(x, w) = 1$ con probabilità polinomiale in ϵ. L'algoritmo per $\mathscr{K}$ è il seguente:

1. Fissa la randomicità ω per $\mathscr{P}^*$. Scegli $\beta, \beta' \xleftarrow{\$} \mathcal{B}$.

[82] Quest'ipotesi è senza perdita di generalità, cf. Esercizio 13.5.

2. Lancia $\mathscr{P}^*(\cdot;\omega)$ con input β, β' ottenendo due prove $\pi = (\alpha, \beta, \gamma)$ e $\pi' = (\alpha, \beta', \gamma')$.

3. Ritorna $w \leftarrow \mathscr{K}^*(1^n, \alpha, \beta, \beta', \gamma, \gamma')$, essendo $\mathscr{K}^*$ l'estrattore della proprietà SS, cf. Definizione 13.7.

Ovviamente $\mathscr{K}$ ha successo se entrambe π e π' sono valide, ovvero se $\mathscr{P}^*$ ha successo entrambe le volte in cui è invocato. Purtroppo le invocazioni non sono indipendenti, quindi non possiamo concludere che $\mathscr{P}^*$ ha successo con probabilità ϵ^2.

Sia $\epsilon_x = \mathbb{P}[\mathscr{P}^* \text{ ha successo per un dato } x]$; abbiamo $\epsilon = \mathbb{E}_x[\epsilon_x]$. Sia inoltre $\epsilon_{x,\omega} = \mathbb{P}[\mathscr{P}^*(\cdot;\omega) \text{ ha successo per un dato } x]$; ovviamente $\epsilon_x = \mathbb{E}_\omega[\epsilon_{x,\omega}]$. Pertanto:

$$\begin{aligned}
\mathbb{P}[\mathscr{K} \text{ ha successo per un dato } x] &= \sum_{\omega\in\Omega} \frac{1}{|\Omega|}\epsilon_{x,\omega}\left(\epsilon_{x,\omega} - \frac{1}{|\mathcal{B}|}\right) \\
&= \mathbb{E}_\omega\left[\epsilon_{x,\omega}^2\right] - \frac{1}{|\mathcal{B}|}\mathbb{E}_\omega\left[\epsilon_{x,\omega}\right] \\
&\geq [\text{usando la disuguaglianza di Jensen}] \\
&\geq \left(\mathbb{E}_\omega\left[\epsilon_{x,\omega}\right]\right)^2 - \frac{\epsilon_x}{|\mathcal{B}|} \\
&= \epsilon_x^2 - \frac{\epsilon_x}{|\mathcal{B}|}.
\end{aligned}$$

Abbiamo così trovato:

$$\begin{aligned}
\mathbb{P}[\mathscr{K} \text{ ha successo}] &\geq \mathbb{E}_x\left[\epsilon_x^2 - \frac{\epsilon_x}{|\mathcal{B}|}\right] \\
&\geq [\text{usando la disuguaglianza di Jensen}] \\
&\geq \epsilon^2 - \frac{\epsilon}{|\mathcal{B}|},
\end{aligned}$$

che è polinomiale in ϵ poiché $1/|\mathcal{B}|$ è trascurabile. □

Lo svantaggio è che per avere successo con probabilità almeno $1/2$ bisogna lanciare $\mathscr{K}^{\mathscr{P}^*}$ circa $1/\epsilon^2$ volte. (In realtà è possibile migliorare la simulazione per ottenere un fattore $\approx 1/\epsilon$, ma non discuteremo tale estensione.)

Possiamo anche mostrare che ogni protocollo-Σ ha anche indistinguibilità di indizi.

Lemma 13.7 (Protocolli-Σ ⇒ WI). *I protocolli-Σ sono anche WI.*

Dimostrazione. Assumeremo che il protocollo-Σ sia in realtà S-HVZK (cf. nota a piè di pagina 81). Ciò significa che esiste un simulatore $\mathscr{Z}^*$ tale che, per ogni $\beta \in \mathcal{B}$, la coppia $(\alpha^*, \gamma^*) \leftarrow \mathscr{Z}^*(1^n, x, \beta)$ soddisfa $(\alpha^*, \gamma^*) \overset{c}{\approx} (\alpha, \gamma)$ come in una reale esecuzione del protocollo con sfida β.

Per ogni $\mathscr{V}$, costruiremo un algoritmo $\mathscr{M}$ (con input x) tale che, per ogni w che soddisfa $\varrho(x, w) = 1$, si ha:

$$\{\mathscr{M}(1^n, x) \leftrightarrows \mathscr{V}(1^n, x)\} \overset{c}{\approx} \{\mathscr{P}(1^n, x, w) \leftrightarrows \mathscr{V}(1^n, x)\}.$$

Siccome il primo membro non dipende dall'indizio w, e poiché l'equazione vale per ogni w, segue che per ogni coppia di indizi (w, w') tali che $\varrho(x, w) = \varrho(x, w') = 1$, si ha:

$$\{\mathscr{P}(1^n, x, w) \leftrightarrows \mathscr{V}(1^n, x)\} \overset{c}{\approx} \{\mathscr{P}(1^n, x, w') \leftrightarrows \mathscr{V}(1^n, x)\},$$

ovvero il protocollo è WI.

L'algoritmo per $\mathscr{M}$ è il seguente:

1. Fissa $\overline{\beta} \in \mathcal{B}$ e lancia $(\alpha, \overline{\gamma}) \leftarrow \mathscr{Z}^*(1^n, x, \overline{\beta})$. Invia α a $\mathscr{V}$.
2. Sia $\beta \in \mathcal{B}$ la sfida di $\mathscr{V}$. Lancia $\mathscr{Z}^*(1^n, x, \beta)$ fino ad ottenere una coppia (α, γ) in cui α è lo stesso del punto (1).
3. Rispondi con γ.

Notare che $\mathscr{M}$ termina, in quanto un verificatore onesto può scegliere il valore β usato al passo (2) con probabilità non nulla e quindi deve esistere γ tale che $(\alpha, \gamma) \leftarrow \mathscr{Z}(1^n, x, \beta)$. Ovviamente il tempo di esecuzione è esponenziale, ma ciò non è importante in quanto $\mathscr{M}$ è usato solamente per mostrare che il protocollo è WI. □

Concludiamo il paragrafo studiando alcuni esempi concreti di protocolli-Σ, basati su alcune relazioni a tempo polinomiale legate alla teoria dei numeri.

Il protocollo di Schnorr. Lo schema seguente è dovuto a Schnorr [Sch89]. Sia $\mathbb{G}$ un gruppo ciclico con ordine q e generatore g. Posto $h = g^w$, consideriamo il protocollo in Fig. 13.2 per il linguaggio $\mathcal{L} = \{(\mathbb{G}, g, h) : \exists\ w \text{ per cui } h = g^w\}$. Osserviamo che il protocollo soddisfa la proprietà di completezza in quanto, se

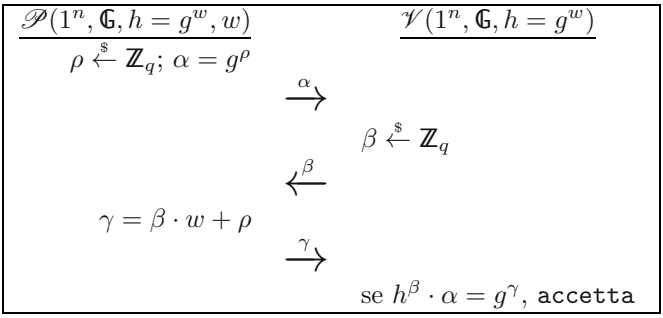


Fig. 13.2. Protocollo-Σ per dimostrare la conoscenza di $w = \log_g(h)$

$\mathscr{P}$ e $\mathscr{V}$ sono onesti, si ha:

$$g^\gamma = g^{\beta w + \rho} = g^{\beta w} g^\rho = h^\beta \cdot \alpha.$$

Inoltre, è semplice mostrare che il protocollo soddisfa la proprietà di validità se il problema CDH è difficile in $\mathbb{G}$. Mostriamo invece che sono verificate le proprietà di HVZK e di SS.

Per provare che il protocollo è HVZK dobbiamo costruire un simulatore $\mathscr{Z}$ in grado di produrre un prova indistinguibile da una reale, ottenibile da un verificatore onesto. Il simulatore sceglie semplicemente $\beta, \gamma \xleftarrow{\$} \mathbb{Z}_q$, calcola $\alpha = g^\gamma \cdot h^{-\beta}$ e restituisce (α, β, γ). Non è difficile accorgersi che, quando il verificatore è onesto, questa terna è identicamente distribuita all'output di un'esecuzione reale del protocollo. Infatti β è uniforme in $\mathbb{Z}_q$ in ambo i casi. D'altra parte, siccome g^γ è uniforme in $\mathbb{G}$ e quindi (per ogni β) $g^\gamma \cdot h^{-\beta}$ è uniforme in $\mathbb{G}$, possiamo concludere che anche α è uniforme in $\mathbb{G}$, come nel protocollo reale (in cui ρ è scelto a caso). Infine in entrambi i casi γ è calcolato in modo deterministico a partire da α e β, ovvero $\gamma = \beta \cdot \log_g h + \rho$ ha la stessa distribuzione nei due casi. (Non è difficile convincersi che in realtà $\mathscr{Z}$ è un simulatore per la proprietà di SS-HVZK.)

Per mostrare che il protocollo soddisfa la SS, dobbiamo provare che a partire da due prove valide $\pi = (\alpha, \beta, \gamma)$ e $\pi' = (\alpha, \beta', \gamma')$ possiamo costruire un estrattore $\mathscr{K}^*$ in grado di estrarre l'indizio w. Siccome π e π' sono prove valide, si deve avere:

$$g^\gamma = \alpha \cdot h^\beta \qquad g^{\gamma'} = \alpha \cdot h^{\beta'};$$

quindi, dividendo le due espressioni, si ottiene $g^{\gamma-\gamma'} = x^{\beta-\beta'}$. Ora, dato che $\beta \neq \beta'$, abbiamo $(\beta - \beta') \neq 0$ ed in particolare possiamo calcolare $(\beta - \beta')^{-1}$ usando l'algoritmo di Euclide esteso (cf. Appendice B.1). Pertanto:

$$g^{(\gamma-\gamma')(\beta-\beta')^{-1}} = h,$$

così che abbiamo estratto $w = (\gamma - \gamma')(\beta - \beta')^{-1}$.

Il protocollo di Guillou-Quisquater. Lo schema seguente è dovuto a Guillou e Quisquater [GQ88]. Sia $N = p \cdot q$ il prodotto di due primi forti. Sia inoltre e un primo non troppo piccolo, tale che $\gcd(e, \varphi(N)) = 1$. Posto $h = w^e \bmod N$, consideriamo il protocollo in Fig. 13.3 per il linguaggio $\mathcal{L} = \{(N, e, h) : \exists\ w \text{ per cui } h = w^e \bmod N\}$. Ovviamente il protocollo soddisfa la proprietà di completezza in quanto, se $\mathscr{P}$ e $\mathscr{V}$ sono onesti, si ha:

$$\gamma^e = (\rho \cdot w^\beta)^e = \rho^e \cdot w^{e\beta} = \rho^e \cdot (w^e)^\beta = \alpha \cdot h^\beta.$$

Inoltre, è semplice mostrare che la proprietà di validità è verificata se il problema dell'esponenziale modulare (cf. Definizione 6.5) è difficile in $\mathbb{Z}_N^*$ (la dimostrazione è lasciata come esercizio). Mostriamo invece che il protocollo soddisfa la SS-HVZK e la SS.

Sia $x = (N, e, h)$ un elemento nel linguaggio. Per provare la SS-HVZK dobbiamo dimostrare che esiste $\mathscr{Z}^{**}$ tale che, per ogni $\beta \in \mathbb{Z}_e$, il valore $\alpha^{**} \leftarrow \mathscr{Z}^{**}(1^n, x, \beta, \gamma^{**})$ — dove γ^{**} è scelto uniformemente a caso — rende la coppia $(\alpha^{**}, \gamma^{**})$ indistinguibile da una coppia (α, γ) risultante da un'esecuzione di

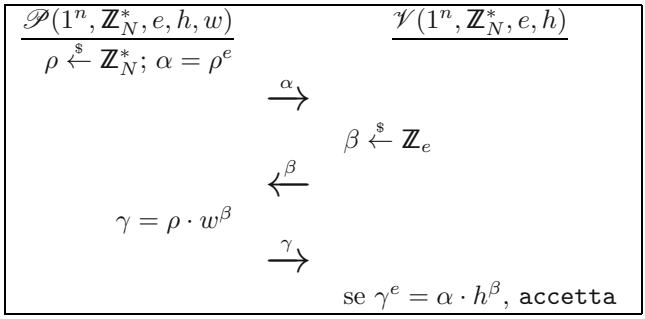


Fig. 13.3. Protocollo-Σ per la conoscenza di $w = \sqrt[e]{h} \in \mathbb{Z}_N^*$

$\mathscr{P}(x,w)$ quando la sfida è β. Il simulatore $\mathscr{Z}^{**}$ estrae $\gamma^{**} \xleftarrow{\$} \mathbb{Z}_N^*$ e calcola $\alpha^{**} = (\gamma^{**})^e \cdot h^{-\beta} \bmod N$. Notare che la distribuzione di γ^{**} è uniforme come in una reale esecuzione del protocollo e che, in entrambi i casi, $\alpha = \gamma^e \cdot h^{-\beta} \bmod N$ (e questo vale indipendentemente dal valore di β).

Resta da dimostrare che vale la SS. Per fare ciò abbiamo bisogno del trucco di Shamir:

Lemma 13.8 (Trucco di Shamir). *Supponiamo che $g^a = h^b \bmod N$ per $a, b > 0$, e sia $c = \gcd(a,b)$. Allora, esiste un algoritmo efficiente che calcola la radice a-sima di h^c, ovvero un elemento w tale che $w^a = h^c \bmod N$.*

Dimostrazione. Usando l'algoritmo di Euclide esteso possiamo trovare $u, v \in \mathbb{Z}$ che soddisfano l'identità di Béazout $au + bv = c$. Ciò equivale a dire $g^{av} = h^{bv} = h^{c-au}$ e $h^c = g^{av} \cdot h^{au} = (g^v \cdot h^u)^a$. Quindi, dati g ed h, basta porre $w = g^v \cdot h^u$. □

Dobbiamo costruire $\mathscr{K}^*$ in grado di estrarre un indizio da due prove valide $\pi = (\alpha, \beta, \gamma)$ e $\pi' = (\alpha, \beta', \gamma')$ per $x = (N, e, h)$. Siccome π e π' sono prove valide, deve essere $\gamma^e = \alpha \cdot h^\beta$ e $(\gamma')^e = \alpha \cdot h^{\beta'}$. Quindi:

$$\left(\frac{\gamma}{\gamma'}\right)^e = h^{\beta - \beta'}.$$

Applicando il trucco di Shamir, possiamo calcolare w tale che $w^e = h^{\gcd(e, \beta - \beta')}$. Siccome e è primo, $\beta \neq \beta'$ ed in particolare $\beta - \beta' < e$ (poiché $\beta, \beta' \in \mathbb{Z}_e$); pertanto $\gcd(e, \beta - \beta') = 1$. Abbiamo così calcolato w tale che $w^e = h \bmod N$, come desiderato.

Un protocollo per certi aspetti simili, nel caso $e = 2$ e basato sul problema della residuosità quadratica, è stato proposto da Goldwasser, Micali e Rabin [GMR89] (cf. Esercizio 13.11).

Il protocollo di Okamoto. Sia $\mathbb{G}$ un gruppo ciclico di ordine un primo q. Il seguente protocollo è dovuto ad Okamoto [Oka92]. Se g_1, g_2 sono generatori di $\mathbb{G}$, diremo che la coppia (w_1, w_2) è una *rappresentazione* di $h \in \mathbb{G}$, se possiamo scrivere $g_1^{w_1} g_2^{w_2} = h$. Osserviamo che, per ogni $h \in \mathbb{G}$ ed ogni $w_1 \in \mathbb{Z}_q$, esiste un unico $w_2 \in \mathbb{Z}_q$ tale che la coppia (w_1, w_2) è una rappresentazione di h. Ci sono quindi q possibili indizi per una data rappresentazione. Consideriamo il protocollo in Fig. 13.4 per il linguaggio $\mathcal{L} = \{(\mathbb{G}, g_1, g_2, h) : \exists\ (w_1, w_2) \text{ per cui } h = g_1^{w_1} g_2^{w_2}\}$.

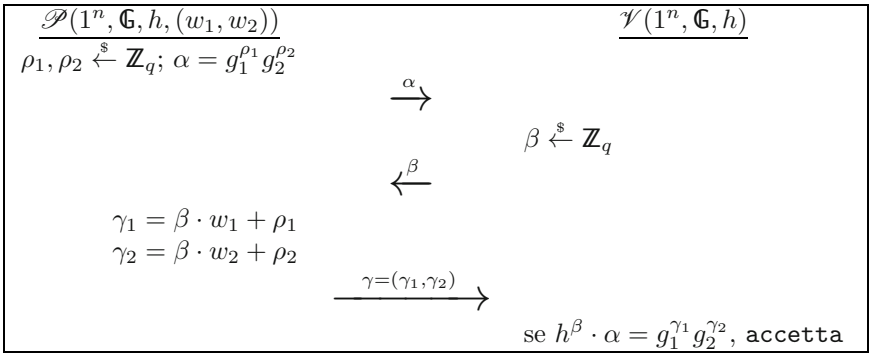


Fig. 13.4. Protocollo-Σ per la conoscenza di una rappresentazione (w_1, w_2) di $h = g_1^{w_1} g_2^{w_2}$

La proprietà di completezza è banalmente soddisfatta in quanto, se entrambi $\mathscr{P}$ e $\mathscr{V}$ sono onesti, abbiamo:

$$\alpha \cdot h^\beta = g_1^{\rho_1} g_2^{\rho_2} g_1^{\beta \cdot w_1} g_2^{\beta \cdot w_2} = g_1^{\rho_1 + \beta \cdot w_1} g_2^{\rho_2 + \beta \cdot w_2} = g_1^{\gamma_1} g_2^{\gamma_2}.$$

Inoltre, non è complesso mostrare che il protocollo soddisfa la proprietà di validità se il problema CDH è difficile (la prova è lasciata come esercizio). Mostriamo qui che sono verificate le proprietà di SS-HVZK e di SS.

Sia $x = (\mathbb{G}, g_1, g_2, h)$ un elemento del linguaggio. Per un dato β, il simulatore $\mathscr{Z}^{**}$ estrae $\gamma_1, \gamma_2 \xleftarrow{\$} \mathbb{Z}_q$ e calcola $\alpha = g_1^{\gamma_1} g_2^{\gamma_2} \cdot h^{-\beta}$. Non è complesso verificare che la distribuzione di $\gamma = (\gamma_1, \gamma_2)$ è uniforme come in un'esecuzione onesta del protocollo; inoltre in entrambi i casi $\alpha = g_1^{\gamma_1} g_2^{\gamma_2} \cdot h^{-\beta}$.

Quanto alla SS, mostriamo come costruire $\mathscr{K}^*$ che estrae un indizio a partire da due prove valide $\pi = (\alpha, \beta, \gamma)$ e $\pi' = (\alpha, \beta', \gamma')$ per x. Siccome π e π' sono prove valide, deve essere $g_1^{\gamma_1} g_2^{\gamma_2} = \alpha \cdot h^\beta$ e $g_1^{\gamma_1'} g_2^{\gamma_2'} = \alpha \cdot h^{\beta'}$. Quindi:

$$h^{\beta - \beta'} = g_1^{\gamma_1 - \gamma_1'} g_2^{\gamma_2 - \gamma_2'} \quad \Rightarrow \quad h = g_1^{\frac{\gamma_1 - \gamma_1'}{\beta - \beta'}} g_2^{\frac{\gamma_2 - \gamma_2'}{\beta - \beta'}},$$

così che $w = (w_1, w_2) = \left(\frac{\gamma_1 - \gamma_1'}{\beta - \beta'}, \frac{\gamma_2 - \gamma_2'}{\beta - \beta'} \right)$ è un indizio per $x = (\mathbb{G}, g_1, g_2, h)$.

Conoscenza nulla ed oltre. Abbiamo accennato al fatto che la proprietà di conoscenza nulla non è mantenuta, almeno in generale, quando si ripete un

protocollo interattivo in parallelo. Una proprietà interessante dei protocolli-Σ, è che tutte le proprietà sono invarianti per composizione parallela (cf. Esercizio 13.4).

I protocolli-Σ sono a conoscenza nulla solamente quando il verificatore è onesto. Ha senso chiedersi cosa accade nel caso di verificatori $\mathscr{V}^*$ scorretti, ovvero se i protocolli-Σ sono a conoscenza nulla oppure no. La risposta a questa domanda è affermativa quando lo spazio delle sfide $\mathcal{B} = \{0,1\}^\ell$ non è troppo grande. Questo fatto è chiaro se prendiamo il protocollo di Schnorr (cf. Fig. 13.2), quando la sfida β è costituita da un singolo bit. In questo caso, infatti, il simulatore può indovinare la sfida β che un verificatore scorretto $\mathscr{V}^*$ sceglierà, con probabilità $1/2$ e simulare correttamente un'esecuzione del protocollo (come di consueto). Purtroppo ora l'errore di validità è alto, il che rende necessario ripetere il protocollo in parallelo r volte, inficiando considerevolmente l'efficienza tipica dei protocolli-Σ. Una soluzione migliore consiste nell'utilizzare uno schema d'impegno (cf. Esercizio 14.9), il che consente di trasformare (in maniera efficiente) ogni protocollo-Σ in una prova di conoscenza a conoscenza nulla.

13.4 Conoscenza nulla non-interattiva

Come abbiamo visto (cf. Lemma 13.3) nel modello standard non esiste un sistema di prova non-interattivo a conoscenza nulla, se non per linguaggi banali. In questo paragrafo vedremo come costruire sistemi di prova non-interattivi a conoscenza nulla facendo ulteriori ipotesi, in particolare assumendo l'esistenza di una stringa comune di riferimento, oppure di un oracolo casuale.

Modello della stringa comune di riferimento. Nel modello della stringa comune di riferimento (*Common Reference String*, CRS) si ipotizza che il dimostratore ed il verificatore condividano (oltre all'input comune x) una stringa comune di riferimento δ a distribuzione uniforme su un insieme Δ. Assumeremo implicitamente $\Delta = \{0,1\}^d = \{0,1\}^{\mathrm{poly}(n)}$.

Definizione 13.9 (Sistema di prova non-interattivo). Sia $n \in \mathbb{N}$ un parametro statistico di sicurezza. Una coppia di algoritmi PPT $(\mathscr{P}, \mathscr{V})$ è detta un sistema di prova non-interattivo per un linguaggio $\mathcal{L}$ se soddisfa quanto segue:

- *Completezza.* Per ogni $x \in \mathcal{L}$

$$\mathbb{P}\left[\mathscr{V}(1^n, x, \delta, \pi) = \texttt{accetta} : \ \delta \xleftarrow{\$} \Delta, \pi \leftarrow \mathscr{P}(1^n, x, \delta)\right] \geq 1 - \mathrm{negl}(n).$$

- *Validità.* Per ogni $x \notin \mathcal{L}$ e per ogni algoritmo $\mathscr{P}^*$ (anche computazionalmente illimitato)

$$\mathbb{P}\left[\mathscr{V}(1^n, x, \delta, \pi) = \texttt{accetta} : \ \delta \xleftarrow{\$} \Delta, \pi \leftarrow \mathscr{P}^*(1^n, x, \delta)\right] \leq \text{negl}(n).$$

■

La definizione di conoscenza nulla non-interattiva (*Non-Interactive Zero Knowledge*, NIZK), è semplificata dal fatto che appunto non c'è interazione tra $\mathscr{P}$ e $\mathscr{V}$. In particolare possiamo ignorare completamente il verificatore, in quanto ciò che esso produce può essere ricavato a partire da δ e dalla prova π.

Definizione 13.10 (Conoscenza nulla non-interattiva, NIZK). Un sistema di prova non-interattivo per un linguaggio $\mathcal{L}$ è a conoscenza nulla se esiste un simulatore PPT $\mathscr{Z}$ tale che i seguenti insiemi di distribuzioni sono indistinguibili:

$$\{x, U_d, \mathscr{P}(1^n, x, U_d)\}_{x \in \mathcal{L}} \overset{c}{\approx} \{x, \mathscr{Z}(1^n, x)\}_{x \in \mathcal{L}}.$$

È importante osservare che durante la simulazione, il simulatore produce anche la stringa comune di riferimento δ; è proprio questa caratteristica che consente di evitare il risultato negativo espresso dal Lemma 13.3. Considereremo anche una versione più forte della proprietà di conoscenza nulla che chiameremo conoscenza nulla *adattiva*, in cui l'input comune x può essere scelto in modo adattivo dopo aver visto la stringa comune di riferimento. (La stessa variante può essere enunciata anche relativamente alla proprietà di validità.)

Il paradigma di Naor-Yung. I sistemi di prova non-interattivi hanno diverse applicazioni pratiche. Discutiamo qui come sia possibile combinare un cifrario a chiave pubblica CPA sicuro con un sistema di prova NIZK, per ottenere sicurezza CCA. Lo schema è dovuto a Naor ed Yung [NY90] ed è mostrato nel Crittosistema 13.1 a pagina 373.

Intuitivamente la prova π ha lo scopo di dimostrare che i crittotesti c_1, c_2 sono cifrature dello stesso messaggio m corrispondenti alle chiavi pk_1, pk_2, vale a dire c_1, c_2 sono elementi del linguaggio:

$$\mathcal{L} = \left\{(c_1, c_2) : \exists\ (m, \omega_1, \omega_2) \text{ t.c. } c_1 \leftarrow \mathsf{Enc}_{pk_1}(m; \omega_1), c_2 \leftarrow \mathsf{Enc}_{pk_2}(m; \omega_2)\right\}.$$

Notare che $\mathcal{L} \subset \mathbf{NP}$, in quanto la stringa $w = (m, \omega_1, \omega_2)$ è un indizio per un elemento $x = (c_1, c_2) \in \mathcal{L}$. Il seguente teorema mostra che Π_{NY} è CCA1-sicuro:

Teorema 13.9 (Π_{NY} è CCA1-sicuro). *Se Π_E è CPA-$(t_{\mathsf{CPA}}, Q, \epsilon_{\mathsf{CPA}})$-sicuro e Π_{NIZK} è un sistema di prova NIZK adattivo con errore di validità ϵ_s e con errore di conoscenza nulla ϵ_{NIZK}, allora il cifrario Π_{NY} è CCA1-$(t_{\mathsf{NY}}, Q, \epsilon_{\mathsf{NY}})$-sicuro, con*

$$t_{\mathsf{NY}} \approx t_{\mathsf{CPA}} \qquad \epsilon_{\mathsf{NY}} \leq 2\epsilon_{\mathsf{NIZK}} + \epsilon_{\mathsf{CPA}} + \epsilon_s.$$

Dimostrazione. Ricordiamo che nel caso di sicurezza CCA1, l'attaccante (nell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{NY}}}(\mathscr{A}, n)$ di Definizione 6.4) ha accesso all'oracolo di decifratura solamente *prima* di scegliere i messaggi (m_0, m_1) . Per dimostrare il teorema passeremo per una sequenza di giochi che mostreremo essere computazionalmente indistinguibili, grazie alle ipotesi sulle primitive componenti lo schema di Naor-Yung. I giochi estremi sono definiti di seguito:

Gioco G_0.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\delta \xleftarrow{\$} \Delta$
$pk^* = (pk_1, pk_2, \delta)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow \mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(\boxed{m_0}; \omega_1)$
$c_2 \leftarrow \mathsf{Enc}_{pk_2}(\boxed{m_0}; \omega_2)$
$\pi \leftarrow \mathscr{P}(1^n, (c_1, c_2), \delta, (\omega_1, \omega_2, \boxed{m_0}))$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

Gioco G_4.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\delta \xleftarrow{\$} \Delta$
$pk^* = (pk_1, pk_2, \delta)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow \mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(\boxed{m_1}; \omega_1)$
$c_2 \leftarrow \mathsf{Enc}_{pk_2}(\boxed{m_1}; \omega_2)$
$\pi \leftarrow \mathscr{P}(1^n, (c_1, c_2), \delta, (\omega_1, \omega_2, \boxed{m_1}))$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

Useremo la notazione $\boxed{\cdot}$ per indicare le differenze incrementali tra un gioco e l'altro. Una semplice ispezione dei due giochi sopra definiti è sufficiente a concludere che il primo gioco è equivalente ad un'esecuzione dell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{NY}}}(\mathscr{A}, n)$ in cui l'attaccante vede una cifratura di m_0, mentre il secondo gioco è equivalente ad un'esecuzione dell'esperimento $\mathbf{Exp}^{\mathrm{ind-cca}}_{\mathsf{PKE},\Pi_{\mathsf{NY}}}(\mathscr{A}, n)$ in cui l'attaccante vede una cifratura di m_1. Pertanto, per dimostrare l'asserto, basta provare che $|\mathbb{P}[G_0 = 1] - \mathbb{P}[G_4 = 1]|$ è trascurabile (cf. Esercizio 6.1).

Gioco G_1. Questo gioco è identico al gioco G_0, con la sola differenza che la stringa comune di riferimento δ e la prova π sono "simulate" usando il simulatore di ZK $\mathscr{Z}$.

Crittosistema 13.1. Il paradigma di Naor e Yung

Sia $\Pi_E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ un crittosistema a chiave pubblica con spazio della randomicità Ω e $\Pi_{\mathsf{NIZK}} = (\mathscr{P}, \mathscr{V})$ un sistema di prova NIZK adattivo per linguaggi in **NP**, con spazio delle stringhe comuni di riferimento Δ. Il cifrario $\Pi_{\mathsf{NY}} = (\mathsf{Gen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ è definito come segue:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo Gen^* usa due volte Gen per derivare due coppie di chiavi: $(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^n)$ e $(pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$. Estrae poi la stringa comune di riferimento $\delta \xleftarrow{\$} \Delta$ da usare nel sistema di prova. Infine pone $pk^* = (pk_1, pk_2, \delta)$ ed $sk^* = sk_1$ (la chiave sk_2 è scartata).
- `Cifratura.` Dato il messaggio m da cifrare, l'algoritmo di cifratura estrae due stringhe casuali $\omega_1, \omega_2 \xleftarrow{\$} \Omega$ ed usa Enc per cifrare due volte m calcolando $c_1 \leftarrow \mathsf{Enc}_{pk_1}(m; \omega_1)$ e $c_2 = \mathsf{Enc}_{pk_2}(m; \omega_2)$. Quindi viene generata la prova $\pi \leftarrow \mathscr{P}(1^n, x, \delta, w)$ per la stringa $x = (c_1, c_2)$ con indizio $w = (m, \omega_1, \omega_2)$. Il crittotesto è quindi $(c_1, c_2, \pi) \leftarrow \mathsf{Enc}^*_{pk^*}(m)$.
- `Decifratura.` Dato un testo cifrato $c = (x, \pi)$, se $\mathscr{V}(1^n, \delta, x, \pi) =$ `rifiuta` restituisci $\perp$. Altrimenti calcola $m = \mathsf{Dec}_{sk^*}(c) = \mathsf{Dec}_{sk_1}(c_1)$.

Gioco G_1.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\boxed{\delta \leftarrow \mathscr{Z}(1^n)}$
$pk^* = (pk_1, pk_2, \delta)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow \mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_0; \omega_1)$
$c_2 \leftarrow \mathsf{Enc}_{pk_2}(m_0; \omega_2)$
$\boxed{\pi \leftarrow \mathscr{Z}(1^n, (c_1, c_2))}$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

Mostriamo che G_1 e G_0 sono indistinguibili. Supponiamo che esista un attaccante PPT $\mathscr{A}$ in grado di distinguere G_0 da G_1. Possiamo usare $\mathscr{A}$ per costruire un attaccante PPT $\mathscr{D}$ in grado di distinguere prove "reali" da prove "simulate", contro l'ipotesi che Π_{NIZK} sia un sistema di prova NIZK. L'avversario $\mathscr{D}$ riceve

una sfida $(x = (c_1, c_2), \delta, \pi)$ relativa ad un elemento x nel linguaggio $\mathcal{L}$, e deve stabilire se essa è stata prodotta usando Π_{NIZK} oppure "simulata" usando $\mathscr{Z}$.

1. Simula la generazione delle chiavi in Π_{NY}: estrai $(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$, poni $pk^* = (pk_1, pk_2)$ ed $sk^* = sk_1$.
2. Quando $\mathscr{A}$ effettua una richiesta all'oracolo Dec_{sk^*} rispondi usando sk^*.
3. Quando $\mathscr{A}$ sceglie i due messaggi (m_0, m_1), estrai $\omega_1, \omega_2 \leftarrow \Omega$ e calcola $c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_0; \omega_1)$ e $c_2 \leftarrow \mathsf{Enc}_{pk_2}(m_0; \omega_2)$.
4. Lancia $\mathscr{A}(pk^*, c_1, c_2, \pi)$ e ritorna lo stesso bit b' che ritorna $\mathscr{A}$.

Ovviamente quando δ è una stringa veramente casuale e π una prova "reale" per l'elemento x nel linguaggio $\mathcal{L}$, la simulazione eseguita da $\mathscr{D}$ è identica ad un'esecuzione del gioco G_0. D'altra parte quando δ e π sono "simulate", la distribuzione dell'esperimento simulato da $\mathscr{D}$ coincide con quella di un'esecuzione del gioco G_1. (Notare che affinché ciò sia vero, abbiamo bisogno della proprietà di conoscenza nulla adattiva in quanto il simulatore deve poter scegliere x *dopo* aver visto δ.) Siccome Π_{NIZK} è a conoscenza nulla, si deve avere:

$$|\mathbb{P}[G_0 = 1] - \mathbb{P}[G_1 = 1]| \leq \epsilon_{\mathsf{NIZK}}.$$

Gioco G_2. Il gioco G_2 è diverso dal gioco G_1 in quanto invece di cifrare due volte m_0, cifreremo una volta m_0 e l'altra m_1.

Gioco G_2.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\delta \leftarrow \mathscr{Z}(1^n)$
$pk^* = (pk_1, pk_2, \delta)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow \mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_0; \omega_1)$
$\boxed{c_2 \leftarrow \mathsf{Enc}_{pk_2}(m_1; \omega_2)}$
$\pi \leftarrow \mathscr{Z}(1^n, (c_1, c_2))$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

Osserviamo che in questo caso il simulatore riceve come input la coppia $x = (c_1, c_2)$ in cui i due crittotesti sono relativi a messaggi *diversi*. Siccome tale elemento x non è in $\mathcal{L}$, non possiamo dire molto sull'output di $\mathscr{Z}$. Tuttavia,

useremo il fatto che Π_E è CPA-sicuro per mostrare che l'output nel gioco G_1 e nel gioco G_2 sono indistinguibili.

Possiamo, infatti, usare un avversario PPT $\mathscr{A}$ (in grado di distinguere i due giochi) per costruire un attaccante PPT $\mathscr{B}_E$ in grado di violare la sicurezza semantica di Π_E. L'avversario $\mathscr{B}_E$ deve distinguere la cifratura di due messaggi m_0, m_1 (a sua scelta) conoscendo solamente la chiave pubblica pk del cifrario Π_E. L'algoritmo di $\mathscr{B}_E$ è il seguente.

1. Simula la generazione delle chiavi in Π_{NY}: poni $pk_2 = pk$ ed estrai $(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^n)$. Usa il simulatore del sistema di prova per ottenere $\delta \leftarrow \mathscr{Z}(1^n)$. Ritorna $pk^* = (pk_1, pk_2, \delta)$ ed $sk^* = sk_1$.

2. Quando $\mathscr{A}$ effettua una richiesta all'oracolo Dec_{sk^*} rispondi usando sk^*.

3. Quando $\mathscr{A}$ sceglie i due messaggi (m_0, m_1), dichiara di voler distinguere la cifratura degli stessi messaggi.

4. Ricevi il crittotesto sfida c_2 (corrispondente ad una cifratura con chiave pk di uno tra m_0 ed m_1 usando una stringa casuale $\omega_2 \xleftarrow{\$} \Omega$ non nota). Estrai $\omega_1 \xleftarrow{\$} \Omega$ e calcola $c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_0; \omega_1)$.

5. Usa il simulatore del sistema di prova per ottenere $\pi \leftarrow \mathscr{Z}(1^n, c_1, c_2)$. Infine lancia $\mathscr{A}(pk^*, c_1, c_2, \pi)$ e ritorna lo stesso bit deciso da $\mathscr{A}$.

Ovviamente quando il crittotesto c_2 è una cifratura con chiave $pk = pk_2$ del messaggio m_0, la vista di $\mathscr{A}$ nella simulazione effettuata da $\mathscr{B}_E$ è identica alla vista di $\mathscr{A}$ nel gioco G_1. D'altra parte quando il crittotesto c_2 è una cifratura con chiave $pk = pk_2$ del messaggio m_1, la vista di $\mathscr{A}$ nella simulazione effettuata da $\mathscr{B}_E$ è identica alla vista di $\mathscr{A}$ nel gioco G_2. Quindi la probabilità che $\mathscr{A}$ distingua G_1 da G_2 è esattamente la probabilità che $\mathscr{B}_E$ distingua una cifratura di m_0 da una cifratura di m_1 calcolate usando Π_E. Siccome Π_E è CPA-sicuro, possiamo concludere:

$$|\mathbb{P}[G_1 = 1] - \mathbb{P}[G_2 = 1]| \leq \epsilon_{\mathsf{CPA}}.$$

Gioco G_2'. A questo punto saremmo tentati di sostituire m_0 con m_1 nel calcolo di c_1. Purtroppo, non possiamo farlo in modo diretto! Il motivo è che per provare che il gioco ottenuto è indistinguibile dal gioco precedente, dovremmo costruire un attaccante $\mathscr{B}_E$ che riceve una sfida c_1 prodotta usando $pk = pk_1$ e deve distinguere se si tratta di una cifratura di m_0 oppure di una

cifratura di m_1. A tale scopo $\mathscr{B}_E$ dovrebbe simulare l'oracolo di cifratura per $\mathscr{A}$, il che richiede sk_1 che ora $\mathscr{B}_E$ non conosce.

Per risolvere questo problema passeremo per un gioco intermedio in cui sostituiremo sk_1 con sk_2 in decifratura.

Gioco G_2'.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\delta \leftarrow \mathscr{Z}(1^n)$
$pk^* = (pk_1, pk_2, \delta)$, $\boxed{sk^* = sk_2}$
$(m_0, m_1) \leftarrow \boxed{\mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)}$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_0; \omega_1)$
$c_2 \leftarrow \mathsf{Enc}_{pk_2}(m_1; \omega_2)$
$\pi \leftarrow \mathscr{Z}(1^n, (c_1, c_2))$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

Sia $\mathcal{E}$ l'evento in cui $\mathscr{A}$ invia una richiesta (c_1, c_2, π) all'oracolo di decifratura tale che $\mathscr{V}(1^n, \delta, (c_1, c_2), \pi) = \texttt{accetta}$, ma $\mathsf{Dec}_{sk_1}(c_1) \neq \mathsf{Dec}_{sk_2}(c_2)$. Osserviamo innanzitutto che fin tanto che $\mathcal{E}$ non si verifica, il gioco G_2 ed il gioco G_2' sono indistinguibili, in quanto se c_1 e c_2 cifrano lo stesso messaggio, non fa alcuna differenza decifrare usando sk_1 o sk_2. Quindi c'è una differenza tra G_2 e G_2' solo quando $\mathcal{E}$ si verifica. D'altra parte è immediato accorgersi che la probabilità di tale evento è la stessa nei due giochi. Pertanto basta mostrare che $\mathbb{P}[\mathcal{E} \text{ in } G_2]$ è trascurabile.

Non è difficile convincersi che $\mathbb{P}[\mathcal{E} \text{ in } G_2] = \mathbb{P}[\mathcal{E} \text{ in } G_1]$, perché i due giochi G_1 e G_2 sono completamente identici nella parte in cui l'evento $\mathcal{E}$ può verificarsi (cioè fino a quando $\mathscr{A}$ può interrogare l'oracolo di decifratura). Infine, notiamo che la probabilità dell'evento $\mathcal{E}$ nel gioco originale G_0 non è altro che la probabilità di generare una prova π valida per un elemento $x = (c_1, c_2) \notin \mathcal{L}$; tale probabilità è al più ϵ_s perché Π_{NIZK} soddisfa la proprietà di validità.

È sufficiente, quindi, provare che $|\mathbb{P}[\mathcal{E} \text{ in } G_1] - \mathbb{P}[\mathcal{E} \text{ in } G_0]|$ è trascurabile per concludere. A tale scopo mostriamo come costruire un attaccante $\mathscr{D}$ in grado di distinguere una stringa comune di riferimento casuale da una simulata usando il simulatore del sistema di prova. L'algoritmo di $\mathscr{D}$ è il seguente.

1. Ricevi come input la sfida δ. Lancia $(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$ e poni $pk^* = (pk_1, pk_2, \delta)$.
2. Quando $\mathscr{A}$ interroga l'oracolo di decifratura rispondi alle sue richieste normalmente tranne quando $\mathscr{A}$ effettua una richiesta (c_1, c_2, π) tale per

cui $\mathscr{V}(1^n, \delta, (c_1, c_2), \pi) = \texttt{accetta}$, ma $\mathsf{Dec}_{sk_1}(c_1) \neq \mathsf{Dec}_{sk_2}(c_2)$. (Osserviamo che al passo (1) non bisogna scartare la chiave sk_2 in quanto serve per effettuare quest'ultima verifica.) Quando ciò accade ritorna 1 e fermati.

3. Altrimenti, quando $\mathscr{A}$ ha terminato la prima fase dell'attacco (cioè prima che $\mathscr{A}$ scelga i messaggi m_0, m_1) ritorna 0 e fermati.

Ovviamente quando la stringa δ è casuale, la probabilità che $\mathscr{D}$ ritorni 1 è esattamente la probabilità che l'evento $\mathcal{E}$ si verifichi in G_0. D'altra parte, se δ è la stringa prodotta dal simulatore, la probabilità che $\mathscr{D}$ restituisca 1 è esattamente la probabilità che si verifichi l'evento $\mathcal{E}$ in G_1. Poiché però Π_{NIZK} è a conoscenza nulla, deve essere:

$$|\mathbb{P}[\mathcal{E} \text{ in } G_1] - \mathbb{P}[\mathcal{E} \text{ in } G_0]| \leq \epsilon_{\mathsf{NIZK}}.$$

Siccome abbiamo già osservato che $\mathbb{P}[\mathcal{E} \text{ in } G_0] \leq \epsilon_s$, abbiamo che $\mathbb{P}[\mathcal{E} \text{ in } G_2] = \mathbb{P}[\mathcal{E} \text{ in } G_1] \leq \epsilon_{\mathsf{NIZK}} + \epsilon_s$ è trascurabile, come desiderato.

Gioco G_3. Possiamo ora sostituire il messaggio m_0 con il messaggio m_1 nel calcolo di c_1.

Gioco G_3.
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^n)$
$\delta \leftarrow \mathscr{Z}(1^n)$
$pk^* = (pk_1, pk_2, \delta)$, $sk^* = sk_2$
$(m_0, m_1) \leftarrow \mathscr{A}^{\mathsf{Dec}_{sk^*}(\cdot)}(pk^*)$
$\omega_1, \omega_2 \xleftarrow{\$} \Omega$
$\boxed{c_1 \leftarrow \mathsf{Enc}_{pk_1}(m_1; \omega_1)}$
$c_2 \leftarrow \mathsf{Enc}_{pk_2}(m_1; \omega_2)$
$\pi \leftarrow \mathscr{Z}(1^n, (c_1, c_2))$
$b' \leftarrow \mathscr{A}(pk^*, c_1, c_2, \pi)$.

La dimostrazione che $|\mathbb{P}[G_3 = 1] - \mathbb{P}[G_2' = 1]|$ è trascurabile è simile a quella che abbiamo usato nella transizione da G_1 a G_2 ed è pertanto omessa.

Gioco G_3'. In questo gioco ritorniamo ad usare sk_1 al posto di sk_2 in decifratura. La dimostrazione che G_3' e G_3 sono indistinguibili è identica a quella usata per passare da G_2 a G_2' ed è pertanto omessa.

Gioco G_4. L'ultimo gioco è quello che abbiamo già definito all'inizio della dimostrazione. Osserviamo che esso è identico al gioco G'_3 con la differenza che la stringa δ e la prova π "simulate" sono sostituite con quelle reali. La dimostrazione che G_4 e G'_3 sono indistinguibili è identica a quella usata per passare da G_0 a G_1 ed è pertanto omessa.

Sommando tutti i termini otteniamo l'asserto, concludendo la dimostrazione. □

Non è complesso vedere che lo schema di Naor-Yung non è in generale CCA2-sicuro. In particolare, esiste sempre un sistema di prova Π_{NIZK} tale che Π_{NY} può essere violato con un attacco CCA2. Sia $(\mathscr{P}', \mathscr{V}')$ un sistema di prova NIZK adattivo; costruiamo il sistema di prova $(\mathscr{P}, \mathscr{V})$ definito come segue:

$$\mathscr{P}(1^n(c_1, c_2), \delta, (\omega_1, \omega_2)) = \mathscr{P}'(1^n, (c_1, c_2), \delta, (\omega_1, \omega_2)) \mid\mid 0$$
$$\mathscr{V}(1^n, (c_1, c_2), \delta, \pi \mid\mid 0) = \mathscr{V}'(1^n, (c_1, c_2), \delta, \pi).$$

Detto a parole, l'algoritmo $\mathscr{P}$ lancia semplicemente $\mathscr{P}'$ ed aggiunge il bit 0 in fondo alla prova π ottenuta, mentre $\mathscr{V}$ scarta l'ultimo bit della prova e lancia $\mathscr{V}'$ per verificare una prova $\pi||0$. È facile mostrare che $(\mathscr{P}, \mathscr{V})$ è ancora un sistema di prova NIZK adattivo. Purtroppo, se usiamo questo schema in Π_{NY} possiamo violare il cifrario con un attacco CCA2. L'idea alla base dell'attacco è molto semplice e sfrutta il fatto che $\mathscr{V}$ ignora l'ultimo bit nella verifica di una prova: l'avversario sceglie due messaggi (m_0, m_1) e riceve il crittotesto sfida $((c_1, c_2), \pi||0)$; sostituisce quindi $\pi||0$ con $\pi||1$ ed invia all'oracolo di decifratura $((c_1, c_2), \pi||1)$ ottenendo così il messaggio corrispondente al crittotesto sfida. (Ripercorrendo la dimostrazione, non è difficile convincersi che in questo caso la probabilità dell'evento $\mathcal{E}$ non è più trascurabile).

Un soluzione al problema è stata proposta da Dolev, Dwork e Naor [DDN91] usando uno schema di firma digitale monouso. Un'altra possibilità [Sah99; Lin03], senza ricorrere alle firme digitali, è quella di usare un sistema di prova NIZK con validità di simulazione (*Simulation Soundness*).

L'uso del paradigma di Naor-Yung genera solitamente cifrari poco efficienti, a causa dell'inefficienza intrinseca dei sistemi di prova NIZK. Recentemente Cramer, Hofheinz e Kiltz [CHK10] hanno mostrato come sia possibile applicare il paradigma di Naor-Yung evitando l'uso dei sistemi di prova NIZK, basandosi solamente sull'ipotesi CDH oppure RSA ed ottenendo così cifrari CCA-sicuri ed allo stesso tempo efficienti.

Non vedremo una costruzione esplicita di sistemi di prova NIZK. Comunque si può mostrare che tali sistemi esistono ed in particolare che, se esistono

le permutazioni con botola, ogni linguaggio in **NP** ammette un sistema di prova NIZK adattivo [LS90; FLS90; Fei90; FLS99]. Siccome abbiamo già visto che l'esistenza dei sistemi di prova NIZK è sufficiente per costruire cifrari CCA-sicuri, ne deriva il corollario che l'esistenza delle permutazioni con botola implica l'esistenza di cifrari CCA-sicuri.

NIZK nell'euristica di Fiat-Shamir. Un altro modo per ottenere un sistema di prova non-interattivo a conoscenza nulla è quello di ipotizzare che esista un oracolo casuale (accessibile a tutti).

L'idea è quella di partire da un protocollo-Σ (cf. Definizione 13.7). Il dimostratore genera innanzitutto α in modo onesto. Quindi anziché ricevere la sfida da $\mathscr{V}$, la calcola usando l'oracolo casuale: $\beta = H(\alpha, x)$. Una volta nota la sfida β è possibile calcolare il valore γ, ottenendo così la prova $\pi = (\alpha, \beta, \gamma)$ senza interazione. Nella definizione dobbiamo richiedere che le proprietà di completezza e validità siano soddisfatte per ogni possibile scelta della funzione H. Inoltre nella proprietà di conoscenza nulla il simulatore deve essere in grado di simulare anche l'accesso all'oracolo per $\mathscr{V}^*$. Il punto chiave qui è che il simulatore ha pieno controllo sull'oracolo, nel senso che se $\mathscr{V}$ richiede il valore $H(\alpha, x)$, il simulatore può programmare l'oracolo in modo opportuno rispondendo con un valore diverso (purché questo rispetti la distribuzione uniforme).

Non daremo una dimostrazione esplicita del fatto che il sistema di prova non-interattivo così ottenuto è a conoscenza nulla. (Comunque, la dimostrazione può essere ricavata a partire dal caso delle firme digitali nell'euristica di Fiat-Shamir, che discutiamo di seguito).

Un problema dei sistemi NIZK nel modello dell'oracolo è che le prove diventano "trasferibili": se $\mathscr{V}$ ha ricevuto da $\mathscr{P}$ la prova π per un elemento $x \in \mathcal{L}$, ora $\mathscr{V}$ può convincere chiunque (con accesso allo stesso oracolo casuale) che $x \in \mathcal{L}$. Questo fatto in un certo senso contraddice il significato intuitivo di conoscenza nulla, ovvero il fatto che $\mathscr{V}$ non dovrebbe aver imparato nulla dopo aver interagito con $\mathscr{P}$ (se non che $x \in \mathcal{L}$). Si può risolvere questo problema ricorrendo ad oracoli non programmabili [YZ06].

Firme digitali nell'euristica di Fiat-Shamir. Mostriamo che un qualsiasi protocollo-Σ può essere trasformato in uno schema di firma digitale nel modello dell'oracolo casuale. L'idea è dovuta a Fiat e Shamir [FS86] (vedi anche [Abd+02; Abd+08]) ed è mostrata nel Crittosistema 13.2.[83]

[83] Più in generale il risultato di Fiat e Shamir si applica ad ogni protocollo di autenticazione a tre round con sicurezza passiva. Un protocollo-Σ rientra in questa categoria, cf. Esercizio 13.10.

Crittosistema 13.2. Euristica di Fiat-Shamir

Sia $(\mathscr{P}, \mathscr{V})$ un protocollo-Σ per un linguaggio $\mathcal{L}$ con relazione ϱ. Sia inoltre H una funzione hash con codominio $\mathcal{B} = \{0,1\}^{\ell}$. Consideriamo il seguente schema di firma digitale $\Pi_{\mathsf{FS}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$:

- `Generazione delle chiavi.` Dato il parametro di sicurezza n, l'algoritmo Gen restituisce $(x, w) \leftarrow \mathsf{Gen}(1^n)$ tali che $(\varrho(x, w) = 1$. Poni $pk = x$ ed $sk = w$.
- `Creazione delle firme.` Dato un messaggio m da firmare e la chiave segreta w, calcola α come in un'esecuzione onesta del protocollo-Σ. Poni quindi $\beta = H(m, \alpha)$ e calcola la risposta alla sfida β come in un'esecuzione onesta del protocollo-Σ. La firma è $\sigma = (\alpha, \gamma) \leftarrow \mathsf{Sign}_{sk}(m)$.
- `Verifica.` Data una coppia (m, σ) e la chiave pubblica x, calcola $\beta = H(m, \alpha)$. Quindi $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ se e solo se $\mathscr{V}(1^n, x, (\alpha, \beta, \gamma))$ ritorna `accetta`.

Teorema 13.10 (Sicurezza dell'euristica di Fiat-Shamir). *Sia $(\mathscr{P}, \mathscr{V})$ un protocollo-Σ con errore di validità ϵ_s. Assumiamo inoltre che $\ell = \omega(\log n)$ e che $\mathbf{H}_{\infty}(\alpha|x) = \omega(\log n)$. Allora lo schema di firma Π_{FS} del Crittosistema 13.2 è $(t, Q, \epsilon_{\mathsf{FS}})$-ufcma nel modello dell'oracolo casuale per*

$$t, Q = \mathrm{poly}(n) \qquad\qquad \epsilon_{\mathsf{FS}} \leq Q_H \cdot (\epsilon_s + \mathrm{negl}(n)),$$

avendo indicato con Q_H il numero di richieste all'oracolo casuale H.

Dimostrazione. Supponiamo che esista un forgiatore $\mathscr{F}$ in grado di forgiare una firma $(m^*, (\alpha^*, \gamma^*))$ per un messaggio fresco con probabilità $> Q_H(\epsilon_s + \mathrm{negl}(n))$, effettuando Q richieste all'oracolo Sign_{sk} e Q_H richieste all'oracolo casuale H. Mostreremo come costruire un algoritmo $\mathscr{P}^*$ in grado di convincere $\mathscr{V}$ (senza conoscere $sk = w$), semplicemente osservando Q esecuzioni oneste del protocollo-Σ. Supponiamo (senza perdita di generalità) che $\mathscr{F}$ invochi l'oracolo H con input (m^*, α^*) in corrispondenza della richiesta i^* (per un qualche indice $i^* \leq Q_H$). L'algoritmo $\mathscr{P}^*$ simula l'ambiente per $\mathscr{F}$ come segue:

1. Estrai $j^* \xleftarrow{\$} \{1, \ldots, Q_H\}$. Siano $(\alpha_1, \beta_1, \gamma_1), \ldots, (\alpha_Q, \beta_Q, \gamma_Q)$ le prove generate da Q esecuzioni oneste del protocollo-Σ.

2. Per ogni $j < j^*$, in corrispondenza di una richiesta $(\widehat{\alpha}_j, \widehat{m}_j)$ all'oracolo H, se $H(\widehat{\alpha}_j, \widehat{m}_j)$ è già definito restituisci tale valore, altrimenti estrai un valore a caso in $\{0,1\}^\ell$.

3. In corrispondenza della richiesta $(\widehat{\alpha}_{j^*}, \widehat{m}_{j^*})$ all'oracolo H, inoltra $\widehat{\alpha}_{j^*}$ a $\mathscr{V}$. Sia β^* la risposta di $\mathscr{V}$. Poni $H(\widehat{\alpha}_{j^*}, \widehat{m}_{j^*}) = \beta^*$.

4. Continua a rispondere alle richieste rimanenti per H come in (2).

5. In corrispondenza di una richiesta m_i (per ogni $i = 1, \ldots, Q$) all'oracolo Sign_{sk}, poni $H(\alpha_i, m_i) = \beta_i$. Se $H(\alpha_i, m_i)$ è già definito, ritorna $\perp$. Poni $\sigma_i = (\alpha_i, \gamma_i)$.

6. Quando $\mathscr{F}$ ritorna la forgiatura $(m^*, \sigma^* = (\alpha^*, \gamma^*))$, se $(\alpha^*, m^*) \neq (\widehat{\alpha}_{j^*}, \widehat{m}_{j^*})$ restituisci $\perp$. Altrimenti restituisci γ^* come risposta alla sfida β^* di $\mathscr{V}$.

Sia $\mathcal{E}$ l'evento che $\mathscr{P}^*$ fallisca al punto (5). In pratica $\mathscr{P}^*$ spera che j^* sia esattamente l'indice i^* per cui $\mathscr{F}$ restituisce una forgiatura. Quando questo accade (quindi con probabilità $1/Q_H$, a patto che $\mathcal{E}$ non si verifichi), se la firma forgiata da $\mathscr{F}$ è valida, $\mathscr{P}^*$ ha violato la validità del protocollo-Σ. Comunque, la probabilità di $\mathcal{E}$ è trascurabile in n, perché $\mathbf{H}_\infty(\alpha|x) = \omega(\log(n))$. Quindi, avendo supposto che $\mathscr{F}$ forgia una firma valida con probabilità $\epsilon_{\mathsf{FS}} > Q_H \cdot (\epsilon_s + \mathrm{negl}(n))$, segue:

$$\begin{aligned}
\mathbb{P}\left[\mathscr{P}^*(1^n, x) \leftrightarrows \mathscr{V}(1^n, x) = \texttt{accetta}\right] &= \mathbb{P}\left[\mathscr{F} \text{ forgia} \wedge \overline{\mathcal{E}} \wedge j^* = i^*\right] \\
&= (1 - \mathbb{P}\left[\mathcal{E}\right]) \cdot \mathbb{P}\left[\mathscr{F} \text{ forgia} \wedge j^* = i^* | \overline{\mathcal{E}}\right] \\
&\geq \frac{\epsilon_{\mathsf{FS}}}{Q_H} - \mathrm{negl}(n) > \epsilon_s,
\end{aligned}$$

contro l'ipotesi che il protocollo-Σ abbia validità ϵ_s. □

Nella pratica la funzione H è rimpiazzata, ad esempio, da SHA-1. Un risultato negativo [GK03] mostra che esistono schemi di identificazione a tre round con sicurezza passiva tali che l'euristica di Fiat-Shamir genera firme non sicure per *ogni possibile* implementazione della funzione H. Tali esempi sono comunque artificiali e non inficiano la validità dell'euristica di Fiat-Shamir.

Esercizi

Esercizio 13.1. Mostrare che la ripetizione parallela (per r volte) del protocollo di Fig. 13.1 mantiene la proprietà di HVZK.

Esercizio 13.2. Dimostrare che se esiste un sistema di prova interattivo per un linguaggio $\mathcal{L}$ con errore di validità $1/3$ (ed errore di completezza nullo), allora esiste un sistema di prova interattivo per $\mathcal{L}$ con errore di validità 2^{-r} (per una costante $r \in \mathbb{N}$).
(Suggerimento: considerare r ripetizioni in parallelo del sistema di prova.)

Esercizio 13.3. Mostrare che ogni protocollo-Σ con spazio delle sfide $\mathcal{B} = \{0,1\}^{\ell}$, è un sistema di prova interattivo con errore di validità $2^{-\ell}$.

Esercizio 13.4. Mostrare che tutte le proprietà dei protocolli-Σ sono invarianti per composizione parallela. *(Suggerimento: considerare il caso di due protocolli-Σ per linguaggio $\mathcal{L}$ che usano la stessa sfida $\beta \in \{0,1\}^{\ell}$ e mostrare che ciò è equivalente ad un unico protocollo-Σ con spazio delle sfide $\mathcal{B} = \{0,1\}^{2\ell}$. Chiarire perché usare la stessa sfida non compromette la sicurezza.)*

Esercizio 13.5. Mostrare che se esiste un protocollo-Σ per un linguaggio $\mathcal{L}$, allora, per ogni $\ell' > 0$, esiste un protocollo-Σ per $\mathcal{L}$ con spazio delle sfide $\mathcal{B} = \{0,1\}^{\ell'}$.

Esercizio 13.6. In questo esercizio mostriamo che ogni protocollo-Σ può essere trasformato in un altro protocollo-Σ che soddisfa la proprietà di S-HVZK speciale. Sia $(\mathscr{P}, \mathscr{V})$ un protocollo-Σ per un linguaggio $\mathcal{L}$ con relazione ϱ. Consideriamo il seguente protocollo $(\mathscr{P}', \mathscr{V}')$ per $\mathcal{L}$.

> $\mathscr{P}'$ inizia lanciando $\mathscr{P}$ per ottenere α, quindi sceglie β' a caso ed invia (α, β') a $\mathscr{V}'$. Il verificatore sceglie una sfida casuale $\beta'' \xleftarrow{\$} \{0,1\}^{\ell}$. Quindi $\mathscr{P}'$ calcola $\beta = \beta' \oplus \beta''$ ed usa $\mathscr{P}$ per determinare la risposta γ relativa a β. Invia quindi γ a $\mathscr{V}'$. Il verificatore accetta se e solo se $(x, (\alpha, \beta' \oplus \beta'', \gamma))$ è una prova accettata da $\mathscr{V}$.

Mostrare che $(\mathscr{P}', \mathscr{V}')$ è un protocollo-Σ che soddisfa la proprietà S-HVZK.

Esercizio 13.7. Sia $\mathbb{G}$ un gruppo con ordine un primo q. Consideriamo il seguente protocollo $(\mathscr{P}, \mathscr{V})$ per dimostrare che $x = (g_1, g_2, g_3, g_4)$ è di tipo DH, ovvero esiste $w \in \mathbb{Z}_q$ tale che $g_3 = g_1^w$ e $g_4 = g_2^w$.

Il dimostratore sceglie $\rho \xleftarrow{\$} \mathbb{Z}_q$ ed inoltra $\alpha = (g_3^\rho, g_4^\rho)$. Il verificatore ritorna la sfida $\beta \xleftarrow{\$} \mathbb{Z}_q$. Quindi $\mathscr{P}$ risponde con $\gamma = \rho + \beta \cdot w$ e $\mathscr{V}$ controlla che $g_1^\gamma = \alpha_1 \cdot g_3^\beta$ e $g_2^\gamma = \alpha_2 \cdot g_4^\beta$.

1. Dimostrare che $(\mathscr{P}, \mathscr{V})$ è un protocollo-Σ per il linguaggio:
$$\mathcal{L} = \{x = (g_1, g_2, g_3, g_4) : \exists w \text{ tale che } g_3 = g_1^w, g_4 = g_2^w\}.$$
2. Consideriamo il cifrario di ElGamal con chiave pubblica $pk = (\mathbb{G}, g, q, h = g^w)$ e chiave segreta $sk = w$. Sia $c = (c_0, c_1) = (g^\rho, h^\rho \cdot m)$ una cifratura di m. Usare il protocollo del punto precedente in modo che $\mathscr{P}$ possa dimostrare di conoscere m, ovvero ricavare un protocollo-Σ per il linguaggio:
$$\mathcal{L} = \{(c_0, c_1, pk) : \exists(\rho, m) \text{ tali che } c_0 = g^\rho, c_1 = h^\rho \cdot m\}.$$

Esercizio 13.8. Sia $\Sigma = (\mathscr{P}, \mathscr{V})$ un protocollo-Σ per un linguaggio $\mathcal{L}$ e supponiamo che Σ soddisfi la proprietà di S-HVZK. Dati $x_0, x_1 \in \mathcal{L}$, questo esercizio mostra come sia possibile costruire un protocollo-Σ per dimostrare che si conosce un indizio relativamente ad x_0 *oppure* ad x_1. Più precisamente, sia $\mathcal{L}$ un linguaggio con relazione ϱ e siano $x_0, x_1 \in \mathcal{L}$. Consideriamo il seguente protocollo $\Sigma' = (\mathscr{P}', \mathscr{V}')$ per il linguaggio:

$$\mathcal{L}_{OR} = \{(x_0, x_1) : \exists w \text{ tale che } \varrho(x_0, w) = 1 \text{ oppure } \varrho(x_1, w) = 1\}.$$

Supponiamo che $\mathscr{P}'$ possegga un indizio w_0 per x_0, ma non per x_1. Inizialmente il dimostratore sceglie una sfida a caso $\beta_1 \xleftarrow{\$} \mathcal{B}$ e lancia il simulatore $\mathscr{Z}$ del protocollo Σ per ottenere una prova simulata $\pi_1 = (\alpha_1, \beta_1, \gamma_1) \leftarrow \mathscr{Z}(\beta_1, x_1)$. A questo punto, $\mathscr{P}'$ invia $\alpha = (\alpha_0, \alpha_1)$, in cui α_0 è generato lanciando l'algoritmo $\mathscr{P}$ di Σ. Sia $\beta \xleftarrow{\$} \mathcal{B}$ la sfida scelta da $\mathscr{V}'$. Il dimostratore calcola $\beta_0 = \beta \oplus \beta_1$, quindi risponde con $\gamma = (\gamma_0, \gamma_1)$, dove γ_0 è calcolato invocando $\mathscr{P}$ ed utilizzando l'indizio w_0. Il verificatore $\mathscr{V}'$, infine, accetta se e solo se $\beta = \beta_0 \oplus \beta_1$ e se $\mathscr{V}(x_1, \pi_1) = \mathscr{V}(x_0, w_0) =$ `accetta`.

1. Mostrare che Σ' è un protocollo-Σ per il linguaggio $\mathcal{L}_{OR}$.
2. Suggerire un modo di utilizzare Σ' per dimostrare che si conosce l'indizio relativo ad *uno* tra $r \geq 2$ elementi di $\mathcal{L}$. Dire se lo stesso approccio funziona quando si vuole dimostrare che si conosce l'indizio relativamente a $2 \leq r' < r$ elementi?

Esercizio 13.9. Abbiamo visto che i protocolli-Σ non sono a conoscenza nulla. In questo esercizio, mostriamo che il protocollo Σ' dell'esercizio precedente, tuttavia, offre qualche garanzia contro verificatori disonesti. Diciamo che una relazione ϱ per un linguaggio $\mathcal{L}$ è difficile, se si verifica quanto segue: (i) esiste un algoritmo PPT Gen che, dato in input il parametro di sicurezza n, restituisce $(x, w) \leftarrow \mathsf{Gen}(1^n)$ tali che $\varrho(x, w) = 1$ ed $|x| = n$, (ii) ciascun attaccante $\mathscr{A}$ che riceve x come input è in grado di produrre w^* tale che $\varrho(x, w^*) = 1$ solamente con probabilità trascurabile. Un protocollo-Σ per una relazione ϱ *nasconde l'indizio* se la probabilità di successo di un verificatore disonesto nel seguente esperimento è trascurabile:

> $\mathscr{P}$ riceve come input w, dove $(x, w) \leftarrow \mathsf{Gen}(1^n)$. Quindi $\mathscr{V}^*$ può eseguire $\mathscr{P}$ un numero polinomiale di volte con input comune x. Diremo che $\mathscr{V}^*$ ha successo se è in grado di produrre w^* tale che $\varrho(x, w^*) = 1$.

Mostrare che quando la relazione ϱ è difficile, il protocollo Σ' per il linguaggio $\mathcal{L}_{OR}$ dell'esercizio precedente nasconde l'indizio.

Esercizio 13.10. Mostrare che ogni protocollo Σ è anche un protocollo di autenticazione con sicurezza passiva.

Esercizio 13.11. Sia $N = p \cdot q$. Dati $w_1, \ldots, w_r \xleftarrow{\$} \mathbb{Z}_N^*$, poniamo $h_i = w_i^2 \bmod N$ per ogni $i = 1, \ldots, r$. Consideriamo il seguente protocollo di autenticazione con chiave pubblica $\{h_i\}$ e chiave segreta $\{w_i\}$.

> $\mathscr{P}$ sceglie un valore casuale $\rho \xleftarrow{\$} \mathbb{Z}_N^*$ ed invia $\alpha = \rho^2 \bmod N$. Il verificatore sceglie una sfida casuale $\beta = \beta_1 \ldots \beta_r \xleftarrow{\$} \{0,1\}^r$. Quindi $\mathscr{P}$ risponde con $\gamma = \prod_{i=1}^r \rho \cdot w_i^{\beta_i} \bmod N$. Il verificatore accetta se e solo se $\gamma^2 = \alpha \cdot \prod_{i=1}^r h_i^{\beta_i}$.

Rispondere alle seguenti domande.

1. Mostrare che il protocollo è un protocollo-Σ e specificare per quale linguaggio.
2. Mostrare che il protocollo di autenticazione è sicuro contro attaccanti passivi e specificare sotto quale ipotesi ciò si verifica.
3. Derivare una firma digitale usando l'euristica di Fiat-Shamir.

Esercizio 13.12. Dire come applicare l'euristica di Fiat-Shamir ad uno schema di autenticazione a 5 messaggi per ottenere una firma digitale. Mostrare che la firma ottenuta è sicura.

Letture consigliate

[Abd+02] Michel Abdalla, Jee Hea An, Mihir Bellare e Chanathip Namprempre. "From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security". In: *EUROCRYPT*. 2002, pp. 418–433.

[Abd+08] Michel Abdalla, Jee Hea An, Mihir Bellare e Chanathip Namprempre. "From Identification to Signatures Via the Fiat-Shamir Transform: Necessary and Sufficient Conditions for Security and Forward-Security". In: *IEEE Transactions on Information Theory* 54.8 (2008), pp. 3631–3646.

[Bar01] Boaz Barak. "How to Go Beyond the Black-Box Simulation Barrier". In: *FOCS*. 2001, pp. 106–115.

[BCC88] Gilles Brassard, David Chaum e Claude Crépeau. "Minimum Disclosure Proofs of Knowledge". In: *J. Comput. Syst. Sci.* 37.2 (1988), pp. 156–189.

[BG92] Mihir Bellare e Oded Goldreich. "On Defining Proofs of Knowledge". In: *CRYPTO*. 1992, pp. 390–420.

[Can+01] Ran Canetti, Joe Kilian, Erez Petrank e Alon Rosen. "Black-box concurrent zero-knowledge requires $\widetilde{\Omega}(\log n)$ rounds". In: *STOC*. 2001, pp. 570–579.

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *J. Cryptology* 13.1 (2000), pp. 143–202.

[Car71] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1871.

[CHK10] Ronald Cramer, Dennis Hofheinz e Eike Kiltz. "A Twist on the Naor-Yung Paradigm and Its Application to Efficient CCA-Secure Encryption from Hard Search Problems". In: *TCC*. 2010, pp. 146–164.

[Cra96] Ronald Cramer. "Modular Design of Secure yet Practical Cryptographic Protocol". Tesi di dott. CWI e University of Amsterdam, 1996.

[DDN91] Danny Dolev, Cynthia Dwork e Moni Naor. "Non-Malleable Cryptography (Extended Abstract)". In: *STOC*. 1991, pp. 542–552.

[DNS98] Cynthia Dwork, Moni Naor e Amit Sahai. "Concurrent Zero-Knowledge". In: *STOC*. 1998, pp. 409–418.

[Fei90] Uriel Feige. "Alternative Models for Zero-Knowledge Interactive Proofs". Disponibile su `http://www.wisdom.weizmann.ac.il/~feige`. Tesi di dott. Dept. of Computer Science e Applied Mathematics, Weizmann Institute of Science, 1990.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *CRYPTO*. 2005, pp. 152–168.

[FLS90] Uriel Feige, Dror Lapidot e Adi Shamir. “Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract)”. In: *FOCS*. 1990, pp. 308–317.

[FLS99] Uriel Feige, Dror Lapidot e Adi Shamir. “Multiple NonInteractive Zero Knowledge Proofs Under General Assumptions”. In: *SIAM J. Comput.* 29.1 (1999), pp. 1–28.

[FS86] Amos Fiat e Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO*. 1986, pp. 186–194.

[FS90] Uriel Feige e Adi Shamir. “Witness Indistinguishable and Witness Hiding Protocols”. In: *STOC*. 1990, pp. 416–426.

[GK03] Shafi Goldwasser e Yael Tauman Kalai. “On the (In)security of the Fiat-Shamir Paradigm”. In: *FOCS*. 2003, pp. 102–.

[GK96a] Oded Goldreich e Ariel Kahan. “How to Construct Constant-Round Zero-Knowledge Proof Systems for NP”. In: *J. Cryptology* 9.3 (1996), pp. 167–190.

[GK96b] Oded Goldreich e Hugo Krawczyk. “On the Composition of Zero-Knowledge Proof Systems”. In: *SIAM J. Comput.* 25.1 (1996), pp. 169–192.

[GMR85] Shafi Goldwasser, Silvio Micali e Charles Rackoff. “The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)”. In: *STOC*. 1985, pp. 291–304.

[GMR89] Shafi Goldwasser, Silvio Micali e Charles Rackoff. “The Knowledge Complexity of Interactive Proof Systems”. In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208.

[GMW91] Oded Goldreich, Silvio Micali e Avi Wigderson. “Proofs that Yield Nothing But Their Validity for All Languages in NP Have Zero-Knowledge Proof Systems”. In: *J. ACM* 38.3 (1991), pp. 691–729.

[GO94] Oded Goldreich e Yair Oren. “Definitions and Properties of Zero-Knowledge Proof Systems”. In: *J. Cryptology* 7.1 (1994), pp. 1–32.

[Gol01] Oded Goldreich. *Foundations of Cryptography, Vol. 1: Basic Tools*. Cambridge University Press, 2001.

[Gol02a] Oded Goldreich. “Concurrent zero-knowledge with timing, revisited”. In: *STOC*. 2002, pp. 332–340.

[Gol02b] Oded Goldreich. “Zero-Knowledge twenty years after its invention”. In: *Electronic Colloquium on Computational Complexity (ECCC)* 063 (2002).

[Gol06] Oded Goldreich. “Concurrent Zero-Knowledge with Timing, Revisited”. In: *Essays in Memory of Shimon Even*. 2006, pp. 27–87.

[GQ88] Louis C. Guillou e Jean-Jacques Quisquater. "A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory". In: *EUROCRYPT*. 1988, pp. 123–128.

[KP01] Joe Kilian e Erez Petrank. "Concurrent and resettable zero-knowledge in poly-loalgorithm rounds". In: *STOC*. 2001, pp. 560–569.

[Lin03] Yehuda Lindell. "A Simpler Construction of CCA2-Secure Public-Key Encryption under General Assumptions". In: *EUROCRYPT*. 2003, pp. 241–254.

[LS90] Dror Lapidot e Adi Shamir. "Publicly Verifiable Non-Interactive Zero-Knowledge Proofs". In: *CRYPTO*. 1990, pp. 353–365.

[NY90] Moni Naor e Moti Yung. "Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks". In: *STOC*. 1990, pp. 427–437.

[Oka92] Tatsuaki Okamoto. "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes". In: *CRYPTO*. 1992, pp. 31–53.

[PRS02] Manoj Prabhakaran, Alon Rosen e Amit Sahai. "Concurrent Zero Knowledge with Logarithmic Round-Complexity". In: *FOCS*. 2002, pp. 366–375.

[Qui+89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou e Thomas A. Berson. "How to Explain Zero-Knowledge Protocols to Your Children". In: *CRYPTO*. 1989, pp. 628–631.

[RK99] Ransom Richardson e Joe Kilian. "On the Concurrent Composition of Zero-Knowledge Proofs". In: *EUROCRYPT*. 1999, pp. 415–431.

[Sah99] Amit Sahai. "Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security". In: *FOCS*. 1999, pp. 543–553.

[Sch89] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO*. 1989, pp. 239–252.

[YZ06] Moti Yung e Yunlei Zhao. "Interactive Zero-Knowledge with Restricted Random Oracles". In: *TCC*. 2006, pp. 21–40.

CAPITOLO

14

Computazione a parti multiple

«Per favore, vorreste dirmi...» cominciò guardando timorosamente la Regina Rossa. «Parla quando ti si rivolge la parola!» la interruppe bruscamente la Regina. «Ma se tutti si attenessero a codesta regola» disse Alice, che era sempre pronta a controbattere, «e se voi parlaste quando vi si rivolge la parola e basta e le altre persone aspettassero invece che cominciaste voi a parlare, capite che nessuno direbbe mai niente, cosicché...»

Lewis Carroll, Attraverso lo Specchio e quel che Alice vi trovò [Car71]

La computazione a parti multiple (*MultiParty Computation*, MPC) sicura può essere definita come il problema in cui un insieme di giocatori $\mathcal{P} = \{P_1, \ldots, P_m\}$ vogliono calcolare una certa funzione dei loro input segreti in modo sicuro, dove per "sicuro" intendiamo che: (i) l'output della funzione deve essere corretto e (ii) la segretezza degli input deve essere preservata anche in uno scenario in cui una parte dei giocatori è corrotta. Più precisamente, ciascun giocatore possiede un input x_i e si vuole calcolare $f(x_1, \ldots, x_m) = (y_1, \ldots, y_m)$ in modo che il giocatore i-simo impari y_i e nient'altro. Se la funzione f è randomizzata, assumeremo che la stringa $\omega \in \Omega$ sia uniformemente casuale nello spazio di randomicità Ω (e che essa sia nota a tutti i giocatori).

La questione è stata posta per la prima volta da Yao [Yao82], che ha considerato il seguente "problema dei milionari". Due milionari vogliono sapere chi dei due è più ricco, senza rivelare all'altro quanti soldi possiedono. In questo caso il patrimonio di ciascun giocatore costituisce il suo input segreto, e la funzione f è definita come:

$$f(x_1, x_2) = x_1 <_? x_2 = \begin{cases} 1 & \text{se } x_1 < x_2 \\ 0 & \text{altrimenti.} \end{cases}$$

Se il risultato fosse che il primo milionario è più ricco del secondo, questa dovrebbe essere l'unica informazione che il secondo milionario impara dopo aver eseguito il protocollo per il calcolo di f.

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_14, © Springer-Verlag Italia 2012

Un esempio di maggiore rilevanza pratica è quello degli schemi di *voto elettronico*. In questo caso l'input di ciascun giocatore è un valore intero corrispondente all'identificativo del candidato votato. L'output della funzione è il candidato che ha ricevuto il più alto numero di voti e questo deve essere l'*unico* valore ad essere reso pubblico. Se ci sono solamente due candidati, ciascun giocatore possiede un input $x_i \in \{0,1\}$ dove il valore 0 esprime una preferenza per il primo candidato, mentre il valore 1 esprime una preferenza per il secondo. Quindi la funzione f in questo caso è:

$$f(x_1,\ldots,x_m) = \sum_{i=1}^{m} x_i <_? \frac{m}{2},$$

ed il valore dell'output deve essere l'unica informazione restituita dal protocollo.

Un altro esempio è quello delle *aste digitali*. In questo caso, l'input di ciascun giocatore corrisponde all'offerta per aggiudicarsi l'asta e l'output è del tipo $f(x_1,\ldots,x_m) = (x_j, j)$, dove $1 \leq j \leq m$ è l'indice per cui $x_j = \max_{i=1,\ldots,m} x_i$. (Qualora ci fossero diversi giocatori con l'offerta massima si può scegliere un giocatore a caso tra essi, attraverso una funzione randomizzata.)

Problemi di questo tipo sono l'oggetto principale di questo capitolo. Come vedremo esistono protocolli generici per calcolare in modo sicuro una funzione arbitraria (purché calcolabile in tempo polinomiale). Tuttavia questi schemi hanno per lo più valenza teorica, in quanto altamente inefficienti. D'altra parte, esistono protocolli efficienti per calcolare alcune classi di funzione specifiche (ad esempio nel caso del voto elettronico). Non entreremo nel dettaglio delle tecniche generali per la MPC sicura, per le quali si rimanda alla letteratura. Ottime panoramiche si trovano in [Gol97; Gol02; CDN09].

Guida per il lettore. La struttura del capitolo è la seguente. Nei primi tre paragrafi definiremo alcune primitive fondamentali che costituiscono i "mattoncini" base nel contesto della MPC: la *condivisione di segreti* (nel Paragrafo 14.1), il *trasferimento immemore* (nel Paragrafo 14.2) e gli *schemi di impegno* (nel Paragrafo 14.3). Nel Paragrafo 14.4 daremo quindi una panoramica dei risultati principali nell'area della MPC. Infine, nel Paragrafo 14.5, considereremo più da vicino il caso del voto elettronico.

14.1 Condivisione di segreti

Supponiamo di dover conservare una chiave crittografica in una locazione non sicura. Per evitare che persone non autorizzate accedano alla chiave dobbiamo

cifrarla, così che ora abbiamo il problema di proteggere una nuova chiave (la chiave di cifratura). Sembra impossibile dunque evitare di avere alcune chiavi nel sistema che sono considerate sicure soltanto perché memorizzate in una postazione "fisicamente sicura". A pensarci bene questa è esattamente l'assunzione che si fa nelle infrastrutture a chiave pubblica (cf. Paragrafo 10.1): la chiave segreta dell'autorità di certificazione principale deve essere protetta in modo adeguato; allo stesso tempo tale chiave deve essere disponibile in ogni istante.

Uno schema di condivisione di segreti a soglia (*threshold secret sharing*) consente di realizzare entrambi questi requisiti in apparenza contrastanti. Un tale schema prende come input un segreto s in qualche insieme $\mathcal{S}$ e restituisce m stringhe binarie $s_1, \ldots, s_m$ dette *porzioni* di s. Lo schema dipende ulteriormente da un parametro t detto soglia: richiederemo che quando sono note al più t porzioni di s, ciò non rivela nulla su s; d'altra parte la conoscenza di $t+1$ (o più) porzioni è sempre sufficiente a ricostruire s completamente.

Definizione 14.1 (Condivisione di segreti a soglia). Uno schema di condivisione di segreti a soglia di tipo-(m, t) è un algoritmo randomizzato che prende in ingresso un segreto $s \in \mathcal{S}$ e restituisce m porzioni $s_1, \ldots, s_m$ tali che:

- *Segretezza.* Per ogni sottoinsieme $\Sigma \subset \{1, \ldots, m\}$ di dimensione al più t, la distribuzione di probabilità di $\{s_i : i \in \Sigma\}$ è indipendente da s.
- *Correttezza.* Per ogni sottoinsieme $\Gamma \subset \{1, \ldots, m\}$ di dimensione almeno $t+1$, il valore s è determinato in maniera univoca dagli elementi $\{s_i : i \in \Gamma\}$, ovvero esiste un algoritmo polinomiale per ricostruire s dati i valori $\{s_i : i \in \Gamma\}$. ■

Questa primitiva è stata introdotta per la prima volta da Shamir [Sha79]. Notare che uno schema di questo tipo mitiga il problema che abbiamo discusso all'inizio del paragrafo: basta calcolare le porzioni relative alla chiave da proteggere. In questo modo un avversario ha la vita più difficile, in quanto ora deve recuperare almeno $t+1$ porzioni per risalire alla chiave e, allo stesso tempo, il sistema è robusto nel caso di perdita di al più $m-t-1$ porzioni.

Interpolazione di Lagrange. L'idea di Shamir è basata sull'interpolazione di Lagrange.

Lemma 14.1 (Interpolazione di Lagrange). *Sia* $\mathbb{F}$ *un campo. Per ogni insieme di coppie* $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1}) \in \mathbb{F} \times \mathbb{F}$*, dove gli elementi* x_i *sono*

tutti distinti, esiste un unico polinomio g di grado al più t tale che:

$$g(x_i) = y_i \qquad \forall\ i = 1, \ldots, t+1.$$

I coefficienti del polinomio g possono essere calcolati in modo efficiente a partire da $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$.

Dimostrazione. Basta osservare che il polinomio

$$g_i(X) = \frac{(x_1 - X)(x_2 - X)\cdots(x_{i-1} - X)(x_{i+1} - X)\cdots(x_{t+1} - X)}{(x_1 - x_i)(x_2 - x_i)\cdots(x_{i-1} - x_i)(x_{i+1} - x_i)\cdots(x_{t+1} - x_i)},$$

ha grado t ed inoltre soddisfa $g_i(x_i) = 1$ e $g_i(x_j) = 0$ (per ogni $i \neq j$). Pertanto

$$g(X) = y_1 g_1(X) + \ldots + y_{t+1} g_{t+1}(X),$$

è il polinomio cercato.

Notare che g può essere calcolato in modo efficiente. Inoltre la soluzione è unica, in quanto se esistessero due soluzioni g, g', il polinomio $g(X) - g'(X)$ sarebbe un polinomio non nullo di grado al più t con $t+1$ radici, che non può esistere.[84] □

Supponiamo ora di porre $\mathcal{S} = \mathbb{Z}_p$, per un qualche primo $p > m$, e consideriamo lo schema seguente. Si estraggono dapprima i coefficienti $a_1, \ldots, a_t \xleftarrow{\$} \mathbb{Z}_p$ e si pone $f(X) = s + a_1 X + a_2 X^2 + \ldots + a_t X^t$. (In altri termini f è un polinomio casuale di grado al più t, con coefficienti in $\mathbb{Z}_p$, tale che $f(0) = s$.) Le porzioni di s sono definite da $s_i = f(i) \bmod p$, per ogni $i = 1, 2, \ldots, m$.

Teorema 14.2 (Condivisione di segreti di Shamir). *Sia p un primo ed $m, t \in \mathbb{N}$ interi tali che $t < m < p$. Lo schema di Shamir è uno schema di condivisione di segreti a soglia (per $\mathcal{S} = \mathbb{Z}_p$) di tipo-$(m, t)$.*

Dimostrazione. Il Lemma 14.1 implica immediatamente che lo schema di Shamir soddisfa la proprietà di correttezza. Sia infatti Γ un qualsiasi sottoinsieme di $\{1, \ldots, m\}$ con $t+1$ elementi; i giocatori in $\mathcal{P}$ corrispondenti agli indici in Γ possiedono coppie $(i, s_i)_{i \in \Gamma}$. Tali giocatori possono ricavare il polinomio:

$$g(X) = \sum_{i \in \Gamma} s_i \prod_{\substack{j \in \Gamma \\ j \neq i}} \frac{j - X}{j - i}.$$

[84] Per il Teorema fondamentale dell'algebra (enunciato per la prima volta nel 1629 da Albert Girand e dimostrato da Gauss solo nel 1799), ogni polinomio (non nullo) ad una sola incognita e con coefficienti complessi, ha esattamente tante radici quanto il suo grado (contando le singole radici con la rispettiva molteplicità).

Notare che $g(i) = s_i = f(i)$, ovvero i polinomi f e g sono due polinomi di grado al più t che condividono $t+1$ punti, quindi per il Lemma 14.1 deve essere $f \equiv g$. In particolare $g(0) = s = f(0)$ e lo schema soddisfa la proprietà di correttezza. Osserviamo che il segreto può essere ricostruito come combinazione lineare delle porzioni $\{s_i\}_{i\in\Gamma}$:

$$s = g(0) = \sum_{i\in\Gamma} \alpha_i s_i \qquad \text{con } \alpha_i = \prod_{\substack{j\in\Gamma \\ j\neq i}} \frac{j}{j-i}. \tag{14.1}$$

I coefficienti $\{\alpha_i\}_{i\in\Gamma}$ della combinazione lineare dipendono solo da Γ e non dal segreto s.

Resta da mostrare che ogni insieme costituito da t porzioni non dà alcuna informazione su s. Per vedere ciò, fissiamo un insieme Σ di dimensione t ed un segreto s. Osserviamo che ogni polinomio f di grado al più t e con $f(0) = s$, definisce un insieme di porzioni $\{f(i) : i \in \Sigma\}$. Quindi, per il Lemma 14.1, ogni insieme potenziale di porzioni $A = \{s_i : i \in \Sigma\}$ corrisponde ad un unico polinomio f_A, di grado al più t, tale che $f_A(0) = s$ e $f_A(i) = s_i$ per $i \in \Gamma$. Siccome ci sono p^t polinomi di grado al più t che soddisfano $f(0) = s$, e poiché ci sono p^t insiemi A di porzioni, ci deve essere una corrispondenza uno-a-uno tra i due insiemi. Dato che nello schema di Shamir il polinomio f è scelto a caso tra i polinomi di grado t che soddisfano $f(0) = s$, ogni insieme A di porzioni è ugualmente probabile. Segue che per ogni scelta di Σ e di $s \in \mathcal{S}$, il requisito di segretezza è soddisfatto. □

Lo schema di Shamir è molto efficiente; inoltre le porzioni hanno la stessa dimensione del segreto s. Ciò è in effetti ottimo: per ogni distribuzione di probabilità con cui s è scelto e per ogni schema di condivisione di segreti di tipo-(t, m), l'entropia di ogni porzione deve essere almeno pari all'entropia di s. Per vedere ciò, supponiamo che l'entropia di s sia ℓ bit e consideriamo la prima porzione s_1 (questo è senza perdita di generalità, lo stesso ragionamento si applica ad ogni porzione). Consideriamo un qualsiasi insieme A di porzioni tale che A da solo è insufficiente a costruire s, ma $A \cup \{s_1\}$ consente di recuperare s. Dalla proprietà di segretezza segue che la conoscenza di A porta informazione nulla su s. D'altra parte la conoscenza di s_1 consente di recuperare s, pertanto s_1 deve avere almeno ℓ bit d'entropia.

Condivisione di segreti verificabile. Nello schema di Shamir non è possibile verificare se una porzione ricevuta è corretta oppure no. Per affrontare

tale problema sono stati proposti schemi di condivisione di segreti in cui ciascun giocatore può verificare la correttezza della porzione che ottiene [Fel87; Ped91]. Nello schema di Feldman, ad esempio, si sfrutta un gruppo $\mathbb{G}$ di ordine un primo q con generatore g. L'idea è che quando le porzioni s_i sono generate nel protocollo di Shamir, si generano anche i valori di verifica $\sigma_0 = g^s$ ed in generale $\sigma_i = g^{a_i}$. In questo modo ciascun giocatore può calcolare:

$$g^{f(i)} = \prod_{j=1}^{t} \sigma_j^{(i^j)},$$

e quindi verificare se la porzione s_i che riceve soddisfa $g^{f(i)} = g^{s_i}$. Si può mostrare che lo schema è sicuro, a patto che sia difficile calcolare logaritmi discreti in $\mathbb{G}$ [Fel87].

Strutture d'accesso generalizzate. L'insieme Γ prende il nome di *struttura d'accesso*. Possiamo considerare strutture d'accesso diverse da quella a soglia. Ad esempio si potrebbe voler richiedere che solo determinati sottoinsiemi dell'insieme di giocatori $\mathcal{P} = \{P_1, \ldots, P_m\}$, sia in grado di ricostruire s. Si definiscono *insiemi qualificati* gli insiemi che consentono di ricostruire il segreto. D'altra parte gli altri insiemi, cosiddetti *insiemi ignoranti*, non ottengono alcuna informazione su s.

La *struttura d'accesso* Γ, comprende di fatto tutti gli insiemi qualificati. La collezione degli insiemi ignoranti è indicata invece con Σ e tipicamente si ha $\Sigma = 2^{\mathcal{P}} \setminus \Gamma$. Notare che è possibile classificare uno schema di condivisione dei segreti anche in base alla *struttura di segretezza*, che colleziona gli insiemi ignoranti. Questa scelta è più naturale nel contesto in cui alcuni giocatori possono barare durante l'esecuzione del protocollo — ad esempio non inviando un messaggio, oppure inviando un valore sballato — in quanto in questo caso non è più garantito che gli insiemi qualificati siano in grado di ricostruire il segreto (mentre gli insiemi ignoranti restano ignoranti).

Alcuni schemi di condivisione di segreti per strutture d'accesso generalizzate sono stati costruiti in [Bri89; ISN93; KW93; BI92]. Il problema di questi schemi è che la dimensione delle porzioni è esponenziale nel numero dei giocatori nella struttura di accesso, il che rende queste soluzioni non pratiche. D'altra parte, almeno in linea teorica, devono esistere schemi molto più efficienti: ad esempio Csirmaz [Csi94] ha dimostrato che per ogni m esiste una struttura di accesso su m giocatori tale che è possibile condividere un segreto di n bit con porzioni di lunghezza $\Omega(n \cdot m / \log m)$. Si veda [Bei11] per una panoramica più approfondita di questi risultati.

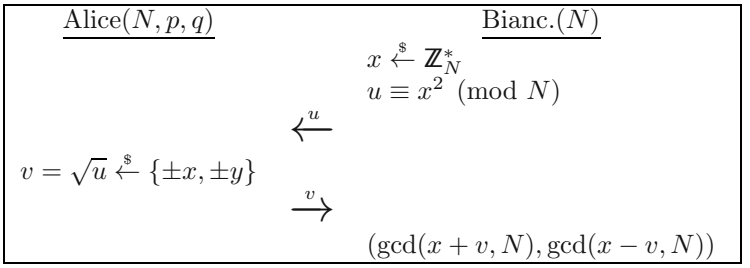


Fig. 14.1. Il protocollo di Rabin per il trasferimento immemore di (p, q)

14.2 Trasferimento immemore

Il trasferimento immemore (*Oblivious Transfer*, OT) è un protocollo inventato da Rabin [Rab81] con proprietà (apparentemente) strane, che si è poi rivelato fondamentale per la realizzazione della MPC sicura [Kil88]. La primitiva originale definita da Rabin consente ad Alice di trasmettere un dato input al Bianconiglio, in modo che quest'ultimo riceva l'input solo con probabilità 1/2 (metaforicamente, Alice non ricorda se il Bianconiglio ha ricevuto il messaggio oppure no). In questa forma la primitiva è indicata con $\frac{1}{2}$-OT.

Lo schema considerato da Rabin è mostrato in Fig. 14.1. Alice sceglie un intero di Blum $N = p \cdot q$ (ovvero $p, q \equiv 3 \pmod 4$, cf. Appendice B.3) e lo rivela al Bianconiglio. Il Bianconiglio sceglie un valore a caso $x \xleftarrow{\$} \mathbb{Z}_N^*$ ed invia $u = x^2 \bmod N$ ad Alice. Quest'ultima calcola la radice $\sqrt{v}$ (ciò può essere fatto efficientemente perché Alice conosce p e q), ottenendo quattro valori $\{\pm x, \pm y\}$ (cf. Lemma B.23). Sceglie quindi v a caso in $\{\pm x, \pm y\}$ e lo invia al Bianconiglio. Ovviamente, se $v = \pm x$ il Bianconiglio non ha imparato nulla, ma quando $v = \pm y$ quest'ultimo può calcolare:

$$p = \gcd(x + v, N) \qquad q = \gcd(x - v, N).$$

Pertanto Alice trasmette al Bianconiglio la coppia (p, q) con probabilità 1/2, ma non sa se effettivamente quest'ultimo ha imparato (p, q) oppure no. [85]

[85] Lo schema non è sicuro se il Bianconiglio non è onesto durante l'esecuzione del protocollo: si può dimostrare che alcuni valori di u consentono di fattorizzare N facilmente, a partire dalla risposta v inviata da Alice.

Una definizione alternativa. Esiste una definizione alternativa dovuta ad Even, Goldreich e Lampel [EGL85], che indicheremo nel seguito con $\binom{2}{1}$-OT. In questo caso, Alice possiede due input segreti (x_0, x_1) e si richiede che, al termine del protocollo, il Bianconiglio impari x_θ (per un valore $\theta \in \{0,1\}$ a sua scelta) senza che Alice sappia quale.[86] In [Cré87] si mostra che le due definizioni sono in realtà equivalenti; ci occuperemo solamente del caso passivo (in cui Alice ed il Bianconiglio sono entrambi onesti ma curiosi).

Teorema 14.3 ($\binom{2}{1}$-OT $\Leftrightarrow$ $\frac{1}{2}$-OT). *Le definizioni di* $\binom{2}{1}$-OT *e di* $\frac{1}{2}$-OT *sono equivalenti.*

Dimostrazione (bozza). ($\Rightarrow$) Dato un protocollo che realizza $\binom{2}{1}$-OT costruiamo un protocollo che realizza $\frac{1}{2}$-OT. Supponiamo che Alice conosca un bit $b \in \{0,1\}$ da trasmettere al Bianconiglio. Alice agisce come segue.

1. Estrae due bit casuali $b', b'' \xleftarrow{\$} \{0,1\}$, quindi lancia il protocollo $\binom{2}{1}$-OT con input due bit (x_0, x_1) tali che:

$$x_{b''} = b \qquad x_{1-b''} = b'.$$

   (Notare che se $b'' = 0$ l'input è $(x_0, x_1) = (b, b')$, mentre se $b'' = 1$ l'input è $(x_0, x_1) = (b', b)$.)

2. Il Bianconiglio sceglie il valore θ relativo all'input che vuole imparare. Al termine del protocollo $\binom{2}{1}$-OT (quindi dopo che il Bianconiglio ha imparato x_θ), Alice inoltra il valore b''.

Al termine dell'interazione, il Bianconiglio può confrontare il valore di θ con b'': se $\theta = b''$ ha ricevuto b. D'altra parte quando $\theta \neq b''$, il Bianconiglio ha ricevuto b', che è completamente indipendente da b. Quindi Alice ha trasmesso al Bianconiglio l'input b con probabilità $1/2$, come desiderato.

($\Leftarrow$) Dato un protocollo che realizza $\frac{1}{2}$-OT, mostriamo come utilizzarlo per costruirne un altro che realizza $\binom{2}{1}$-OT; indichiamo con $(x_0, x_1) \in \{0,1\}^2$ gli input di Alice e supponiamo che il Bianconiglio voglia imparare x_θ, per un valore $\theta \in \{0,1\}$.

[86] In questo senso, il trasferimento immemore può essere visto come un protocollo a due giocatori in cui (x_0, x_1) è l'input di P_1 ed il bit θ è l'input di P_2; l'output del protocollo è la stringa vuota per P_1 ed il valore x_θ per P_2.

1. Alice sceglie una stringa casuale $\mathbf{s} = (s_1, \ldots, s_{3n}) \xleftarrow{\$} \{0,1\}^{3n}$ da usare come input per il protocollo $\frac{1}{2}$-OT. (Si intende qui che il protocollo $\frac{1}{2}$-OT è eseguito indipendentemente per ognuno dei bit che compone la stringa $\mathbf{s}$.) Al termine dell'esecuzione del protocollo, il Bianconiglio riceve approssimativamente la metà dei bit, ma non impara nulla sull'altra metà. (Senza perdita di generalità, supporremo che riceve il valore $\perp$ in corrispondenza di questi elementi.)

2. Il Bianconiglio invia ad Alice due insiemi di indici $\mathcal{I}_0, \mathcal{I}_1 \subset [3n]$ con dimensione n, tali che:

   (a) $\mathcal{I}_\theta$ contiene solo indici in corrispondenza dei quali il Bianconiglio ha imparato i corrispondenti bit in $\mathbf{s}$ (cioè per cui il Bianconiglio non ha ricevuto $\perp$);

   (b) $\mathcal{I}_{1-\theta}$ è scelto a caso.

3. Alice invia al Bianconiglio $(x_0 \oplus \bigoplus_{i \in \mathcal{I}_0} s_i, x_1 \oplus \bigoplus_{i \in \mathcal{I}_1} s_i)$.

Per il limite di Chernoff (cf. Teorema A.5), la probabilità che al termine dell'interazione al punto (1) il Bianconiglio riceva meno di n oppure più di $2n$ valori $\perp$ è esponenzialmente piccola in n. Pertanto è sempre possibile trovare un insieme di indici $\mathcal{I}_\theta$ che soddisfa la proprietà (2a). Inoltre possiamo assumere che l'insieme $\mathcal{I}_{1-\theta}$ estratto in (2b) contenga almeno un valore $\perp$. Ne segue che il Bianconiglio conosce esattamente uno tra $\bigoplus_{i \in \mathcal{I}_0} s_i$ e $\bigoplus_{i \in \mathcal{I}_1} s_i$ e quindi è in grado di calcolare esattamente uno tra x_0 e x_1, come richiesto. □

Esistono alcune generalizzazioni, ad esempio si può richiedere che il Bianconiglio impari un certo numero k tra $m > k$ segreti [NP99; CNS07] (cf. Esercizio 14.2). (Quando $m = 2$ e $k = 1$ ritroviamo $\binom{2}{1}$-OT.)

14.3 Schemi di impegno

Un problema senz'altro curioso è quello del "testa o croce" al telefono, considerato per la prima volta da Blum [Blu82]. Supponiamo che Alice ed il Bianconiglio stiano parlando al telefono; durante la conversazione si trovano in disaccordo e decidono di risolvere la contesa lanciando una moneta. Come possono effettuare il lancio di moneta a distanza, assicurandosi che nessuno dei due stia barando tentando di polarizzare il risultato?

Possiamo formalizzare il problema come un protocollo a due giocatori. Alice possiede un input $x_A \in \{0,1\}$ (che rappresenta la sua puntata sull'esito del

lancio di moneta, ovvero "testa" oppure "croce") ed il Bianconiglio possiede un input $x_B \in \{0,1\}$ (che rappresenta l'esito del lancio). Alice vince se e solo se $x_A \oplus x_B = 0$, quindi dobbiamo costruire un protocollo per calcolare la funzione $f(x_A, x_B) = (x_A \oplus x_B, x_A \oplus x_B)$ in modo sicuro.

Il problema non è banale; ad esempio Alice non può inviare la sua puntata in chiaro: se il Bianconiglio sceglie x_B *dopo* aver visto x_A, può sempre fare in modo che x_B soddisfi $x_A \oplus x_B \neq 0$. Dal punto di vista di Alice sarebbe desiderabile che, seppur il Bianconiglio si comporti in modo disonesto (ovvero in modo diverso rispetto a quanto previsto nel protocollo), al termine dell'interazione il valore $x_A \oplus x_B$ abbia comunque distribuzione uniforme. Un requisito simile è desiderabile per il Bianconiglio: al termine del protocollo il valore di output deve essere comunque uniforme, anche se Alice è disonesta (in altri termini, Alice non deve essere in grado di polarizzare l'esito del lancio).

Potremmo pensare di risolvere il problema richiedendo che le due parti si scambino i valori x_A ed x_B *contemporaneamente*. Purtroppo raramente è possibile contare su un canale perfettamente sincrono (ad esempio ciò è impensabile su Internet). Per risolvere il problema useremo uno schema d'impegno (*commitment*).

Schemi di impegno binari. Il termine "schema di impegno" è apparso per la prima volta in relazione ad un protocollo di Even [Eve81]. Tuttavia l'idea di base era già presente (indipendentemente) nei lavori di Blum [Blu82] e di Shamir, Rivest ed Adleman [SRA81]. Intuitivamente, uno schema di impegno binario consente ad Alice di impegnarsi a trasmettere al Bianconiglio un input binario b in modo che: (i) il Bianconiglio non abbia alcuna informazione su b fino a quando non sarà Alice a deciderlo e (ii) Alice non possa modificare il valore di b rispetto al quale si è precedentemente impegnata. Volendo fare una metafora, è come se Alice mettesse b in un baule chiuso con una chiave di cui lei è unica proprietaria, e spedisse il baule al Bianconiglio. Quest'ultimo non può aprire il baule, a meno che Alice non riveli la chiave. D'altra parte Alice non può più cambiare il contenuto del baule.

Uno schema di impegno che soddisfa queste proprietà, consente di risolvere facilmente il problema del lancio di moneta al telefono. Alice invia al Bianconiglio l'impegno corrispondente al suo input x_A ed il Bianconiglio risponde inviando x_B ad Alice. Quindi Alice apre l'impegno, così che il Bianconiglio impara il valore di x_A ed entrambe le parti possono calcolare $x_A \oplus x_B$. Notare che Alice può sempre rifiutarsi di aprire l'impegno dopo aver visto B (ad esempio perché sa di aver perso), ma ciò è inevitabile e può essere semplicemente

interpretato come il fatto che Alice voglia barare (quindi se rifiuta di aprire l'impegno, perde automaticamente).

Un protocollo di impegno è una coppia di algoritmi PPT $(\mathsf{Commit}, \mathsf{Open})$ definiti come segue.

- *Fase di impegno.* Dato come input il bit b, Alice calcola l'impegno $c \leftarrow \mathsf{Commit}(b; \omega)$ (usando la stringa casuale $\omega \xleftarrow{\$} \Omega$ come randomicità) ed inoltra c al Bianconiglio.

- *Fase di apertura.* Alice apre l'impegno, rivelando b ed ω. Quindi il Bianconiglio verifica che c sia il valore corretto: l'algoritmo di apertura $\mathsf{Open}(c, b; \omega) \in \{0, 1\}$ ritorna 1 se e solo se c è l'impegno relativo al bit b, usando ω come randomicità.

Sebbene non ci occuperemo esplicitamente di questo caso, la definizione può essere estesa al caso di stringhe binarie a lunghezza arbitraria (cf. Esercizio 14.3); inoltre, a volte, il calcolo della stringa di impegno c può essere interattivo (cf. Esercizio 14.7 ed Esercizio 14.8).

Non presenteremo una definizione completamente formale di sicurezza (si veda ad esempio [BCC88]), ma cercheremo di coglierne l'essenza. Il tipo di sicurezza che si ottiene dipende dal potere computazionale delle parti coinvolte. Diremo che uno schema di impegno è: (i) perfettamente vincolante (*perfectly binding*), quando Alice (pur avendo potere computazionale illimitato) non può cambiare il valore di b per cui si è impegnata e (ii) perfettamente celante (*perfectly hiding*), quando il Bianconiglio (pur avendo potere computazionale illimitato) non può recuperare b fino a quando Alice non decide di aprire l'impegno. (Quando le due parti sono computazionalmente limitate si parla, rispettivamente, di schemi computazionalmente vincolanti e computazionalmente celanti.)

Non è complesso mostrare che *non* può esistere uno schema di impegno che sia contemporaneamente perfettamente celante e perfettamente vincolante. Supponiamo che uno schema di impegno sia perfettamente vincolante: data un'esecuzione onesta del protocollo Alice (in fase di apertura) può convincere il Bianconiglio ad accettare un unico valore b come valido. Se il Bianconiglio ha potere computazionale illimitato, può provare tutti i possibili valori di randomicità associati ai valori $b = 0$ e $b = 1$, e determinare così quale dei due valori ha generato l'impegno relativo all'input b di Alice.

Concludiamo il paragrafo presentando alcuni schemi di impegno binari e discutendone la relativa sicurezza.

Uno schema basato sul problema della residuosità quadratica. Sia $N = pq$ un modulo RSA. Alice calcola la stringa d'impegno per il bit b scegliendo c uniformemente a caso in $\mathbb{QR}_N$ se $b = 0$ oppure in $\mathbb{Z}_N^* \setminus \mathbb{QR}_N$ se $b = 1$. Quindi invia (N, c) al Bianconiglio. L'algoritmo di apertura prevede che Alice invii al Bianconiglio i fattori (p, q) di N. Il Bianconiglio controlla che $N = pq$; se ciò non accade restituisce 0. Altrimenti ritorna 1 e verifica se c è un residuo quadratico o meno ($b = 0$ nel primo caso e $b = 1$ nel secondo). (Notare che l'ultima verifica può essere effettuata efficientemente una volta nota la fattorizzazione di N, cf. Appendice B.3.)

Si può mostrare che lo schema è perfettamente vincolante e computazionalmente celante come informalmente argomentato di seguito. Supponiamo che Alice abbia inviato (N, c). Ci sono due possibilità: N non è un modulo RSA, oppure lo è. Nel primo caso il Bianconiglio rifiuta; nel secondo caso, se $c \in \mathbb{QR}_N$ il Bianconiglio ottiene $b = 0$, altrimenti ottiene $b = 1$. Comunque, Alice non può cambiare la stringa d'impegno in modo che essa sia relativa ad $1-b$ anziché a b.

D'altra parte lo schema è computazionalmente celante, in quanto il Bianconiglio per distinguere $b = 0$ da $b = 1$ prima che Alice apra la stringa d'impegno deve distinguere $\mathbb{QR}_N$ da $\mathbb{Z}_N^* \setminus \mathbb{QR}_N$, il che è infattibile se il problema della residuosità quadratica è difficile (cf. Definizione 6.10).

Uno schema basato sul problema del logaritmo discreto. Il protocollo seguente è basato sulla difficoltà del problema del logaritmo discreto nell'insieme dei residui quadratici modulo p. Il Bianconiglio comincia l'interazione scegliendo un primo p, un generatore g di $\mathbb{QR}_p$ ed un elemento casuale $h \xleftarrow{\$} \mathbb{QR}_p$; invia quindi ad Alice (p, g, h). Alice controlla che p sia primo e che g ed h siano elementi di $\mathbb{QR}_p$. Sceglie quindi un elemento $\omega \xleftarrow{\$} \mathbb{Z}_{(p-1)/2}$ e calcola:

$$c = \begin{cases} g^\omega & \text{se } b = 0 \\ h \cdot g^\omega & \text{se } b = 1. \end{cases}$$

La stringa c è l'impegno di Alice relativo al bit b. Per aprire l'impegno, Alice rivela al Bianconiglio ω: se $c = g^\omega$ il Bianconiglio decide per $b = 0$; se invece $c = h \cdot g^\omega$, decide per $b = 1$.

Si può mostrare che lo schema descritto è computazionalmente celante e perfettamente vincolante, come argomentato informalmente di seguito. Per aprire l'impegno in due modi diversi, Alice necessita di conoscere due elementi

ω, ω', tali che, per un qualche c, risulti:

$$g^{\omega} = c = h \cdot g^{\omega'},$$

il che implica $g^{\omega-\omega'} = h$. Ma allora, Alice sarebbe in grado di risolvere il problema del logaritmo discreto in $\mathbb{QR}_p$ e, se questo è ritenuto difficile, lo schema deve essere computazionalmente vincolante. D'altra parte, non è difficile convincersi che c è un elemento casuale di $\mathbb{QR}_p$, così che il Bianconiglio non ha modo di distinguere il caso in cui $b = 0$ dal caso in cui $b = 1$ prima che Alice apra l'impegno.

Uno schema basato su un qualsiasi PRG. Il seguente schema è dovuto a Naor [Nao91]. Sia $G : \{0,1\}^n \to \{0,1\}^{3n}$ un PRG (cf. Definizione 3.2). Come vedremo, l'algoritmo per calcolare l'impegno è in questo caso interattivo. Il Bianconiglio comincia l'interazione scegliendo $h \xleftarrow{\$} \{0,1\}^{3n}$ ed inviandolo ad Alice. Alice sceglie $\omega \xleftarrow{\$} \{0,1\}^n$, e calcola:

$$c = \begin{cases} G(\omega) \oplus h & \text{se } b = 0 \\ G(\omega) & \text{se } b = 1. \end{cases}$$

La stringa c è l'impegno relativo al bit b. Per aprire l'impegno è sufficiente rivelare ω: se $c = G(\omega) \oplus h$, il Bianconiglio decide per $b = 0$; se invece $c = G(\omega)$, decide per $b = 1$.

Si può mostrare che lo schema è perfettamente vincolante e computazionalmente celante, come argomentato informalmente di seguito. Per aprire l'impegno in due modi differenti Alice necessita di conoscere due elementi ω, ω' tali che, per un qualche c, si abbia:

$$G(\omega) \oplus h = c = G(\omega'),$$

il che implica $G(\omega) \oplus G(\omega') = h$. Non è difficile vedere che ci sono al più $(2^n)^2$ elementi h tali che esistono ω, ω' che soddisfano $h = G(\omega) \oplus G(\omega')$. Pertanto, la probabilità che un valore $h \xleftarrow{\$} \{0,1\}^{3n}$ consenta ad Alice di aprire l'impegno in due modi diversi è $2^{2n}/2^{3n} = 2^{-n}$. Tale valore può essere reso trascurabile scegliendo opportunamente il parametro n.

D'altra parte, se il Bianconiglio fosse in grado di distinguere il caso $b = 0$ dal caso $b = 1$ prima che Alice apra l'impegno, allora sarebbe in grado di distinguere $G(\omega)$ da un elemento completamente uniforme in $\{0,1\}^{3n}$, il che è infattibile se G è un PRG computazionalmente sicuro.

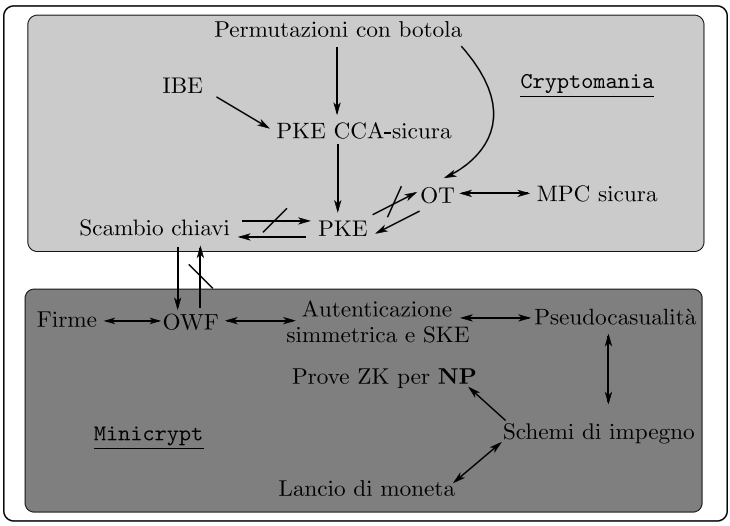


Fig. 14.2. Alcune implicazioni generali in crittografia. La separazione tra funzioni unidirezionali e trasferimento immemore è dovuta a [IR89; Ger+00]

Otteniamo come corollario che l'esistenza delle funzioni unidirezionali (e quindi dei PRG) implica l'esistenza degli schemi d'impegno. Siccome anche l'altra direzione è verificata (cioè se gli schemi di impegno esistono allora esistono le funzioni unidirezionali), gli schemi di impegno sono membri di Minicrypt. Uno schema riassuntivo di alcune implicazioni tra primitive crittografiche eterogenee è mostrata in Fig. 14.2.

14.4 MPC generale: una panoramica

In questo paragrafo ci concentreremo sul problema della MPC nella sua forma generale. Dopo aver descritto in dettaglio una costruzione concreta (tratta da [GRR98]), daremo una breve panoramica relativa ai risultati noti in quest'area.

Un caso semplice. Sia $\mathbb{F}$ un campo finito. Consideriamo m giocatori in un insieme $\mathcal{P} = \{P_1, \ldots, P_m\}$: ciascun giocatore P_i possiede un input segreto $x_i \in \mathbb{F}$; i giocatori vogliono calcolare una funzione $f(x_1, \ldots, x_m)$ dei loro input

scambiandosi messaggi attraverso canali protetti (sia per quanto riguarda la confidenzialità che per quanto riguarda l'integrità) e mantenendo la segretezza dei rispettivi input e la correttezza dell'output. Per semplicità tratteremo solamente il caso di avversari a soglia, in grado di corrompere al più $t < m$ giocatori. Ci concentreremo inoltre sul caso passivo, in cui l'avversario non può modificare il comportamento dei giocatori corrotti (quindi si limita ad imparare il valore di input e tutti i valori intermedi imparati successivamente dal giocatore). Questo modella il comportamento di avversari "onesti ma curiosi" (*honest but curious*). Diremo che un protocollo MPC in questo contesto è sicuro, se soddisfa i seguenti requisiti:

- *Segretezza.* Ogni insieme Σ di al più t giocatori non impara niente al di fuori dei valori $\{x_j\}_{j\in\Sigma}$ e del valore di output della funzione.
- *Correttezza.* Al termine del protocollo ciascun giocatore ottiene l'output $f(x_1, \ldots, x_m)$.

La funzione f che andremo a considerare è completamente arbitraria, purché calcolabile attraverso un circuito aritmetico che gestisce elementi del campo $\mathbb{F}$. Consideriamo un circuito aritmetico su $\mathbb{F}$ con m input. Un circuito aritmetico è un grafo senza cicli in cui: (i) esiste un unico nodo con grado di uscita 0 (ovvero l'output del circuito), (ii) esistono m nodi con grado di ingresso 1 (ovvero gli input del circuito), (iii) ogni nodo interno ha grado di ingresso 2 e grado di uscita 1 ed è un nodo di moltiplicazione oppure di addizione. L'output del circuito è quindi valutato in modo naturale a partire dagli input, dove tutte le operazioni sono eseguite in $\mathbb{F}$. Assumendo che ciascun giocatore P_i possegga x_i come input, vogliamo definire un protocollo per calcolare l'output del circuito in modo sicuro. Il numero di round nel protocollo sarà lineare nel numero di nodi interni al circuito; per questo motivo la soluzione che presenteremo può considerarsi pratica solo per circuiti di dimensioni non troppo grandi.

L'idea di base è quella di usare lo schema di condivisione di segreti di Shamir (cf. Paragrafo 14.1). Il protocollo finale userà al suo interno due sottoprotocolli: uno per calcolare in modo sicuro la somma di due input segreti, l'altro per calcolare in modo sicuro il prodotto di due input segreti. Ci occorre innanzitutto il seguente:

Lemma 14.4 (Somme e prodotti di segreti). *Siano $s_1, s_2 \in \mathbb{F}$ due segreti. Per $i \in \{1, 2\}$ indichiamo con $s_1^{(i)}, \ldots, s_m^{(i)}$ le porzioni di s_i ottenute usando uno schema di Shamir di tipo-(m, t). Allora i valori $s_1^{(1)} + s_1^{(2)}, \ldots s_m^{(1)} + s_m^{(2)}$ sono le porzioni di $s_1 + s_2$ relative allo stesso schema di condivisione. Analogamente, i*

Crittosistema 14.1. Protocollo di addizione dei valori x_1, x_2

Ciascun giocatore P_i (con $i \in \{1, \ldots, m\}$) possiede le porzioni $s_i^{(1)}, s_i^{(2)}$ relative agli input x_1, x_2.

- `Finalizzazione`. Ciascun giocatore P_i calcola $s_i = s_i^{(1)} + s_i^{(2)}$.

valori $s_1^{(1)} \cdot s_1^{(2)}, \ldots s_m^{(1)} \cdot s_m^{(2)}$ *sono le porzioni di* $s_1 \cdot s_2$ *in uno schema di Shamir di tipo-*$(m, 2t)$.

Dimostrazione. Siano g_1 e g_2 i due polinomi di grado al più t che generano (rispettivamente) le porzioni $s_1^{(1)}, \ldots, s_m^{(1)}$ ed $s_1^{(2)}, \ldots, s_m^{(2)}$. Osserviamo che $g_i(0) = s_i$ e $g_i(j) = s_j^{(i)}$ per $i \in \{1, 2\}$ e $j \in \{1, \ldots, m\}$. Sia $g(X) = g_1(X) + g_2(X)$. Tale polinomio ha grado al più t e soddisfa $g(0) = g_1(0) + g_2(0) = s_1 + s_2$ e $g(j) = g_1(j) + g_2(j) = s_j^{(1)} + s_j^{(2)}$. Questo mostra la prima parte dell'enunciato.

Analogamente, è immediato verificare che il polinomio $g(X) = g_1(X) \cdot g_2(X)$ ha grado al più $2t$ e soddisfa $g(0) = s_1 \cdot s_2$ e $g(j) = s_j^{(1)} \cdot s_j^{(2)}$, come richiesto. □

Supponiamo ora di disporre delle porzioni relative ai segreti x_1 ed x_2, condivisi usando uno schema di Shamir di tipo-(m, t). Per il lemma precedente ciascun giocatore può calcolare le porzioni relative alla somma $x_1 + x_2$ *senza* interazione, come mostrato nel Crittosistema 14.1.

In modo simile, ciascun giocatore può calcolare le porzioni del prodotto $x_1 \cdot x_2$; tuttavia, come visto, queste sono relative ad uno schema di Shamir di tipo-$(m, 2t)$. Mostriamo che, interagendo, i giocatori possono calcolare le porzioni di $x_1 \cdot x_2$ relative ad uno schema di Shamir di tipo-(m, t). L'interazione è mostrata nel Crittosistema 14.2. Notare che, per il Lemma 14.4, i valori $s_1, \ldots, s_m$ sono le porzioni relative al valore $x_1 \cdot x_2$ in uno schema di Shamir di tipo-$(m, 2t)$. Pertanto $x_1 \cdot x_2 = \sum_{j=1}^{m} \alpha_j s_j$. Siccome però i valori $\overline{s}_1^{(j)}, \ldots, \overline{s}_m^{(j)}$ sono le porzioni relative ad s_j in uno schema di Shamir di tipo-(m, t), il Lemma 14.4 implica che i valori $u_1, \ldots, u_m$ sono anch'essi porzioni di $x_1 \cdot x_2$ nello stesso schema di tipo-(m, t). Affinché la segretezza sia mantenuta è necessario comunque assumere che $m = 2t + 1$, ovvero che la maggioranza dei giocatori sia onesta.

La versione finale del protocollo è mostrata nel Crittosistema 14.3. Indichiamo con $G_1, \ldots, G_d$ i nodi del circuito, ed assumiamo che esista un lato da G_j a G_i se e solo se $i > j$; i primi m nodi sono etichettati con gli input del circuito. Notare che, per la correttezza dei protocolli di addizione e moltiplicazione definiti in precedenza, alla fine dell'interazione i giocatori posseggono le

Crittosistema 14.2. Protocollo di moltiplicazione dei valori x_1, x_2.

Ciascun giocatore P_i (con $i \in \{1, \ldots, m\}$) possiede le porzioni $s_i^{(1)}, s_i^{(2)}$ relative agli input x_1, x_2.

- `Round 1.` Il giocatore P_i calcola $s_i = s_i^{(1)} \cdot s_i^{(2)}$. Quindi genera le porzioni relative ad s_i utilizzando uno schema di Shamir di tipo-(m, t). Indichiamo con $\overline{s}_1^{(i)}, \ldots, \overline{s}_m^{(i)}$ tali porzioni. Ciascun giocatore P_j invia la porzione $\overline{s}_i^{(j)}$ al giocatore P_i.
- `Finalizzazione.` Siano $\alpha_1, \ldots, \alpha_m$ le costanti definite in Eq. (14.1), per la ricostruzione del segreto in uno schema di Shamir di tipo-$(m, 2t)$. Ciascun giocatore P_i calcola $u_i = \sum_{j=1}^{m} \alpha_j \cdot \overline{s}_i^{(j)}$.

porzioni relative all'output y del circuito; quindi la proprietà di correttezza è rispettata. D'altra parte in ciascun round, ogni insieme di t giocatori corrotti possiede solamente t porzioni dello schema di Shamir di tipo-(m, t) sottostante; pertanto anche la proprietà di segretezza è rispettata.

Panoramica dei risultati. Un primo aspetto da considerare volendo definire la sicurezza di uno schema MPC, è il potere dell'avversario. Innanzitutto, dobbiamo considerare *quali* giocatori l'attaccante è in grado di corrompere. Ciò è fatto specificando la cosiddetta *struttura dell'avversario* Δ. Abbiamo già discusso il caso di struttura a soglia: l'attaccante può corrompere un qualsiasi sottoinsieme di $\mathcal{P}$, purché esso abbia dimensione inferiore ad un certo parametro $t \in \mathbb{N}$ (detto appunto la soglia).

Fissata quindi una struttura dell'avversario Δ, possiamo classificare gli attaccanti in base:

1. *Alla loro natura.* Ovvero, attaccanti *passivi* ed attaccanti *attivi*. Un attaccante passivo impara tutti i segreti ed i messaggi ricevuti dai giocatori corrotti, con la speranza di ricavare qualche informazione non lecita al termine del protocollo; comunque, tutti i giocatori corrotti continuano ad eseguire il protocollo in modo onesto (questo è il caso di attaccanti "onesti ma curiosi" che abbiamo analizzato precedentemente). Un attaccante attivo può ulteriormente modificare il comportamento dei giocatori corrotti, deviandolo da quello normalmente previsto dal protocollo.

Crittosistema 14.3. Protocollo per la computazione sicura di un circuito aritmetico

Ciascun giocatore P_i (con $i \in \{1, \ldots, m\}$) possiede un input $x_i \in \mathbb{F}$.

- `Inizializzazione.` Il giocatore P_i condivide x_i usando uno schema di Shamir di tipo-(m, t) (la cui descrizione è nota a tutti). Siano $s_1^{(i)}, \ldots, s_m^{(i)}$ le porzioni risultanti. Ogni giocatore P_i invia $s_j^{(i)}$ al giocatore P_j.
- `Computazione.` Per $k = m + 1, \ldots, d$, si calcola l'output del nodo G_k come segue:
  1. Supponiamo che i lati in ingresso al nodo G_k provengano dai nodi G_i, G_j (per $i, j < k$) ed indichiamo con $s_1^{(i)}, \ldots, s_m^{(i)}$ ed $s_1^{(j)}, \ldots, s_m^{(j)}$ le porzioni dell'output dei nodi G_i e G_j.
  2. Se G_k è un nodo di addizione, ciascun giocatore calcola localmente la porzione della somma degli output dei nodi G_i e G_j come descritto nel Crittosistema 14.1.
  3. Se G_k è un nodo di moltiplicazione, i giocatori eseguono il protocollo del Crittosistema 14.2 per calcolare le porzioni relative al prodotto degli output dei nodi G_i e G_j.
- `Finalizzazione.` Ciascun giocatore P_i invia la porzione $s_i^{(d)}$ al giocatore P_1. Quindi P_1 ricostruisce l'output del circuito y a partire da t porzioni, come previsto nello schema di Shamir di tipo-(m, t) sottostante, ed invia il valore ottenuto a tutti gli altri giocatori.

2. *Alla loro attività.* Ovvero, attaccanti *statici* ed attaccanti *adattivi*. Un attaccante statico decide quali giocatori corrompere una volta per tutte all'inizio del protocollo. Un attaccante adattivo può corrompere nuovi giocatori *durante* l'esecuzione del protocollo (scegliendo le nuove parti corrotte sulla base di quanto ha imparato fino a quel momento) purché l'insieme totale dei giocatori corrotti sia ammissibile, ovvero sia sempre un sottoinsieme di Δ.

3. *Al loro potere computazionale.* Vale a dire, avversari *limitati computazionalmente* ed avversari *illimitati computazionalmente*.

Ciò premesso, bisogna specificare come definire formalmente la sicurezza di

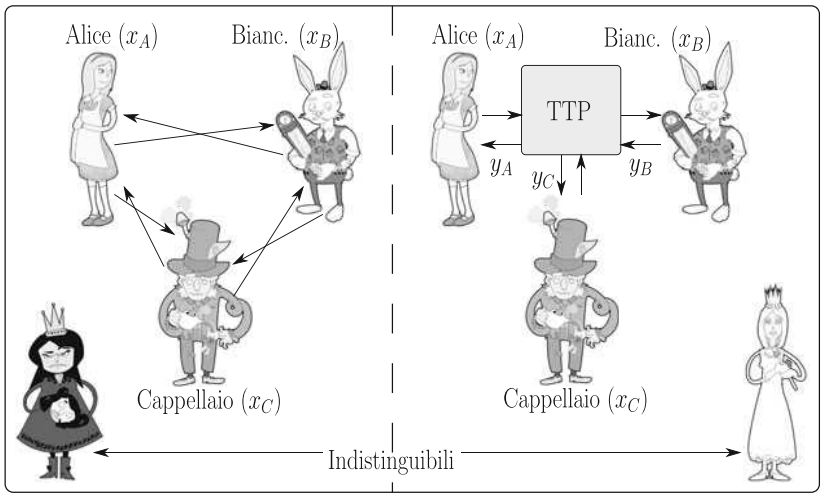


Fig. 14.3. Sicurezza di un protocollo MPC per il calcolo di $(y_A, y_B, y_C) = f(x_A, x_B, x_C)$

un protocollo MPC. Tipicamente, ci si affida al paradigma della simulazione che abbiamo già incontrato nel Capitolo 13. Supponiamo che Alice, il Bianconiglio ed il Cappellaio Matto eseguano un protocollo MPC. La sicurezza del protocollo è definita comparando il mondo "reale" (in cui avviene l'interazione tra i giocatori) ad un mondo "ideale" in cui si assume l'esistenza di una terza parte fidata (*Trusted Third Party*, TTP). Se esiste una tale entità (di cui tutti si fidano), Alice, il Bianconiglio ed il Cappellaio, possono semplicemente consegnare il loro input alla TTP, che si preoccuperà di calcolare per loro l'output e di restituire il risultato a ciascun giocatore. (In questo mondo "ideale", dunque, il problema della MPC è banale.) Intuitivamente, un protocollo MPC è sicuro se simula abbastanza bene il mondo "ideale", ovvero se tutto ciò che la Regina Rossa (l'avversario con struttura Δ) può imparare corrompendo i giocatori nel mondo "reale" può essere simulato dalla Regina Bianca (il simulatore) nel mondo "ideale" (cf. Fig. 14.3). La simulazione può essere *perfetta* (se la distribuzione globale nel mondo reale e nel mondo ideale sono identicamente distribuite), *statistica* (se le due distribuzioni sono "vicine" in distanza statistica, cf. Definizione A.3) oppure *computazionale* (se le due distribuzioni sono computazionalmente indistinguibili).

Tab. 14.1. Panoramica dei risultati per MPC sicura nel caso di avversari a soglia

Sicurezza	Avversario	Condizione	Riferimento
computazionale	passivo	$t < m$	[GMW87]
computazionale	attivo	$t < m/2$	[GMW87]
perfetta	passivo	$t < m/2$	[BOGW88; CCD88]
perfetta	attivo	$t < m/3$	[BOGW88; CCD88]
perfetta, con broadcast	attivo	$t < m/2$	[RBO89; Bea91]

I risultati per il caso di avversari a soglia (cioè in grado di corrompere al più $t < m$ giocatori) sono riassunti nella Tab. 14.1. Il caso della sicurezza computazionale è stato studiato per la prima volta da Goldreich, Micali e Wigderson [GMW87]. Nel caso di attaccanti attivi, la condizione necessaria e sufficiente per ottenere sicurezza computazionale è che si abbia $t < m/2$; nel caso di attaccanti passivi, la condizione è $t < m$.

Il caso di sicurezza perfetta è stato analizzato da Ben-Or, Goldwasser e Wigderson [BOGW88], i quali hanno provato che la condizione è $t < m/3$ nel caso attivo e $t < m/2$ nel caso passivo. Lo stesso risultato è stato dimostrato indipendentemente da Chaum, Crèpeau e Damgård [CCD88]. Qualora sia presente un canale di radiodiffusione circolare (detto anche di *broadcast*) — ovvero un canale che diffonde a tutti i giocatori il messaggio che riceve in input — si può ottenere sicurezza perfetta se e solo se $t < m/2$ [RBO89; Bea91]. Esiste anche un modello misto, in cui alcuni giocatori possono essere corrotti in modo attivo ed altri in modo passivo. Se t_p è la soglia per le parti corrotte passivamente e t_a quella per le parti corrotte attivamente, si può ottenere sicurezza perfetta se e solo se $3t_a + 2t_p < m$ [FHM98].

Tutti i risultati in Tab. 14.1 possono essere generalizzati al caso di avversario con struttura arbitraria. Consideriamo la seguente operazione (commutativa ed associativa) $\sqcup$: per ogni collezione di insiemi Δ_1, Δ_2 definiamo $\Delta_1 \sqcup \Delta_2$ come la struttura composta dall'unione di tutti gli elementi di Δ_1 con tutti gli elementi di Δ_2. In simboli:

$$\Delta_1 \sqcup \Delta_2 = \{\mathcal{S}_1 \cup \mathcal{S}_2 : \ \mathcal{S}_1 \in \Delta_1 \text{ e } \mathcal{S}_2 \in \Delta_2\}.$$

Data una struttura dell'avversario Δ, sia $\mathbb{Q}^2(\Delta)$ la condizione per cui non esistono due insiemi in Δ in grado di coprire tutto $\mathcal{P}$, ovvero:

$$\mathbb{Q}^2(\Delta) \Leftrightarrow \mathcal{P} \notin \Delta \sqcup \Delta.$$

Analogamente, sia $\mathbb{Q}^3(\Delta)$ la condizione per cui non esistono tre insiemi in Δ in grado di coprire tutto $\mathcal{P}$, ovvero:

$$\mathbb{Q}^3(\Delta) \Leftrightarrow \mathcal{P} \notin \Delta \sqcup \Delta \sqcup \Delta.$$

Si può mostrare [HM97] che, nel caso in cui la struttura dell'avversario è arbitraria, la condizione necessaria e sufficiente per la sicurezza diventa $\mathbb{Q}^2(\Delta)$ nel caso passivo e $\mathbb{Q}^3(\Delta)$ nel caso attivo. (La presenza di una canale di broadcast rilassa la condizione a $\mathbb{Q}^2(\Delta)$ anche nel caso attivo.)

Rapporto tra sicurezza statica ed adattiva. Finora non abbiamo fatto distinzione tra avversari adattivi ed avversari statici. L'intuito ci porterebbe a concludere che un avversario adattivo sia sempre più potente di uno statico. Sorprendentemente questa intuizione non sempre è corretta, come mostrato in [Can+04]. Ad esempio si può mostrare che se il numero di parti non è eccessivamente grande, le due nozioni sono di fatto equivalenti.

Componibilità universale. Supponiamo di avere due protocolli MPC Π_1 e Π_2 che implementano in modo sicuro (rispettivamente) le funzioni f_1 ed f_2. Cosa si può dire della sicurezza relativa al protocollo $\Pi_1 \circ \Pi_2$ ottenuto componendo i singoli protocolli? Qualora si potesse mostrare che la sicurezza è mantenuta sarebbe possibile progettare protocolli crittografici complessi in maniera modulare, preoccupandosi solamente della sicurezza dei singoli blocchi, semplificando così l'analisi di sicurezza di un sistema complesso. Canetti [Can00; Can01] ha formulato e dimostrato un potente "teorema di composizione" che permette di individuare le condizioni in cui ciò sia possibile. Il paradigma della componibilità universale (*Universal Composability*, UC) è stato studiato intensamente negli anni successivi, ad esempio per i protocolli di scambio delle chiavi [CH06], per gli schemi di autenticazione su base password [Can+05] e per i sistemi di prova a conoscenza nulla [GMY06].

14.5 Voto elettronico

La panoramica che segue è ispirata a [TI05; MKD07]. Il termine voto elettronico (*electronic voting*, e-voting) è un termine generale che si applica a diversi scenari. Essenzialmente, un sistema di voto elettronico consente di realizzare elezioni in cui i voti espressi dai votanti sono registrati attraverso mezzi digitali. Il caso più semplice è quello in cui il votante deve comunque recarsi a votare in punti prestabiliti supervisionati da un'entità autorizzata. L'identificazione dei

votanti può avvenire con mezzi ordinari (come avviene nelle elezioni odierne, tramite documento di riconoscimento) o con mezzi digitali (ad esempio attraverso il riconoscimento di qualche tratto biometrico), ma il voto ed il conteggio dei voti sono effettuati tramite dispositivi elettronici.

Più complesso è lo scenario in cui gli utenti possono votare da remoto, ad esempio attraverso Internet. È bene sottolineare che l'ostacolo principale nell'impiego di un sistema di voto elettronico da Internet non è esclusivamente legato alla sicurezza, ma comprende una serie di altri impedimenti politici, sociali, tecnologici e di costo [TI05]. Un sistema di voto elettronico che abbia la pretesa di essere usato su larga scala, deve essere progettato secondo un insieme ben definito di criteri universalmente riconosciuti [Neu93; BT94; CC97]. In particolare dovrebbe essere soddisfatte le seguenti proprietà (in termini di sicurezza e praticità).

- *Completezza.* Solo gli aventi diritto al voto devono poter votare. Inoltre, se tutte le parti coinvolte sono oneste, il sistema deve funzionare correttamente.
- *Singolo voto.* Ciascun votante deve votare una ed una sola volta.
- *Anonimato.* Durante l'elezione tutti i voti devono restare segreti e non deve essere possibile collegare un voto a chi lo ha espresso.
- *Imparzialità.* Non deve essere possibile predire l'esito della votazione prima che questa termini.
- *Assenza di ricevuta.* Nessun avente diritto al voto deve essere in grado di dimostrare a terzi la preferenza espressa durante la votazione.
- *Incoercibilità.* Non deve essere possibile costringere un votante a rivelare il suo voto.
- *Correttezza del conteggio.* Chiunque osservi il processo dall'esterno deve essere convinto che il conteggio dei voti avvenga in modo onesto.
- *Verificabilità.* Tutti devono essere convinti che i voti espressi siano stati conteggiati.
- *Validità.* Nessuno deve poter interferire con l'elezione (interrompendola oppure influenzandone il risultato).
- *Onestà.* Se l'autorità è onesta, nessun votante deve poter dimostrare il contrario. D'altra parte un'autorità disonesta non deve poter pilotare l'elezione.

- *Praticità.* Il sistema deve consentire a chi lo usa di votare in modo rapido, con attrezzatura minima e senza richiedere alcuna abilità particolare.

Notare come molti dei requisiti automaticamente soddisfatti nelle elezioni canoniche, diventano non banali per un sistema di voto elettronico. Inoltre, alcuni requisiti sembrano a prima vista contraddittori: non è chiaro, ad esempio, come un sistema possa essere privo di ricevuta e verificabile allo stesso tempo. Altre problematiche sono invece completamente nuove; ad esempio, le proprietà di assenza di ricevuta (*receipt-freeness*) ed incoercibilità (*uncoercibility*), nel caso di una normale elezione, sono soddisfatte per costruzione: poiché la votazione avviene in pubblico non si necessita di alcuna ricevuta e quindi nessuno può "vendere" il suo voto.

Uno schema basato sulle firme cieche. Tipicamente un sistema di voto elettronico si articola in tre fasi. (i) *registrazione*: prima dell'elezione i potenziali votanti dimostrano la propria identità ed il loro diritto a votare; (ii) *validazione*: durante l'elezione i votanti sono autenticati prima di esprimere la propria preferenza; (iii) *raccolta dei voti e conteggio*: si raccolgono tutti i voti, si esegue il conteggio e si pubblicano i risultati. Discutiamo qui una semplice realizzazione, basata su uno schema di firma cieca.

Uno schema di firma cieca (*blind signature*) — introdotto per la prima volta da Chaum [Cha83] — consente di ottenere la firma di un messaggio in modo che chi genera la firma non sia in grado di risalire al contenuto del messaggio firmato (in questo senso la firma è prodotta "alla cieca"). Per realizzare una firma cieca, si può usare una coppia di funzioni (χ, χ^{-1}), come segue. Supponiamo che il Bianconiglio voglia ottenere una firma (alla cieca) di Alice, per un messaggio x. Per prima cosa il Bianconiglio calcola $x' = \chi(x;\omega)$, essendo ω un valore casuale, ed invia x' ad Alice. Quindi, Alice genera la firma $\sigma(x')$ relativa al messaggio x' e la restituisce al Bianconiglio, il quale può ottenere la firma $\sigma(x)$ relativa ad x calcolando $\sigma(x) = \chi^{-1}(\sigma(x');\omega)$.

Per chiarire le idee consideriamo il caso di RSA (cf. Crittosistema 6.1). Supponiamo che il Bianconiglio voglia ottenere una firma cieca del messaggio x. A tale scopo calcola dapprima $x' = \chi(x,\omega) = x \cdot \omega^e \bmod N$, essendo ω un valore casuale tale che $\gcd(\omega, N) = 1$, ed invia x' ad Alice. Alice firma quindi il messaggio x', calcolando $\sigma(x') = (x')^d \bmod N$. Il Bianconiglio può ricavare la firma di x calcolando $\sigma(x) = \chi^{-1}(\sigma(x'),\omega) = \sigma(x') \cdot \omega^{-1} \bmod N$. Siccome $\omega^{ed} \equiv 1 \bmod N$, infatti:

$$\sigma(x) \equiv \sigma(x') \cdot \omega^{-1} \equiv (x')^d \cdot \omega^{-1} \equiv x^d \omega^{ed} \omega^{-1} \equiv x^d \pmod{N}.$$

Consideriamo un insieme $\{V_1, \ldots, V_m\}$ di votanti, un amministratore A ed una bacheca pubblica. La soluzione di Fujioka, Okamoto e Ohta [FOO92] consente di ottenere uno schema di voto elettronico usando una firma cieca ed uno schema di impegno (cf. Paragrafo 14.3), assumendo che siano disponibili *canali anonimi* tra i votanti e l'amministratore. Tali canali consentono di inviare un messaggio mantenendo l'anonimato e possono essere realizzati usando, ad esempio, le tecniche in [Cha81]. Si assume inoltre che ciascun votante possegga le chiavi relative ad uno schema di firma digitale, mentre l'amministratore dispone della chiave segreta dello schema di firma cieca. L'elezione avviene come di seguito:

- *Preparazione.* L'elettore V_j (con identità id_j) seleziona il suo voto v_j e calcola l'impegno $c_j \leftarrow \mathsf{Commit}(v_j; \omega_j)$ (con randomicità ω_j). Quindi pone $c'_j = \chi(c_j, \omega'_j)$ (con randomicità ω'_j), genera la firma $\sigma(c'_j)$ relativa al valore c'_j ed invia la tripletta $(id_j, c'_j, \sigma(c'_j))$ ad A, usando il canale anonimo.
- *Amministrazione.* L'amministratore verifica (attraverso l'identità id_j) che l'elettore abbia diritto al voto e controlla di non aver ricevuto altri voti corrispondenti ad id_j. Successivamente verifica la firma $\sigma(c'_j)$ e (se la verifica và a buon fine) firma a sua volta c'_j calcolando $\sigma_A(c'_j)$; invia quindi a V_j il risultato. Infine, A pubblica sulla bacheca la terna $(id_j, c'_j, \sigma(c'_j))$.
- *Voto.* Il votante V_j può ora ottenere la firma $\sigma_A(c_j)$, relativa all'impegno c_j, calcolando $\sigma_A(c_j) = \chi^{-1}(\sigma_A(c'_j), \omega'_j)$. Dopo aver verificato che $(c_j, \sigma_A(c_j))$ è effettivamente una firma valida, invia la coppia $(c_j, \sigma_A(c_j))$ sul canale anonimo.
- *Raccolta dei voti.* Dopo aver controllato che $\sigma_A(c_j)$ sia una firma valida, il gestore della bacheca popola una lista $\mathcal{L}$ contenente le terne del tipo $(j, c_j, \sigma_A(c_j))$ (qui j è semplicemente l'indice di riga). Al termine della votazione la lista è resa pubblica.
- *Apertura.* Ciascun votante può controllare che il suo voto sia presente nella lista pubblicata ed in caso contrario dimostrare, esibendo $(c_j, \sigma_A(c_j))$, che il suo voto (già validato dall'amministratore) è assente. Inoltre V_j può verificare che il numero di voti espressi coincida con il numero previsto di votanti. Terminate queste verifiche V_j impara il valore dell'indice i associato al suo voto ed apre l'impegno inviando la coppia (i, ω_j) sul canale anonimo.
- *Conteggio.* Tramite il valore ω_j, il gestore della bacheca può aprire l'impegno e recuperare v_j, verificare che sia un voto valido e conteggiare tutti i voti. Non resta che pubblicare l'esito della votazione.

Notare che l'uso del canale anonimo nasconde (di fatto) la corrispondenza tra elettore e voto espresso (per questo motivo il requisito di anonimato è banalmente soddisfatto). Non è complesso verificare che (se lo schema di impegno, gli schemi di firma degli utenti e lo schema di firma cieca sono sicuri) la costruzione descritta soddisfa tutti i requisiti di sicurezza tranne le proprietà di incoercibilità e di assenza di ricevute (cf. Esercizio 14.10). Si rimanda a [Ohk+99] per alcune estensioni.

Altri approcci. Esistono diverse tecniche per implementare uno schema di voto elettronico e numerose soluzioni sono apparse in letteratura [CGS97; Abe98; Bau+01; BG02; JJR02; DJN10]. La proprietà di incoercibilità si è rivelata particolarmente complessa da realizzare; in proposito si vedano [BT94; NR94; SK95; HS00; UMQ10].

Esercizi

Esercizio 14.1. Sia $\mathbb{F} = \mathbb{Z}_p$ per $p = 31$. Consideriamo uno schema di condivisione di segreti di tipo-$(3, 7)$, per condividere il segreto $s = 7 \in \mathbb{Z}_{31}$. Il polinomio di condivisione è $g(X) = 7 + 19 \cdot X + 21 \cdot X^2$.

1. Calcolare le porzioni di s e mostrare come esso possa essere recuperato da un qualsiasi insieme di 3 porzioni.

2. Recuperare $g(X)$ da un qualsiasi insieme di 3 porzioni.

Esercizio 14.2. Dato uno schema $\binom{2}{1}$-OT, costruire uno schema $\binom{m}{1}$-OT. *(Suggerimento: supponiamo che Alice possegga m stringhe $x_1, \ldots, x_m \in \{0,1\}^n$; il Bianconiglio vuole imparare la stringa corrispondente all'indice $\theta \in [m]$. A tale scopo, i due interagiscono come indicato di seguito. Alice sceglie $y_0 = 0^n$ ed $y_1, \ldots, y_m \xleftarrow{\$} \{0,1\}^n$. Quindi, per ogni $j = 1, \ldots, m$, il Bianconiglio calcola $y_j^* = x_j \oplus \bigoplus_{i=0}^{j-1} y_i$ e i due eseguono il protocollo $\binom{2}{1}$-OT con input (y_j^*, y_j); se $\theta = j$, il Bianconiglio decide di imparare y_j^* ed altrimenti impara y_j.)*

Esercizio 14.3. Formulare una definizione di uno schema di impegno perfettamente vincolante per stringhe binarie. Supponiamo di costruire uno schema di impegno per stringhe binarie inviando l'impegno relativo a ciascun bit indipendentemente. Dimostrare che se lo schema binario è computazionalmente celante, anche lo schema ottenuto soddisfa la stessa proprietà.
(Suggerimento: usare un argomento ibrido.)

Esercizio 14.4. Consideriamo il seguente protocollo per il lancio di moneta al telefono, basato su uno schema di impegno perfettamente celante e computazionalmente vincolante.

> Alice invia al Bianconiglio l'impegno relativo al bit $x_A \in \{0,1\}$. Il Bianconiglio invia ad Alice l'impegno relativo al bit $x_B \in \{0,1\}$. Alice apre l'impegno, rivelando così x_A al Bianconiglio. Quindi il Bianconiglio apre a sua volta l'impegno, rivelando x_B ad Alice. (Se uno dei due si accorge che l'altro ha barato termina il protocollo.) Il risultato del lancio di moneta è $x_A \oplus x_B$.

Rispondere alle seguenti domande.

1. Spiegare perché Alice, non può polarizzare il risultato del lancio.

2. Supponiamo che Alice ed il Bianconiglio utilizzino lo schema di impegno del Paragrafo 14.3, basato sul problema del logaritmo discreto. Mostrare che il Bianconiglio può polarizzare il risultato del lancio di moneta.

Esercizio 14.5. Sia $(\mathscr{P}, \mathscr{V})$ un protocollo-Σ. Consideriamo il seguente schema di impegno.

> Sia ϱ una relazione difficile per il linguaggio $\mathcal{L}$ (cf. Esercizio 13.9) con algoritmo di generazione Gen. Supponiamo che sia sempre possibile riconoscere in modo efficiente che $x \in \mathcal{L}$, ovvero dato x è facile decidere se esiste w tale che $\varrho(x, w) = 1$. Inizialmente il Bianconiglio lancia Gen, ottenendo $(x, w) \leftarrow \mathsf{Gen}(1^n)$ ed invia x ad Alice, che controlla se $x \in \mathcal{L}$. Per vincolarsi alla stringa $\beta \in \{0,1\}^\ell$, Alice lancia il simulatore di HVZK $\mathscr{Z}$ con input (x, β), ottenendo (α, β, γ) ed inoltrando α al Bianconiglio. Per aprire l'impegno, Alice invia (β, γ) ed il Bianconiglio verifica che $\pi = (\alpha, \beta, \gamma)$ è accetta da $\mathscr{V}$ (relativamente all'input x).

Spiegare perché è necessario che il protocollo-Σ soddisfi la proprietà di S-HVZK. Mostrare che lo schema di impegno ottenuto è perfettamente celante e computazionalmente vincolante.

Esercizio 14.6. Dato un grafo $G = (V, E)$ si definisce cammino hamiltoniano un cammino tale che ogni nodo è attraversato una sola volta. Consideriamo il seguente sistema di prova $(\mathscr{P}, \mathscr{V})$, in cui dato il grafo $G = (V, E)$ come input comune (dove $|V| = n$), il dimostratore $\mathscr{P}$ vuole convincere $\mathscr{V}$ che conosce un ciclo hamiltoniano $C \subset E$.

1. $\mathscr{P}$ seleziona una permutazione casuale π dei vertici in V e calcola l'impegno (utilizzando uno schema di impegno perfettamente vincolante) relativo agli elementi della matrice delle adiacenze del grafo permutato. In altri termini, $\mathscr{P}$ invia una matrice $n \times n$ in cui l'elemento in posizione $(\pi(i), \pi(j))$ è l'impegno relativo al valore 1 se e solo se $(i, j) \in E$; altrimenti l'elemento in posizione $(\pi(i), \pi(j))$ contiene l'impegno relativo al valore 0.

2. $\mathscr{V}$ seleziona un bit casuale $b \xleftarrow{\$} \{0, 1\}$ e lo invia a $\mathscr{P}$.

3. La risposta di $\mathscr{P}$ dipende dal valore b ricevuto:

   - se $b = 0$, allora $\mathscr{P}$ invia π ed apre l'impegno relativo a tutti i valori nella matrice di adiacenze;
   - se $b = 1$, allora $\mathscr{P}$ apre l'impegno relativo a (tutti e soli) gli elementi in posizione $(\pi(i), \pi(j))$ per ogni $(i, j) \in C$.

4. $\mathscr{V}$ accetta se e solo se:

   - quando $b = 0$, il grafo ottenuto applicando π è isomorfo a G;
   - quando $b = 1$, tutti gli impegni aperti sono 1 ed i nodi corrispondenti formano un ciclo lungo n.

   In entrambi i casi $\mathscr{V}$ controlla che i valori aperti siano corretti.

Dimostrare che $(\mathscr{P}, \mathscr{V})$ ha errore di validità trascurabile e che è a conoscenza nulla.

Esercizio 14.7. Sia $(\mathscr{P}, \mathscr{V})$ un protocollo-Σ che soddisfa la proprietà di HV-ZK speciale. Indichiamo inoltre con $(\mathsf{Commit}, \mathsf{Open})$ uno schema di impegno perfettamente celante, in cui l'algoritmo Commit è un protocollo interattivo definito come segue. Supponiamo che Alice voglia impegnarsi per la stringa x; il Bianconiglio invia un messaggio h, quindi l'impegno è calcolato in funzione di tale valore come $c \leftarrow \mathsf{Commit}_h(x; \omega)$. Consideriamo il seguente protocollo interattivo $(\mathscr{P}', \mathscr{V}')$:

> Inizialmente $\mathscr{P}'$ invia il primo messaggio h dell'algoritmo di impegno. Il verificatore si impegna ad usare la sfida β, inviando al dimostratore $c \leftarrow \mathsf{Commit}_h(\beta; \omega)$. Quindi $\mathscr{P}'$ lancia $\mathscr{P}$ per ottenere α ed invia il risultato a $\mathscr{V}'$. Il verificatore apre l'impegno inviando (β, ω). A questo punto $\mathscr{P}'$ verifica che $c \leftarrow \mathsf{Commit}_h(\beta; \omega)$ e, se questo è il caso, calcola la risposta γ utilizzando $\mathscr{P}$. Il verificatore $\mathscr{V}'$ accetta se e solo se $\mathscr{V}$ accetta $(x, (\alpha, \beta, \gamma))$.

Rispondere alle seguenti domande.

1. Verificare che la proprietà di validità è soddisfatta.

2. Mostrare che $(\mathscr{P}', \mathscr{V}')$ è a conoscenza nulla. *(Suggerimento: considerare il seguente simulatore $\mathscr{Z}'$. Dopo aver ricevuto l'impegno c, lancia il simulatore di S-HVZK del protocollo $(\mathscr{P}, \mathscr{V})$, ottenendo $\alpha' \leftarrow \mathscr{Z}(x, \beta')$, dove il valore β' è scelto a caso. Se il verificatore si rifiuta di aprire l'impegno, $\mathscr{Z}'$ ritorna $\perp$. Altrimenti, $\mathscr{Z}'$ può recuperare β. Una volta noto β, il simulatore può riavvolgere l'esecuzione al punto in cui il verificatore ha già mandato c. Quindi lancia nuovamente $\mathscr{Z}(x, \beta)$ (usando stavolta il valore corretto di β) per ottenere il corrispondente valore α. Se, a questo punto, il verificatore si rifiuta di aprire l'impegno, il simulatore riavvolge*

nuovamente e prova ancora, fintanto che il verificatore non apre l'impegno. Noti α e β, si può infine ricavare γ completando la simulazione. Con quale probabilità la simulazione ha successo?)

3. La stessa simulazione funziona se lo schema di impegno è computazionalmente celante? Giustificare la risposta.

4. Stabilire se $(\mathscr{P}', \mathscr{V}')$ è anche una prova di conoscenza. *(Suggerimento: la risposta è no; spiegare perché. Si veda l'Esercizio 14.9 per una soluzione al problema.)*

Esercizio 14.8. Il seguente schema di impegno è dovuto a Pedersen. Sia $\mathbb{G}$ un gruppo finito con ordine un primo q. Supponiamo che Alice voglia impegnarsi per il valore $x \in \mathbb{Z}_q$. L'algoritmo Commit è un protocollo interattivo in cui inizialmente il Bianconiglio sceglie un valore casuale ω' ed invia $h = g^{\omega'}$ ad Alice. Quindi Alice calcola l'impegno come $c = g^{\omega} \cdot h^{x} \leftarrow \mathsf{Commit}_h(x; \omega)$.

1. Mostrare che lo schema è perfettamente celante. *(Suggerimento: osservare che per ogni x, y esistono sempre valori ω_x, ω_y tali che $\omega_x + \omega' \cdot x = \omega_y + \omega' \cdot y \bmod q$.)*

2. Mostrare che lo schema è computazionalmente vincolante. *(Suggerimento: osservare che se ciò non fosse vero sarebbe possibile calcolare il logaritmo discreto di h.)*

Esercizio 14.9. Diciamo che uno schema di impegno è *equivocabile*, se esiste una botola *tk* tale che, data *tk*, è possibile aprire un impegno c verso un valore arbitrario (quindi diverso dal valore rispetto a cui ci si è precedentemente impegnati).

1. Mostrare che il protocollo di Pedersen dell'esercizio precedente è equivocabile, con botola $tk = \omega'$ pari al logaritmo discreto di h.

2. Consideriamo la seguente modifica del protocollo $(\mathscr{P}', \mathscr{V}')$ dell'Esercizio 14.7. Supponiamo che il verificatore $\mathscr{V}'$ utilizzi lo schema di Pedersen. Dopo che $\mathscr{V}'$ ha aperto l'impegno e $\mathscr{P}'$ ha recuperato β, il dimostratore risponde inviando γ ed in aggiunta la botola $tk = \omega'$. Il verificatore controlla che *tk* sia una botola valida (cioè che $h = g^{\omega'}$) e che $(x, (\alpha, \beta, \gamma))$ sia una prova accettata.

Mostrare che Σ' è una prova di conoscenza a conoscenza nulla. *(Suggerimento: sfruttare il fatto che se esiste $\mathscr{P}^*$ in grado di convincere il verificatore, allora abbiamo imparato la botola tk e quindi è sufficiente riavvolgere al punto in cui il verificatore apre l'impegno ed aprire l'impegno verso un valore arbitrario usando tk.)*

Esercizio 14.10. Analizzare la sicurezza del protocollo di voto elettronico presentato nel Paragrafo 14.5. Spiegare perché il protocollo non soddisfa le proprietà di assenza di ricevuta ed incoercibilità.

Letture consigliate

[Abe98] Masayuki Abe. "Universally Verifiable Mix-net with Verification Work Indendent of the Number of Mix-servers". In: *EUROCRYPT*. 1998, pp. 437–447.

[Bau+01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern e Guillaume Poupard. "Practical multi-candidate election system". In: *PODC*. 2001, pp. 274–283.

[BCC88] Gilles Brassard, David Chaum e Claude Crépeau. "Minimum Disclosure Proofs of Knowledge". In: *J. Comput. Syst. Sci.* 37.2 (1988), pp. 156–189.

[Bea91] Donald Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority". In: *J. Cryptology* 4.2 (1991), pp. 75–122.

[Bei11] Amos Beimel. "Secret-Sharing Schemes: A Survey". In: *IWCC*. 2011, pp. 11–46.

[BG02] Dan Boneh e Philippe Golle. "Almost entirely correct mixing with applications to voting". In: *ACM Conference on Computer and Communications Security*. 2002, pp. 68–77.

[BI92] Michael Bertilsson e Ingemar Ingemarsson. "A Construction of Practical Secret Sharing Schemes using Linear Block Codes". In: *AUSCRYPT*. 1992, pp. 67–79.

[Blu82] Manuel Blum. "Coin Flipping by Telephone - A Protocol for Solving Impossible Problems". In: *COMPCON*. 1982, pp. 133–137.

[BOGW88] Michael Ben-Or, Shafi Goldwasser e Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *STOC*. 1988, pp. 1–10.

[Bri89] Ernest F. Brickell. "Some Ideal Secret Sharing Schemes". In: *EUROCRYPT*. 1989, pp. 468–475.

[BT94] Josh Cohen Benaloh e Dwight Tuinstra. "Receipt-free secret-ballot elections (extended abstract)". In: *STOC*. 1994, pp. 544–553.

[Can+04] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai e Tal Malkin. "Adaptive versus Non-Adaptive Security of Multi-Party Protocols". In: *J. Cryptology* 17.3 (2004), pp. 153–207.

[Can+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell e Philip D. MacKenzie. "Universally Composable Password-Based Key Exchange". In: *EUROCRYPT*. 2005, pp. 404–421.

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *J. Cryptology* 13.1 (2000), pp. 143–202.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS*. 2001, pp. 136–145.

[Car71] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1871.

[CC97] Lorrie Faith Cranor e Ron Cytron. "Sensus: A Security-Conscious Electronic Polling System for the Internet". In: *HICSS (3)*. 1997, pp. 561–570.

[CCD88] David Chaum, Claude Crépeau e Ivan Damgård. "Multiparty Unconditionally Secure Protocols (Extended Abstract)". In: *STOC*. 1988, pp. 11–19.

[CDN09] Ronald Cramer, Ivan Dåmgard e Jesper Buus Nielsen. *Multiparty Computation, an Introduction*. Disponible su `http://www.daimi.au.dk/~jbn/smc.pdf`. 2009.

[CGS97] Ronald Cramer, Rosario Gennaro e Berry Schoenmakers. "A Secure and Optimally Efficient Multi-Authority Election Scheme". In: *EUROCRYPT*. 1997, pp. 103–118.

[CH06] Ran Canetti e Jonathan Herzog. "Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols". In: *TCC*. 2006, pp. 380–403.

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* 24.2 (1981), pp. 84–88.

[Cha83] David Chaum. "Blind Signature System". In: *CRYPTO*. 1983, p. 153.

[CNS07] Jan Camenisch, Gregory Neven e Abhi Shelat. "Simulatable Adaptive Oblivious Transfer". In: *EUROCRYPT*. 2007, pp. 573–590.

[Cré87] Claude Crépeau. "Equivalence Between Two Flavours of Oblivious Transfers". In: *CRYPTO*. 1987, pp. 350–354.

[Csi94] László Csirmaz. "The Size of a Share Must Be Large". In: *EUROCRYPT*. 1994, pp. 13–22.

[DJN10] Ivan Damgård, Mads Jurik e Jesper Buus Nielsen. "A generalization of Paillier's public-key system with applications to electronic voting". In: *Int. J. Inf. Sec.* 9.6 (2010), pp. 371–385.

[EGL85] Shimon Even, Oded Goldreich e Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Commun. ACM* 28.6 (1985), pp. 637–647.

[Eve81] Shimon Even. "Protocol for Signing Contracts". In: *CRYPTO*. 1981, pp. 148–153.

[Fel87] Paul Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *FOCS*. 1987, pp. 427–437.

[FHM98] Matthias Fitzi, Martin Hirt e Ueli M. Maurer. “Trading Correctness for Privacy in Unconditional Multi-Party Computation (Extended Abstract)”. In: *CRYPTO*. 1998, pp. 121–136.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto e Kazuo Ohta. “A Practical Secret Voting Scheme for Large Scale Elections”. In: *AUSCRYPT*. 1992, pp. 244–251.

[Ger+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold e Mahesh Viswanathan. “The Relationship between Public Key Encryption and Oblivious Transfer”. In: *FOCS*. 2000, pp. 325–335.

[GMW87] Oded Goldreich, Silvio Micali e Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *STOC*. 1987, pp. 218–229.

[GMY06] Juan A. Garay, Philip D. MacKenzie e Ke Yang. “Strengthening Zero-Knowledge Protocols Using Signatures”. In: *J. Cryptology* 19.2 (2006), pp. 169–209.

[Gol02] Oded Goldreich. *Secure Multi-Party Computation*. Disponibile su `http://www.wisdom.weizmann.ac.il/~oded/pp.html`. 2002.

[Gol97] Shafi Goldwasser. “Multi-Party Computations: Past and Present”. In: *PODC*. 1997, pp. 1–6.

[GRR98] Rosario Gennaro, Michael O. Rabin e Tal Rabin. “Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography”. In: *PODC*. 1998, pp. 101–111.

[HM97] Martin Hirt e Ueli M. Maurer. “Complete Characterization of Adversaries Tolerable in Secure Multi-Party Computation (Extended Abstract)”. In: *PODC*. 1997, pp. 25–34.

[HS00] Martin Hirt e Kazue Sako. “Efficient Receipt-Free Voting Based on Homomorphic Encryption”. In: *EUROCRYPT*. 2000, pp. 539–556.

[IR89] Russell Impagliazzo e Steven Rudich. “Limits on the Provable Consequences of One-Way Permutations”. In: *STOC*. 1989, pp. 44–61.

[ISN93] M. Ito, A. Saio e Takao Nishizeki. “Multiple Assignment Scheme for Sharing Secret”. In: *J. Cryptology* 6.1 (1993), pp. 15–20.

[JJR02] Markus Jakobsson, Ari Juels e Ronald L. Rivest. “Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking”. In: *USENIX Security Symposium*. 2002, pp. 339–353.

[Kil88] Joe Kilian. “Founding Cryptography on Oblivious Transfer”. In: *STOC*. 1988, pp. 20–31.

[KW93] Mauricio Karchmer e Avi Wigderson. “On Span Programs”. In: *Structure in Complexity Theory Conference*. 1993, pp. 102–111.

[MKD07] Emmanouil Magkos, Panayiotis Kotzanikolaou e Christos Douligeris. "Towards secure online elections: models, primitives and open issues". In: *EG* 4.3 (2007), pp. 249–268.

[Nao91] Moni Naor. "Bit Commitment Using Pseudorandomness". In: *J. Cryptology* 4.2 (1991), pp. 151–158.

[Neu93] Peter G. Neumann. "Security Criteria for Electronic Voting". In: *Proceedings of the 6th National Computer Security Conference*. IEEE, 1993.

[NP99] Moni Naor e Benny Pinkas. "Oblivious Transfer with Adaptive Queries". In: *CRYPTO*. 1999, pp. 573–590.

[NR94] Valtteri Niemi e Ari Renvall. "How to Prevent Buying of Votes in Computer Elections". In: *ASIACRYPT*. 1994, pp. 164–170.

[Ohk+99] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka e Tatsuaki Okamoto. "An Improvement on a Practical Secret Voting Scheme". In: *ISW*. 1999, pp. 225–234.

[Ped91] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO*. 1991, pp. 129–140.

[Rab81] M. Rabin. *How to exchange secrets by oblivious transfer*. Rapp. tecn. TR-81. Harvard Aiken Computation Laboratory, 1981.

[RBO89] Tal Rabin e Michael Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: *STOC*. 1989, pp. 73–85.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[SK95] Kazue Sako e Joe Kilian. "Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth". In: *EUROCRYPT*. 1995, pp. 393–403.

[SRA81] Adi Shamir, Ronald L. Rivest e Leonard Adleman. "Mental Poker". In: *The Mathematical Gardner*. Wadsworth, 1981, pp. 37–43.

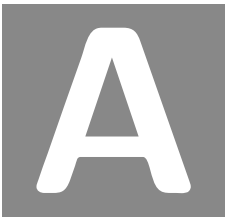
[TI05] Eleni Tsekmezoglou e John Iliadis. *A Critical View on Internet Voting Technology*. The Electronic Journal for E-Commerce Tools and Applications. Disponibile su `http://minbar.cs.dartmouth.edu/greecom/ejeta/fourth-issue/`. 2005.

[UMQ10] Dominique Unruh e Jörn Müller-Quade. "Universally Composable Incoercibility". In: *CRYPTO*. 2010, pp. 411–428.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *FOCS*. 1982, pp. 160–164.

Appendici Matematica

APPENDICE



Teoria dell'informazione

Dovresti chiamarla entropia, per due ragioni. Innanzitutto la tua funzione di incertezza è già nota in meccanica statistica con quel nome. In secondo luogo, ancora più importante, nessuno sa cosa sia con certezza l'entropia, così in una discussione avrai sarai sempre in vantaggio.

J. von Neumann, in risposta a C. Shannon

La teoria dell'informazione (*Information Theory*) è una branca della matematica applicata nata alla fine degli anni '40, grazie al lavoro di Shannon [Sha48] *Una Teoria Matematica della Comunicazione* pubblicato sulla rivista tecnica dei laboratori Bell. La ricerca di Shannon riguardava principalmente il problema della compressione dell'informazione e, più in generale, della memorizzazione e della comunicazione digitale. I suoi risultati hanno gettato le basi per una vera e propria disciplina, con applicazioni in crittografia, statistica, biologia, e molti altri settori.

Guida per il lettore. Questa appendice riassume alcuni concetti di base usati in diverse parti del testo. Nel Paragrafo A.1 richiameremo il concetto di variabile aleatoria e di distanza statistica tra variabili aleatorie. Nel Paragrafo A.2 enunceremo (e dimostreremo) alcune disuguaglianze fondamentali in teoria della probabilità, in particolare il limite di Chernoff. Infine, nel Paragrafo A.3, presenteremo una breve panoramica sui concetti di entropia e di informazione mutua.

Per un'introduzione alla teoria delle probabilità si vedano ad esempio [Tij04; Gut05], per approfondire i concetti di teoria dell'informazione si suggerisce una lettura di [CK97; CT06].

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_15, © Springer-Verlag Italia 2012

A.1 Variabili aleatorie

Una variabile aleatoria (*random variable*) può essere pensata come l'output di un esperimento del quale non si può predire con certezza il risultato. A seconda di come è descritto l'output dell'esperimento parleremo di variabili aleatorie discrete o di variabili aleatorie continue. Nel seguito ci soffermiamo solo sulle prime. La nostra trattazione sarà principalmente informale; si raccomanda [Bil79] per un approccio rigoroso.

Variabili aleatorie discrete. Essenzialmente una variabile aleatoria discreta X mappa un evento $\mathcal{E}$ in un valore di un insieme numerabile $\mathcal{X}$ (ad esempio l'insieme dei numeri interi); ciascun valore ha associata una quantità — detta *probabilità* — maggiore o uguale di zero e minore o uguale di 1. In questo modo, si associa ad ogni variabile aleatoria discreta una distribuzione di probabilità, detta anche densità discreta o funzione di massa di probabilità.

Consideriamo, ad esempio, l'evento $\mathcal{E}$ che descrive il lancio di un dado (non truccato e con sei facce). Dato l'insieme $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$, la variabile aleatoria X associa ad ogni possibile risultato del lancio del dado un elemento dell'insieme $\mathcal{X}$:

$$X = \begin{cases} 1 & \text{se esce } \boxdot, \\ 2 & \text{se esce } \boxdot, \\ 3 & \text{se esce } \boxdot, \\ 4 & \text{se esce } \boxdot, \\ 5 & \text{se esce } \boxdot, \\ 6 & \text{se esce } \boxdot. \end{cases}$$

La densità discreta sarà quindi

$$P_X(x) = \begin{cases} 1/6 & \text{se } x = 1, \ldots, 6 \\ 0 & \text{altrimenti.} \end{cases}$$

Più in generale, data una variabile aleatoria aleatoria X definita su un'insieme numerabile $\mathcal{X}$, la sua densità discreta nel punto $x \in \mathcal{X}$ è la probabilità che X assuma il valore x. In simboli:

$$P_X(x) = \mathbb{P}\left[X = x\right] \quad \forall x \in \mathcal{X}.$$

Siccome la somma delle probabilità di tutti gli eventi è l'evento certo, si ha sempre $\sum_{x \in \mathcal{X}} P_X(x) = 1$.

Indipendenza statistica. Quando si ha a che fare con due variabili aleatorie contemporaneamente, in genere non è sufficiente conoscere i valori di probabilità delle singole variabili. Intuitivamente il motivo è che le due variabili aleatorie possono essere in qualche modo "correlate". In questo senso, la coppia $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ è essa stessa una variabile aleatoria caratterizzata dalla densità discreta (detta *congiunta*):

$$P_{X,Y}(X, Y) = \mathbb{P}[X = x \wedge Y = y].$$

Le densità discrete delle singole variabili aleatorie componenti, cosiddette *marginali*, si ottengono per saturazione della congiunta:

$$P_X(x) = \sum_{y \in \mathcal{Y}} P_{X,Y}(x, y) \tag{A.1}$$

$$P_Y(y) = \sum_{x \in \mathcal{X}} P_{X,Y}(x, y). \tag{A.2}$$

Possiamo esprimere il rapporto tra densità congiunta e marginale attraverso il concetto di *densità discreta condizionata*. La probabilità dell'evento $\mathcal{E}_1$ condizionata all'evento $\mathcal{E}_2$ è la probabilità che l'evento $\mathcal{E}_1$ si verifichi condizionatamente al fatto che si verifichi $\mathcal{E}_2$. Ad esempio, tornando all'esempio del dado, la probabilità che esca ⚄ sapendo che il risultato del lancio sarà ⚀, ⚄ oppure ⚃ è 1/3. Formalmente, abbiamo:

$$\mathbb{P}[\mathcal{E}_1 \mid \mathcal{E}_2] = \frac{\mathbb{P}[\mathcal{E}_1 \wedge \mathcal{E}_2]}{\mathbb{P}[\mathcal{E}_2]}.$$

Due eventi si dicono *statisticamente indipendenti* quando $\mathbb{P}[\mathcal{E}_1 \mid \mathcal{E}_2] = \mathbb{P}[\mathcal{E}_1]$ (ovvero quando la probabilità che $\mathcal{E}_1$ si verifichi dato che si verifica $\mathcal{E}_2$ è uguale alla probabilità che $\mathcal{E}_1$ si verifichi a priori). La relazione tra $\mathbb{P}[\mathcal{E}_1 \mid \mathcal{E}_2]$ e $\mathbb{P}[\mathcal{E}_2 \mid \mathcal{E}_1]$ è data dal teorema di Bayes, espresso dalla seguente equazione:

$$\mathbb{P}[\mathcal{E}_2 \mid \mathcal{E}_1] = \mathbb{P}[\mathcal{E}_1 \mid \mathcal{E}_2] \cdot \frac{\mathbb{P}[\mathcal{E}_2]}{\mathbb{P}[\mathcal{E}_1]}. \tag{A.3}$$

Passando alle densità, date due variabile aleatorie $X \in \mathcal{X}$ ed $Y \in \mathcal{Y}$, la probabilità condizionata di $Y = y$ dato $X = x$ è:

$$\mathbb{P}[Y = y \mid X = x] = \frac{\mathbb{P}[X = x \wedge Y = y]}{\mathbb{P}[X = x]} \overset{(A.3)}{=} \frac{\mathbb{P}[Y = y | X = x]\,\mathbb{P}[Y = y]}{\mathbb{P}[X = x]}.$$

Quindi,

$$P_{X,Y}(x,y) = P_{Y|X}(y|x)P_X(x) = P_{X|Y}(x|y)P_Y(y).$$

Analogamente a quanto visto per gli eventi, due variabile aleatorie si dicono statisticamente indipendenti se $P_{X,Y}(x,y) = P_X(x)P_Y(y)$.

Valore atteso e varianza. La densità discreta di una variabile aleatoria X è caratterizzata da diversi parametri. In alcuni casi può essere sufficiente conoscere quale valore assume la variabile aleatoria in media; si definisce quindi il *valore atteso* di X come:

$$\mu = \mathbb{E}[X] = \sum_{x \in \mathcal{X}} x P_X(x).$$

Non è difficile mostrare che il valore atteso è un operatore lineare: il valore atteso della somma di più variabili aleatorie altro non è che la somma dei singoli valori attesi. Riportiamo il caso di due variabili aleatorie; l'estensione al caso generale è un semplice esercizio (cf. Esercizio A.2).

Lemma A.1 (Linearità del valore atteso). *Siano X ed Y due variabile aleatorie discrete definite (rispettivamente) su due insiemi $\mathcal{X}$ ed $\mathcal{Y}$. Per ogni $\alpha, \beta \in \mathbb{R}$ abbiamo $\mathbb{E}[\alpha X + \beta Y] = \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y]$.*

Dimostrazione. La dimostrazione segue direttamente dalla definizione di densità congiunta:

$$\begin{aligned}
\mathbb{E}[\alpha X + \beta Y] &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (\alpha x + \beta y) \mathbb{P}[X = x \wedge Y = y] \\
&= \alpha \sum_{x \in \mathcal{X}} x \sum_{y \in \mathcal{Y}} \mathbb{P}[X = x \wedge Y = y] + \beta \sum_{y \in \mathcal{Y}} y \sum_{x \in \mathcal{X}} \mathbb{P}[X = x \wedge Y = y] \\
&= [\text{usando l'Eq. (A.1) e l'Eq. (A.2)}] \\
&= \alpha \sum_{x \in \mathcal{X}} x \cdot \mathbb{P}[X = x] + \beta \sum_{y \in \mathcal{Y}} y \cdot \mathbb{P}[Y = y] \\
&= \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y].
\end{aligned}$$

□

Anche quando è noto il valore medio, è lecito chiedersi quanto i valori di X (tipicamente) si allontanano da $\mathbb{E}[X]$; a tale scopo si definisce la *varianza* di una variabile aleatoria come:

$$\sigma^2 = \mathbb{V}[X] = \mathbb{E}\left[(X - \mu)^2\right].$$

La quantità $\sigma = \sqrt{\sigma^2}$ è detta *deviazione standard*. Un semplice calcolo mostra:

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}\left[(X-\mu)^2\right] = \mathbb{E}\left[X^2 - 2\mu X + \mu^2\right] \\ &= [\text{per la linearità del valore atteso (cf. Lemma A.1)}] \\ &= \mathbb{E}\left[X^2\right] - 2\mu^2 + \mu^2 \\ &= \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2.\end{aligned}$$

La varianza (in generale) non è lineare. In alcuni casi particolari però la proprietà di linearità è soddisfatta. Di particolare interesse è il caso di variabili aleatorie *indipendenti a coppie*. Diremo che n variabili aleatorie $X_1, \ldots, X_n$ definite sull'insieme $\mathcal{X}$ sono indipendenti a coppie, se per ogni $i, j \in [n]$ (con $i \neq j$) e per ogni $a, b \in \mathbb{R}$, risulta:

$$\mathbb{P}[X_i = a \wedge X_j = b] = \mathbb{P}[X_i = a] \cdot \mathbb{P}[X_j = b].$$

Nel caso in cui l'indipendenza a coppie è soddisfatta la varianza della variabile aleatoria somma coincide con la somma delle singole varianze:

Lemma A.2 (Linearità della varianza). *Siano $X_1, \ldots, X_n$ variabili aleatorie indipendenti a coppie; poniamo $X = \sum_{i=1}^{n} X_i$. Allora:*

$$\mathbb{V}[X] = \sum_{i=1}^{n} \mathbb{V}[X_i].$$

Dimostrazione. Basta applicare la definizione e sfruttare la proprietà di linearità del valore atteso:

$$\begin{aligned}\mathbb{V}[X] &= \mathbb{E}\left[(X_1 + \ldots + X_n)^2\right] - (\mathbb{E}[X_1 + \ldots + X_n])^2 \\ &= \mathbb{E}[(X_1 + \ldots + X_n) \cdot (X_1 + \ldots + X_n)] - (\mathbb{E}[X_1 + \ldots + X_n])^2 \\ &= \mathbb{E}\left[\sum_{i,j} X_i X_j\right] - \sum_{i,j} \mathbb{E}[X_i] \cdot \mathbb{E}[X_j] \\ &= [\text{dalla linearità del valore atteso}] \\ &= \sum_{i,j} \mathbb{E}[X_i \cdot X_j] - \sum_{i,j} \mathbb{E}[X_i] \cdot \mathbb{E}[X_j] \\ &= \sum_{i} \mathbb{E}\left[X_i^2\right] + \sum_{i \neq j} \mathbb{E}[X_i \cdot X_j] - \sum_{i} (\mathbb{E}[X_i])^2 - \sum_{i \neq j} \mathbb{E}[X_i] \cdot \mathbb{E}[X_j]\end{aligned}$$

$$= \sum_i \left(\mathbb{E}\left[X_i^2\right] - (\mathbb{E}\left[X_i\right])^2 \right) + \sum_{i \neq j} \left(\mathbb{E}\left[X_i \cdot X_j\right] - \mathbb{E}\left[X_i\right] \cdot \mathbb{E}\left[X_j\right] \right)$$
$$= \sum_i \mathbb{V}\left[X_i\right] + \sum_{i \neq j} \left(\mathbb{E}\left[X_i \cdot X_j\right] - \mathbb{E}\left[X_i\right] \cdot \mathbb{E}\left[X_j\right] \right).$$

Siccome le variabili aleatorie $X_1, \ldots, X_n$ sono indipendenti a coppie, l'ultima sommatoria è nulla. □

Spazi metrici e distanza statistica. Dato uno spazio, possiamo dotarlo di una struttura arricchendolo con una *metrica* (detta anche *distanza*). In questo modo gli elementi dello spazio sono luoghi geometrici che (quando non identici) hanno distanza positiva.

Definizione A.1 (Metrica su uno spazio $\mathcal{X}$). Si definisce metrica una funzione $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ che soddisfa le seguenti proprietà:

- *Definita positiva.* $d(x, y) \geq 0$, $\forall\ x, y \in \mathcal{X}$ e $d(x, y) = 0$ se e solo se $x = y$.
- *Simmetria.* $d(x, y) = d(y, x)$, $\forall\ x, y \in \mathcal{X}$.
- *Disuguaglianza triangolare.* $d(x, z) \leq d(x, y) + d(y, z)$, $\forall\ x, y, z \in \mathcal{X}$.

■

In genere la scelta non è univoca, ed infatti esistono tante metriche. Un esempio classico è costituito dallo spazio delle stringhe binarie a lunghezza n (detto anche *spazio di Hamming* ed indicato con $\{0,1\}^n$), dove si definisce la distanza di Hamming.

Definizione A.2 (Distanza di Hamming). Date due stringhe binarie $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ la loro distanza di Hamming $d_H : \{0,1\}^n \times \{0,1\}^n \to \mathbb{R}^+$ è definita come:

$$d_H(\mathbf{x}, \mathbf{y}) = \#\left\{i :\ x_i \neq y_i, \forall\ i = 1, 2, \ldots, n\right\}.$$

■

Non è complesso verificare che la distanza di Hamming è una metrica. In effetti ciò deriva dal fatto che

$$d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|;$$

pertanto, siccome la funzione valore assoluto è una metrica, anche la distanza di Hamming lo è. Si definisce il *peso di Hamming* di una stringa binaria $\mathbf{x} \in \{0,1\}^n$ come il numero di valori "1" in $\mathbf{x}$. Più precisamente $w_H(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0})$, dove $\mathbf{0} = (0, \ldots, 0)$ è il vettore tutto nullo in $\{0,1\}^n$.

Un secondo esempio è quello dello spazio euclideo $\mathbb{R}^n$. Dato un vettore $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, sia

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

la sua norma l-1. È immediato verificare che la mappa $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1$ è una metrica. Analogamente, se consideriamo la norma l-2:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}, \tag{A.4}$$

è banale verificare che la mappa $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ è una metrica.[87] (Questa non è altro che la familiare distanza euclidea definita dal teorema di Pitagora nel piano Cartesiano.) Osserviamo che, per ogni vettore $\mathbf{x} \in \mathbb{R}^n$, risulta:

$$\|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2. \tag{A.5}$$

Generalizzando questi esempi, è possibile definire una distanza tra distribuzioni di probabilità discrete. Tale distanza è detta *distanza statistica*.

Definizione A.3 (Distanza statistica). Date due variabili aleatorie X_0, X_1 con valori in $\mathcal{X}$, la loro distanza statistica è:

$$\Delta(X_0, X_1) = \frac{1}{2} \sum_{x \in \mathcal{X}} \left| \mathbb{P}[X_0 = x] - \mathbb{P}[X_1 = x] \right|.$$

(Notare che $\Delta(X_0, X_1) = 1/2\|X_0 - X_1\|_1$.) ■

Sia U_n la distribuzione uniforme su $\{0,1\}^n$. In genere, quando X assume valori in $\mathcal{X} = \{0,1\}^n$, si è soliti indicare con $d(X) = \Delta(X, U_n)$ la distanza

[87] Più in generale, per ogni $p > 0$, data la norma l-p:

$$\|\mathbf{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p},$$

la mappa $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$ è una metrica.

statistica tra X e la distribuzione uniforme su $\{0,1\}^n$. Quando $d(X) \leq \epsilon$ si dice che X è ϵ-vicina alla distribuzione uniforme. Analogamente:

$$\Delta(X_0, X_1|Y) = \Delta((X_0, Y), (X_1, Y))$$
$$d(X|Y) = \Delta((X, Y), (U_n, Y)).$$

Si può dimostrare che la distanza statistica è una metrica (cf. Definizione A.1).

A.2 Alcune disuguaglianze

In questo paragrafo studieremo alcune disuguaglianze molto importanti in teoria della probabilità. La seguente disuguaglianza di Markov collega probabilità e valori attesi.

Lemma A.3 (Disuguaglianza di Markov). *Ogni variabile aleatoria X non negativa soddisfa:*

$$\mathbb{P}[X \geq n] \leq \frac{\mathbb{E}[X]}{n}.$$

Dimostrazione. Possiamo scrivere:

$$\mathbb{E}[X] = \sum_{x \geq 0} x \cdot \mathbb{P}[X = x] \geq \sum_{0 \leq x < n} \mathbb{P}[X = x] \cdot 0 + \sum_{x \geq n} \mathbb{P}[X = x] \cdot n$$
$$= \mathbb{P}[X \geq n] \cdot n.$$

□

La disuguaglianza di Chebyshev garantisce che, per ogni variabile aleatoria, quasi tutti i valori sono "vicini alla media".

Lemma A.4 (Disuguaglianza di Chebyshev). *Sia X una variabile aleatoria con varianza σ^2. Per ogni $n > 0$, risulta:*

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq n] \leq \frac{\sigma^2}{n^2}.$$

Dimostrazione. Definiamo la variabile aleatoria non negativa $Y = (X - \mathbb{E}[X])^2$ ed applichiamo la disuguaglianza di Markov ad Y:

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq n] \leq \mathbb{P}\left[(X - \mathbb{E}[X])^2 \geq n^2\right] = \mathbb{P}\left[Y \geq n^2\right]$$
$$\leq \frac{\mathbb{E}[Y]}{n^2} = \frac{\mathbb{E}\left[(X - \mathbb{E}[X])^2\right]}{n^2} = \frac{\sigma^2}{n^2}.$$

□

La stima fornita dalla disequazione di Chebyshev è in un certo senso grossolana. Il seguente teorema di Chernoff, consente una caratterizzazione più fine (cf. Esercizio A.6).

Teorema A.5 (Limite di Chernoff). *Siano $X_1, X_2, \ldots, X_n$, variabili aleatorie mutuamente indipendenti su $\{0,1\}$ e sia $\mu = \sum_{i=1}^{n} \mathbb{E}[X_i]$. Allora:*

(I) $\forall\ \delta > 0$,

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu};$$

(II) $\forall\ 0 < \delta \leq 1$,

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq e^{-\mu\delta^2/3};$$

(III) $\forall\ c \geq 6\mu$,

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \geq c\right] \leq 2^{-c}.$$

Dimostrazione. Definiamo $X = \sum_{i=1}^{n} X_i$. Posto $p_i = \mathbb{P}[X_i = 1]$, si ha $\mu = \sum_{i=1}^{n} p_i$. Introduciamo un parametro t il cui ruolo sarà noto a breve; per ogni $t > 0$, possiamo scrivere:

$$\mathbb{P}[X > (1+\delta)\mu] = \mathbb{P}\left[e^{tX} > e^{t(1+\delta)\mu}\right].$$

Applicando la disuguaglianza di Markov al secondo membro, otteniamo:

$$\mathbb{P}[X > (1+\delta)\mu] \leq \frac{\mathbb{E}\left[e^{tX}\right]}{e^{t(1+\delta)\mu}}. \tag{A.6}$$

Notare che, siccome le variabili aleatorie X_i sono mutuamente indipendenti, abbiamo:

$$\mathbb{E}\left[e^{tX}\right] = \mathbb{E}\left[e^{t\sum_{i=1}^{n} X_i}\right] = \mathbb{E}\left[\prod_{i=1}^{n} e^{tX_i}\right] = \prod_{i=1}^{n} \mathbb{E}\left[e^{tX_i}\right]. \tag{A.7}$$

Possiamo inoltre limitare superiormente il termine e^{tX_i} sfruttando il fatto che $e^x > 1 + x$ e che $X_i \in \{0,1\}$:

$$\mathbb{E}\left[e^{tX_i}\right] = p_i e^t + (1 - p_i) = 1 + (e^t - 1)p_i \leq e^{(e^t-1)p_i}.$$

Ma allora, sostituendo nell'Eq. (A.7), si ottiene:

$$\mathbb{E}\left[e^{tX}\right] < \prod_{i=1}^{n} e^{(e^t-1)p_i} = e^{\sum_{i=1}^{n}(e^t-1)p_i} = e^{(e^t-1)\mu}.$$

Sostituendo quest'ultima espressione nell'Eq. (A.6), infine, otteniamo:

$$\mathbb{P}\left[X > (1+\delta)\mu\right] \leq \frac{\mathbb{E}\left[e^{tX}\right]}{e^{t(1+\delta)\mu}} < \frac{e^{(e^t-1)\mu}}{e^{t(1+\delta)\mu}} = \left(e^{e^t-1-t(1+\delta)}\right)^{\mu}.$$

L'ultima espressione è valida per ogni $t > 0$. Per ottenere l'asserto non ci resta che trovare il t ottimo, che rende la disuguaglianza stretta: vogliamo cioè minimizzare $e^t - 1 - t(1+\delta)$. Posto

$$\frac{d}{dt}(e^t - 1 - t(1+\delta)) = e^t - 1 - \delta \stackrel{!}{=} 0,$$

si ottiene la soluzione unica $t = \ln(1+\delta)$. Quindi:

$$\mathbb{P}\left[X > (1+\delta)\mu\right] < \left(e^{\delta-(1+\delta)\ln(1+\delta)}\right)^{\mu} = \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu},$$

che è esattamente l'espressione in (I).

Per dimostrare la (II) scriviamo l'espansione in serie di Mclaurin[88] del termine $\ln(1+\delta)$, ovvero:

$$(1+\delta)\ln(1+\delta) = (1+\delta)\sum_{i\geq 1}(-1)^{i+1}\frac{\delta^i}{i} = \delta + \sum_{i\geq 2}(-1)^i\delta^i\left(\frac{1}{i-1} - \frac{1}{i}\right).$$

[88] In analisi matematica, un risultato di Taylor consente di approssimare una funzione (differenziabile) attorno ad un dato punto attraverso alcuni polinomi, detti polinomi di Taylor, i cui coefficienti dipendono solo dalle derivate della funzione nel punto. In particolare sia $f : [a,b] \to \mathbb{R}^n$ una funzione differenziabile n volte nell'intervallo $[a,b]$ e sia $x_0 \in [a,b]$. Allora:

$$f(x) = \sum_{i=0}^{\infty}\frac{f^{(i)}(x_0)}{i!}(x-x_0)^i,$$

dove con $f^{(i)}(x_0)$ si intende la derivata i-sima di f calcolata nel punto x_0. Quando $x_0 = 0$ si parla anche di *serie di Mclaurin*. Un semplice calcolo mostra:

$$\ln(1+x) = \sum_{i=1}^{\infty}(-1)^{i+1}\frac{x^i}{i} \qquad \text{quando } -1 < x \leq 1.$$

Se $0 \leq \delta < 1$ possiamo ignorare i termini di ordine superiore, ottenendo:

$$(1+\delta)\ln(1+\delta) > \delta + \frac{\delta^2}{2} - \frac{\delta^3}{6} \geq \delta - \frac{\delta^2}{3};$$

quindi

$$\mathbb{P}\left[X > (1+\delta)\mu\right] < \left(e^{\delta-(1+\delta)\ln(1+\delta)}\right)^{\mu} \leq e^{-\frac{\delta^2\mu}{3}} \qquad (0 \leq \delta < 1),$$

come desiderato.

Per quanto riguarda la (III), infine, sia $c = (1+\delta)\mu$. Quando $c \geq 6\mu$ abbiamo $\delta = c/\mu - 1 \geq 5$, quindi usando quanto dimostrato in (I):

$$\begin{aligned}\mathbb{P}\left[X \geq c\right] &\leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \\ &\leq \left(\frac{e}{1+\delta}\right)^{(1+\delta)\mu} \\ &\leq \left(\frac{e}{6}\right)^{c} \leq 2^{-c}.\end{aligned}$$

□

Si possono considerare alcune varianti, discusse brevemente di seguito. Un procedimento simile a quello usato nella dimostrazione del Teorema A.5 mostra la validità delle seguenti disuguaglianze:

(I') $\forall\ 0 < \delta < 1$,

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \leq (1-\delta)\mu\right] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu};$$

(II') $\forall\ 0 < \delta < 1$,

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i \leq (1-\delta)\mu\right] \leq e^{-\mu\delta^2/2}.$$

In particolare, la seconda disuguaglianza afferma che la probabilità che l'output di n esperimenti indipendenti sia δ-lontano dal valore atteso (per $0 \leq \delta < 1$), è dell'ordine di $e^{-\Omega(n\delta^2)}$.

A.3 Entropia

Il termine *entropia* fa riferimento all'uso che ne ha fatto per primo Shannon in [Sha48]. Data una variabile aleatoria $X \in \mathcal{X}$, definiamo la sua entropia (detta appunto di Shannon) come:

$$H(X) = -\sum_{x \in \mathcal{X}} P_X(x) \log P_X(x).$$

Intuitivamente, l'entropia di una variabile aleatoria può essere vista come una *misura del contenuto informativo* associato alla variabile aleatoria stessa; in altri termini la quantità $H(X)$ rappresenta la quantità d'informazione contenuta in X.

Consideriamo ad esempio un lancio di moneta, non necessariamente bilanciata, dove sono note le probabilità che esca "testa" oppure "croce". L'entropia dell'esito del prossimo lancio è massima se la moneta non è truccata, cioè quando "testa" e "croce" hanno la stessa probabilità (pari a 1/2) di verificarsi. Questa è infatti la massima situazione di incertezza, in quanto è più difficile prevedere se uscirà "testa" oppure "croce". D'altra parte, quando la moneta è truccata uno degli esiti avrà una probabilità maggiore di verificarsi e quindi c'è meno incertezza, che si riflette in una minore entropia. Il caso estremo è quello in cui la moneta ha lo stesso simbolo su entrambe le facce: in questo caso non c'è incertezza e l'entropia è zero (cf. Fig. A.1).

L'estensione della definizione di entropia al caso di due variabili aleatorie è pressoché immediata:

$$H(X,Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P_{X,Y}(x,y) \log P_{X,Y}(x,y).$$

Intuitivamente saremmo portati a dire che la quantità d'informazione associata ad X ed Y (congiuntamente) è sempre maggiore della quantità d'informazione associata alla sola X. In effetti, applicando le definizione, non è complesso verificare che $H(X,Y) \geq H(X)$ per ogni coppia di variabili aleatorie $X \in \mathcal{X}$ ed $Y \in \mathcal{Y}$ (cf. Esercizio A.9). Se alla quantità $H(X,Y)$ sottraiamo il contenuto informativo associato ad X otteniamo la quantità d'informazione residua in Y "dopo aver visto X". Tale quantità è detta *entropia condizionata* di Y dato X ed è definita come $H(Y|X) = H(X,Y) - H(X)$. Seguendo il significato intuitivo di entropia, date n variabili aleatorie $X_1, \ldots, X_n$ a valori in $\mathcal{X}$, saremmo portati a pensare che la loro entropia congiunta è esattamente il contenuto d'informazione in X_1 più il contributo di X_2 dopo aver visto X_1 (cioè $H(X_2|X_1)$), più il contributo di X_3 dopo aver visto X_1, X_2 (cioè $H(X_3|X_1, X_2)$), e così via. In effetti

si dimostra che, per ogni scelta delle variabili aleatorie $X_1, \ldots, X_n$, si ha esattamente $H(X_1, \ldots, X_n) = H(X_1) + \sum_{i=2}^{n} H(X_i|X_1, \ldots, X_{i-1})$. Quest'ultima uguaglianza è detta regola della catena (*chain rule*, cf. Esercizio A.10).

Una metafora per l'entropia. Possiamo rendere più esplicita la metafora di "entropia come contenuto informativo associato ad una variabile aleatoria" attraverso il concetto di misura additiva su insiemi. Sia $\mu(\mathcal{X})$ una misura associata all'insieme $\mathcal{X}$ (ad esempio la sua cardinalità). Associamo $\mu(\mathcal{X})$ all'entropia $H(X)$ (cf. anche Fig. A.2). Dati due insiemi $\mathcal{X}$ ed $\mathcal{Y}$ la misura dell'unione non è altro che la misura di $\mathcal{X}$ più la misura di $\mathcal{Y} \setminus \mathcal{X}$, ovvero $\mu(\mathcal{X} \cup \mathcal{Y}) = \mu(\mathcal{X}) + \mu(\mathcal{Y} \setminus \mathcal{X})$. Allo stesso modo $H(X, Y) = H(X) + H(Y|X)$. Continuando a seguire questa metafora, ci aspettiamo che la misura associata ad $\mathcal{Y}$ sia sempre maggiore di quella associata ad $\mathcal{Y} \setminus \mathcal{X}$, ovvero $\mu(\mathcal{Y}) \geq \mu(\mathcal{Y} \setminus \mathcal{X})$. In effetti, si può dimostrare che $H(Y) - H(Y|X) \geq 0$ ed in particolare l'uguaglianza è valida se e solo se le variabili aleatorie X ed Y sono statisticamente indipendenti. La quantità:

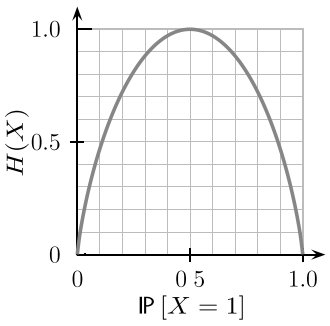


Fig. A.1. Entropia $H(X)$ di un lancio di moneta (misurata in bit) al variare del bilanciamento della moneta

$$I(X \wedge Y) = H(Y) - H(Y|X) = H(X) - H(X|Y) \geq 0,$$

è detta *informazione mutua* delle variabili aleatorie X ed Y e nella nostra metafora rappresenta la misura dell'intersezione $\mu(\mathcal{X} \cap \mathcal{Y})$.[89]

Mettendo insieme quest'ultima disuguaglianza con le definizioni precedenti,

[89] Sebbene questa metafora aiuti a visualizzare molti concetti, bisogna utilizzarla con accortezza. Ecco un controesempio. Si definisce *informazione mutua condizionata* la quantità $I(X \wedge Y|Z) = H(X|Z) + H(Y|X) - H(X, Y|Z)$. La nostra metafora ci porta a dire che $I(X \wedge Y) \geq I(X \wedge Y|Z)$, perché tale quantità corrisponde alla misura dell'intersezione $\mu(\mathcal{X} \cap \mathcal{Y} \cap \mathcal{Z})$. Purtroppo ciò non sempre è verificato, in particolare esistono casi in cui $I(X \wedge Y) \geq I(X \wedge Y|Z)$ ed altri casi in cui $I(X \wedge Y) < I(X \wedge Y|Z)$ (cf. Esercizio A.11).

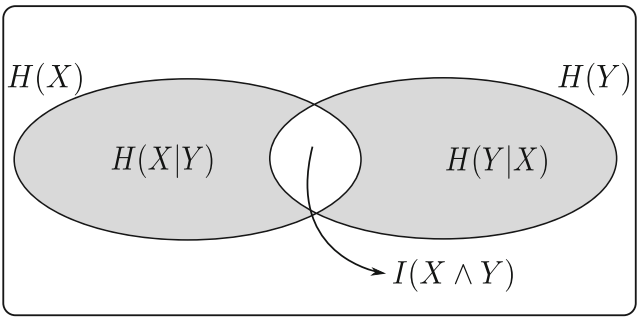


Fig. A.2. Entropia e misura di insiemi

troviamo:

$$\begin{aligned} I(X \wedge Y) &= H(Y) - H(Y|X) \\ &= H(Y) - (H(X,Y) - H(X)) \\ &= H(X) + H(Y) - H(X,Y) \\ &\geq 0, \end{aligned}$$

ed il segno uguale si verifica quando e solo quando X ed Y sono statisticamente indipendenti. Ciò significa che l'informazione associata ad X ed Y (se correlate), è maggiore della somma dei contributi informativi associati ad X ed Y prese singolarmente.

Entropia minima. L'entropia di Rényi [Rén61], è una generalizzazione dell'entropia di Shannon, utile a quantificare diversi gradi di incertezza associati ad una variabile aleatoria generica.

Definizione A.4 (Entropia di Rényi). Sia X una variabile aleatoria su $\mathcal{X} = \{x_1, \ldots, x_n\}$. L'entropia di Rényi di ordine α, con $\alpha \geq 0$, è definita come:

$$\mathbf{H}_\alpha(X) = \frac{1}{1-\alpha} \log \left(\sum_{i=1}^{n} p_i^\alpha \right),$$

essendo $p_i = \mathbb{P}[X = x_i]$. ■

Alcuni casi particolari sono riportati di seguito.

- $\alpha = 0$. Abbiamo $\mathbf{H}_0(X) = \log n = \log |\mathcal{X}|$, che è il logaritmo della cardinalità di $\mathcal{X}$, a volte detto *entropia di Hartley*.
- $\alpha \to 1$. Nel limite per α che tende ad 1, abbiamo:

$$\mathbf{H}_1(X) = \lim_{\alpha \to 1} \mathbf{H}_\alpha(X) = -\sum_{i=1}^{n} p_i \log p_i,$$

  che è l'entropia di Shannon $H(X)$.
- $\alpha \to \infty$. Nel limite per $\alpha \to \infty$ si parla di entropia minima (*min-entropy*):

$$\mathbf{H}_\infty(X) = \lim_{\alpha \to \infty} \mathbf{H}_\alpha(X) = -\log \sup_{i=1,\ldots,n} p_i = -\log \max_{x \in \mathcal{X}} \mathbb{P}\left[X = x\right],$$

  così detta perché è il valore più piccolo di $\mathbf{H}_\alpha$.

L'entropia minima misura la quantità di randomicità in X. Infatti la quantità $\max_{x \in \mathcal{X}} \mathbb{P}\left[X = x\right] = 2^{-\mathbf{H}_\infty(X)}$ rappresenta la massima probabilità di predire correttamente X; in questo senso, l'entropia minima misura quanto è difficile predire X.

Esercizi

Esercizio A.1. Sia $\mathscr{A}$ un algoritmo che prende come input una stringa binaria $x \in \{0,1\}^n$ e restituisce 1 se e solo se i primi 3 bit di x sono 1. Sia $(\mathsf{Enc}, \mathsf{Dec})$ il cifrario OTP del Crittosistema 2.1. Calcolare:

$$\mathbb{P}\left[\mathscr{A}(c) = 1 : \ k \xleftarrow{\$} \{0,1\}^n, c \leftarrow \mathsf{Enc}_k(1^n)\right].$$

Esercizio A.2. Estendere il Lemma A.1 al caso di $n > 2$ variabili aleatorie.

Esercizio A.3. La legge debole dei grandi numeri dice che se $X_1, X_2, X_3, \ldots$ sono variabili aleatorie indipendenti ed identicamente distribuite, con valore atteso μ e deviazione standard σ, allora per ogni $\epsilon > 0$ si ha:

$$\lim_{n\to\infty} \mathbb{P}\left[\left|\frac{X_1 + X_2 + \ldots + X_n}{n} - \mu\right| > \epsilon\right] = 0.$$

Dimostrare la legge debole dei grandi numeri, utilizzando il seguente schema:

1. Sia $X^* = \frac{X_1+X_2+\ldots+X_n}{n}$. Calcolare $\mu^* = \mathbb{E}[X]$.
2. Calcolare $\sigma^* = \mathbb{V}[X]$ e mostrare che per ogni costante $\epsilon > 0$ si ha $\epsilon \geq \sigma/n = \sqrt{n}\sigma^*$ per n sufficientemente grande.
3. Applicare la disuguaglianza di Chebyshev usando i valori determinati ai passi precedenti.

Esercizio A.4. Consideriamo n variabili aleatorie Bernoulliane $X_i \xleftarrow{\$} \mathsf{Ber}_\tau$, ovvero $\mathbb{P}[X_i = 1] = \tau$ per ogni $i \in [n]$. Determinare la probabilità che $\sum_{i=1}^n X_i > n \cdot \tau'$, per una costante $\tau < \tau' < n$.

Esercizio A.5. Consideriamo n variabili aleatorie $X_i \xleftarrow{\$} \{0,1\}$. Determinare la probabilità che $\sum_{i=1}^n X_i \leq n \cdot \tau'$, per una costante $\tau < \tau' < 1/2$.

Esercizio A.6. Sia X la variabile aleatoria che rappresenta il numero di valori "testa" in n lanci di moneta indipendenti. Applicare la disuguaglianza di Chebyshev per limitare la probabilità che X sia più piccola di $n/4$ e più grande di $3n/4$. Confrontare il risultato con lo stesso limite ottenuto applicando il teorema di Chernoff.

Esercizio A.7. Una moneta non truccata è lanciata fintanto che non si ottiene "testa". Sia X il numero di lanci; determinare $H(X)$.

Esercizio A.8. Consideriamo le variabili aleatorie binarie X, Y, con distribuzione di probabilità congiunta determinata dalle equazioni:

$$P_{XY}(0,0) = 1/6 \qquad P_{XY}(0,1) = 1/6$$
$$P_{XY}(1,0) = 1/2 \qquad P_{XY}(1,1) = 1/6.$$

Calcolare $H(X), H(Y), H(X|Y), H(Y|X)$, $H(X,Y)$, $H(Y) - H(Y|X)$ ed $I(X \wedge Y)$.

Esercizio A.9. Dimostrare che per ogni coppia di variabili aleatorie X, Y, definite su due insiemi $\mathcal{X}$, $\mathcal{Y}$, si ha $H(X,Y) \geq H(X)$.
(Suggerimento: applicare la definizione, quindi usare la regola di Bayes.)

Esercizio A.10. Dimostrare la regola della catena, ovvero:

$$H(X_1, \ldots, X_n) = H(X_1) + \sum_{i=2}^{n} H(X_i | X_1, \ldots, X_{i+1}).$$

(Suggerimento: usare ricorsivamente l'uguaglianza $H(A,B) = H(A) + H(B|A)$, cominciando dal caso $A = (X_1, \ldots, X_{n-1})$, $B = X_n$.)

Esercizio A.11. Questo esercizio mostra che non è possibile stabilire un ordinamento generale per le quantità $I(X \wedge Y)$ ed $I(X \wedge Y | Z)$.

1. Sia $X = Y = Z$, con $H(X) > 0$. Mostrare che $I(X \wedge Y) - I(X \wedge Y | Z) > 0$.
2. Siano X, Y variabili aleatorie binarie indipendenti ed uniformemente distribuite, e $Z = X \oplus Y$. Mostrare che $I(X \wedge Y) - I(X \wedge Y | Z) < 0$.

Esercizio A.12. Sia X una variabile aleatoria su un insieme $\mathcal{X}$. Mostrare che per ogni scelta della distribuzione di probabilità P_X, risulta $\mathbf{H}_\infty(X) \leq \log(\#\mathcal{X})$.

Letture consigliate

[Bil79] Patrick Billingsley. *Probability and Measure*. John Wiley e Sons, 1979.

[CK97] Imre Csiszár e János Körner. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Second. Akademiai Kiado, 1997.

[CT06] Thomas M. Cover e Joy A. Thomas. *Elements of Information Theory*. Second. New York: Wiley-Interscience, 2006.

[Gut05] Allan Gut. *Probability: A Graduate Course*. Springer-Verlang, 2005.

[Rén61] Alfréd Rényi. "On Measures of Information and Entropy". In: *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability*. 1961, pp. 547–561.

[Sha48] Claude E. Shannon. "A mathematical theory of communication". In: *Bell Sys. Tech. J.* 27 (1948), pp. 623–656.

[Tij04] Henk Tijms. *Understanding Probability: Chance Rules in Everyday Life*. Cambridge University Press, 2004.

APPENDICE B

Teoria dei numeri

Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos eiusdem nominis fas est dividere cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

Pierre de Fermat (ai margini di una copia dell'Arithmetica di Diofanto)

La teoria dei numeri riguarda lo studio delle proprietà dei numeri interi. La formulazione di molti dei problemi tipici della materia può essere facilmente compresa anche da un non-matematico; tuttavia la soluzione degli stessi problemi è spesso complessa, e richiede tecniche non banali.

La teoria dei numeri nasce nell'antica Grecia, con lo studio dei criteri di divisibilità degli interi (da parte di Euclide ad esempio). Più avanti (nel XVII secolo) fu Pierre de Fermat ad iniziare uno studio più sistematico. Il matematico Francese, in particolare, enunciò numerose congetture, senza fornire per esse una dimostrazione.[90] I contributi successivi, ad opera di Eulero, Lagrange e Gauss, hanno quindi dato inizio alla teoria dei numeri moderna.

Guida per il lettore. Questa appendice richiama alcuni risultati di base in teoria dei numeri, usati in diverse parti del testo. Nel Paragrafo B.1 studieremo i criteri di divisibilità e l'algoritmo euclideo di divisione. Compiremo quindi un viaggio nel mondo dell'aritmetica modulare, studiando le congruenze (nel Paragrafo B.2) e le proprietà dei residui quadratici (nel Paragrafo B.3). nel Paragrafo B.4 ci soffermeremo, infine, sullo studio delle curve ellittiche su un campo finito, che hanno diverse applicazioni in crittografia.

Per un'introduzione più rigorosa alla teoria "classica" dei numeri, si rimanda a [JJ05].

[90] La più famosa di queste è senz'altro il famoso "ultimo teorema", che afferma che non esistono soluzioni intere all'equazione $X^n + Y^n = Z^n$ per $n > 2$. La soluzione dell'enigma di Fermat ha visto impegnate, senza fortuna, intere generazioni di matematici. La congettura è stata dimostrata da Andrew Wiles [Wil95], nel 1994. L'avvincente storia della congettura è raccontata in [Sin97].

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_16,

B.1 Algoritmo euclideo

Cominciamo introducendo i concetti fondamentali di quoziente e resto di una divisione.

Lemma B.1 (Quoziente e resto). *Se a e b sono interi, con $b > 0$, allora esiste un'unica coppia di interi (q, r) tali che $a = qb + r$ e $0 \leq r < b$.*

Dimostrazione. La dimostrazione è costituita da due parti: nella prima parte mostreremo che la coppia (q, r) esiste e nella seconda parte che essa è unica.

Definiamo l'insieme $\mathcal{S} = \{a - nb : n \in \mathbb{Z}\}$. Affermiamo che $\mathcal{S}$ contiene almeno un elemento non-negativo. Infatti se $a > 0$, allora possiamo scegliere $n = 0$ e $a \in \mathcal{S}$. Se invece $a < 0$, possiamo scegliere $n = a$ da cui $a - ab = a(1 - b) \in \mathcal{S}$. Siccome per ipotesi $b > 0$, abbiamo che $(1 - b)$ è negativo e quindi $a(1 - b)$ è un numero positivo oppure nullo (quando $b = 1$). In entrambi i casi comunque $a - ab \in \mathcal{S}$ è non-negativo.

Sia dunque r il minimo intero non negativo contenuto in $\mathcal{S}$. Osserviamo che $r = a - qb$ per un qualche $q \in \mathbb{Z}$, da cui $a = r + qb$. Resta da mostrare che $0 \leq r < b$. Siccome $r \in \mathcal{S}$ è non-negativo per definizione, dobbiamo solamente mostrare che $r < b$. Supponiamo che $r \geq b$, allora:

$$r' = a - (q + 1)b = a - qb - b = r - b \geq 0,$$

contraddicendo il fatto che r sia il minimo elemento non-negativo di $\mathcal{S}$.

Per mostrare che la coppia (q, r) è unica, supponiamo che esistano due coppie distinte (q, r) e (q', r') tali che $a = qb + r$ ed $a = q'b + r'$. Segue $r - r' = b(q' - q)$, ovvero b divide $r - r'$. Siccome però $0 \leq r, r' < b$, deve essere $-b < -r' \leq r - r' < r < b$; pertanto il divisore b è più grande (in valore assoluto) del numero $r - r'$ che divide, il che è possibile solo se $r - r' = 0$ ovvero $r = r'$.

Sostituendo in $qb + r = q'b + r'$, troviamo $qb = q'b$ da cui segue $q = q'$. □

Il resto della divisione è indicato con $a \bmod b = r$. Quando $r = 0$, si è soliti dire che b divide a (oppure che b è un divisore di a) e si scrive $b|a$. Un divisore comune di a e b è un intero che divide sia a che b. Si definisce massimo comun divisore (*greatest common divisor*) di a e b — indicato con $\gcd(a, b)$ — il massimo tra i divisori comuni di a e b. (La definizione è ben posta perché l'insieme dei divisori comuni ad a e b è non vuoto (1 ne fa sempre parte) ed è finito; quindi ha senso parlare di massimo.) Quando $\gcd(a, b) = 1$ si dice che a e b sono *coprimi*.

Descriviamo ora l'algoritmo euclideo per il calcolo del massimo comun divisore tra a e b. L'ingrediente fondamentale è il seguente:

Lemma B.2 (Algoritmo di Euclide). *Siano a e b due interi tali che $a \geq b > 0$. Allora* $\gcd(a, b) = \gcd(b, a \bmod b)$.

Dimostrazione. È sufficiente mostrare che i divisori comuni ad a e b sono gli stessi comuni a b ed $a \bmod b$. Per il Lemma B.1 possiamo sempre scrivere $a = qb + a \bmod b$, dove $q = \left\lfloor \frac{a}{b} \right\rfloor$ e $a \bmod b$ è il resto della divisione tra a e b. Pertanto, un divisore comune ad a e b divide anche $a - qb = a \bmod b$.

D'altra parte, un divisore comune di b ed $a \bmod b$ divide anche $a = qb + a \bmod b$; quindi l'asserto. □

L'algoritmo euclideo sfrutta il lemma precedente in modo iterativo. In primo luogo se $b = 0$ abbiamo immediatamente $\gcd(a, 0) = 0$. Se invece $b \neq 0$ possiamo usare il Lemma B.2 e concludere $\gcd(a, b) = \gcd(b, a \bmod b)$, con $b > a \bmod b \geq 0$, in quanto $a \bmod b$ è il resto della divisione tra a e b. A questo punto applichiamo ricorsivamente il Lemma B.2 agli elementi $(b, a \bmod b)$: ad ogni passo il secondo termine del gcd diminuisce fino a diventare 0. A quel punto:

$$\gcd(a, b) = \gcd(b, a \bmod b) = \cdots = \gcd(d, 0) = d,$$

ed abbiamo calcolato $d = \gcd(a, b)$.

Un esempio chiarisce immediatamente le idee. Sia $a = 185$ e $b = 40$. Applicando iterativamente il Lemma B.2, troviamo:

$$\begin{aligned} \gcd(185, 40) &= \gcd(40, 25) = \gcd(25, 15) = \gcd(15, 10) \\ &= \gcd(10, 5) = \gcd(5, 0) = 5. \end{aligned}$$

Complessità. Vogliamo ora mostrare che l'algoritmo di Euclide è efficiente. Per fare questo dobbiamo innanzitutto discutere la complessità computazionale associata alle operazioni elementari di somma, prodotto e divisione. Sia $O(1)$ il costo relativo alla somma di due bit ed indichiamo con s_a la lunghezza di a (in binario), ovvero $s_a = \lfloor \log a \rfloor + 1$. (Un'espressione identica vale per s_b relativamente a b.) È banale verificare che (cf. Esercizio B.1):

- somma e sottrazione hanno complessità $O(\max\{s_a, s_b\})$;
- la moltiplicazione costa $O(s_a \cdot s_b)$;
- la divisione costa $O(s_q \cdot s_b)$, essendo q è il quoziente della divisione tra a e b.

Osserviamo inoltre che lo spazio richiesto in memoria è $O(s_a + s_b)$.

Per poter valutare la complessità dell'algoritmo di Euclide, dobbiamo in qualche modo limitare il numero di iterazioni necessarie fino al termine dell'algoritmo stesso. Per fare ciò ci serviremo dei numeri di Fibonacci, introdotti per la prima volta dal matematico italiano Leonardo Fibonacci (nel XIII secolo) per studiare la velocità di riproduzione di una famiglia di conigli.

Definizione B.1 (Numeri di Fibonacci). La sequenza di interi $\{F_n\}_{n=0}^{+\infty}$, definita dalle relazioni $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_n + F_{n+1}$, è detta sequenza di Fibonacci. ■

I primi elementi della sequenza di Fibonacci sono $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$. La sequenza soddisfa alcune proprietà interessanti:

Lemma B.3 (Proprietà della sequenza di Fibonacci). *Sia* $\Theta = (1+\sqrt{5})/2$ *il rapporto aureo. Per ogni numero naturale* $n \in \mathbb{N}$ *è valido quanto segue:*

$$(i)\Theta^{n+2} = \Theta^n + \Theta^{n+1} \qquad (ii)(1-\Theta)^{n+2} = (1-\Theta)^n + (1-\Theta)^{n+1}$$

$$(iii)F_n = \frac{1}{\sqrt{5}}\left(\Theta^n - (1-\Theta)^n\right) \qquad (iv)F_{n+1} < \Theta^n < F_{n+2}.$$

Dimostrazione. Per dimostrare (i) e (ii) basta osservare che Θ e $(1-\Theta)$ sono soluzione dell'equazione di secondo grado $X^2 = X + 1$. Pertanto $\Theta^2 = \Theta + 1$ e $(1-\Theta)^2 = (1-\Theta) + 1$. L'asserto segue moltiplicando ambo i membri per Θ^n ed $(1-\Theta)^2$ (rispettivamente).

La (iii) è anche detta formula di Binet (perché mostrata indipendentemente da Binet). Dimostreremo l'espressione per induzione su n. Il caso base si ha in corrispondenza dei valori F_0 ed F_1, ovvero quando $n = 0$ ed $n = 1$:

$$F_0 = \frac{1}{\sqrt{5}}(1-1) = 0 \qquad F_1 = \frac{1}{\sqrt{5}}(\Theta - 1 + \Theta).$$

Assumiamo ora che la formula sia valida per ogni intero $< n$ e mostriamo che ciò implica che essa sia valida anche per n:

$$\begin{aligned} F_n = F_{n-1} + F_{n-2} &= \frac{1}{\sqrt{5}}(\Theta^{n-1} - (1-\Theta)^{n-1} + \Theta^{n-2} - (1-\Theta)^{n-2}) \\ &= \frac{1}{\sqrt{5}}(\underbrace{\Theta^{n-1} + \Theta^{n-2}}_{\Theta^n} - \underbrace{(1-\Theta)^{n-1} + (1-\Theta)^{n-2}}_{(1-\Theta)^n}) \\ &= \frac{1}{\sqrt{5}}(\Theta^n - (1-\Theta)^n). \end{aligned}$$

Dimostreremo la (iv) sempre per induzione su n. Quando $n = 1$ abbiamo in effetti $F_1 = 1 < \Theta < F_2 = 2$. Assumendo che l'espressione sia verificata per ogni intero $< n$, possiamo scrivere:

$$F_{n-1} < \Theta^{n-2} < F_n \qquad F_n < \Theta^{n-1} < F_{n+1}.$$

Ma allora:

$$F_{n-1} + F_n = F_{n+1} < \Theta^{n-2} + \Theta^{n-1} = \Theta^n < F_n + F_{n+1} = F_{n+2},$$

come volevasi dimostrare. □

Il seguente teorema, collega il numero di iterazioni dell'algoritmo di Euclide con la sequenza di Fibonacci.

Teorema B.4 (Teorema di Lamè). *Sia $a \geq b > 0$ ed indichiamo con $E(a, b)$ il numero di passi nell'algoritmo euclideo. Allora per ogni numero naturale $n \in \mathbb{N}$ abbiamo $E(a, b) < n$ ogni volta che $b < F_{n+1}$ oppure $a < F_{n+2}$.*

Dimostrazione. È sufficiente dimostrare che ogni volta in cui $E(a, b) \geq n$, deve aversi $b \geq F_{n+1}$ ed anche $a \geq F_{n+2}$. Procediamo per induzione su n. Nel caso banale in cui $n = 0$, l'asserto diventa:

$$E(a, b) \geq 0 \quad \Rightarrow \quad b \geq F_1 = 1 \text{ e } a \geq F_2 = 2,$$

il che è sicuramente vero in quanto per ipotesi $a \geq b > 0$.

Supponiamo ora che $E(a, b) \geq k$ implichi $b \geq F_{k+1}$ e $a \geq F_{k+2}$, per ogni intero $k \leq n$; mostreremo che ciò implica l'asserto per ogni $k \geq n + 1$. In altri termini dobbiamo dimostrare che se $E(a, b) \geq n + 1$, allora $a \geq F_{n+3}$ e $b \geq F_{n+2}$. Sia $E(a, b) \geq n + 1$; usando il Lemma B.2 possiamo scrivere $\gcd(a, b) = \gcd(b, a \bmod b)$ ed il calcolo di $\gcd(b, a \bmod b)$ richiede esattamente $E(a, b) - 1$ iterazioni. Siccome $E(b, a \bmod b) = E(a, b) - 1 \geq n$, possiamo usare l'ipotesi induttiva per concludere che $b \geq F_{n+2}$ e $a \bmod b \geq F_{n+1}$. Pertanto resta da mostrare che $a \geq F_{n+3}$; ma $a = qb + a \bmod b \geq b + a \bmod b$, da cui $a \geq b + a \bmod b \geq F_{n+2} + F_{n+1} = F_{n+3}$. Questo conclude la prova. □

Il Teorema di Lamè ci consente di calcolare la complessità dell'algoritmo euclideo.

Corollario B.5 (Complessità dell'algoritmo di Euclide). *Poniamo $n = \max\{a, b\}$. La complessità dell'algoritmo di Euclide è $O(\log^3 n)$.*

Dimostrazione. Sia $b = b_0 > 0$ fissato, e concentriamoci su $t = E(a, b_0)$. Il Teorema B.4 implica $a \geq F_{t+2}$. D'altra parte, per il Lemma B.3, sappiamo che $F_{t+2} > \Theta^t$ e quindi $a > \Theta^t$; pertanto:

$$\log_\Theta a > t \quad \Rightarrow \quad t < \log(a) < \log(n).$$

Siccome ogni iterazione dell'algoritmo prevede una divisione (costo $O(\log^2 n)$), la complessità è $O(\log^3 n)$. □

Versione estesa. Possiamo definire una versione estesa dell'algoritmo di Euclide, utile per trovare soluzioni intere ad equazioni della forma $aX + bY = \gcd(a, b)$. La seguente espressione, dovuta a Bézout, ci assicura che una soluzione esiste sempre:

Teorema B.6 (Identità di Bézout). *Se a e b sono interi (non entrambi nulli), allora esistono sempre due interi u, v tali che:*

$$\gcd(a, b) = a \cdot u + b \cdot v.$$

Dimostrazione. Si tratta semplicemente di scrivere le relazioni dell'algoritmo di Euclide alla rovescia. Sia $d = \gcd(a, b) = r_t$ l'ultimo resto non-nullo dell'algoritmo di Euclide (ovvero $r_{t+1} = 0$). Possiamo scrivere:

$$r_t = r_{t-2} - q_{t-1} r_{t-1},$$

esprimendo così d in funzione di r_{t-2} ed r_{t-1}. D'altra parte, al passo precedente dell'algoritmo, si deve avere:

$$r_{t-1} = r_{t-3} - q_{t-2} r_{t-2},$$

e se sostituiamo r_{t-1} nell'espressione per r_t, abbiamo espresso d in funzione di r_{t-2} e di r_{t-3}. Se proseguiamo a ritroso fino al primo passo dell'algoritmo, possiamo esprimere d come somma di un multiplo di $r_0 = a$ ed un multiplo di $r_1 = b$, cioè:

$$\gcd(a, b) = a \cdot u + b \cdot v.$$

□

Più in generale è possibile mostrare che ogni equazione del tipo $aX + bY = c$ ammette soluzione se e solo se c è un multiplo di $\gcd(a, b)$. È facile vedere che i valori di u e v non sono unici; comunque essi possono essere visti come una soluzione dell'equazione $aX + bY = \gcd(a, b)$. Per calcolare tale soluzione, si può usare l'algoritmo esteso di Euclide:

Lemma B.7 (Algoritmo di Euclide esteso). *Definiamo le quantità:*

$$\begin{aligned} x_{k+1} &= q_k x_k + x_{k-1} \qquad x_0 = 1,\ x_1 = 0 \\ y_{k+1} &= q_k x_k + y_{k-1} \qquad y_0 = 0,\ y_1 = 1, \end{aligned}$$

avendo indicato con q_k il k-simo quoziente dell'algoritmo di Euclide applicato agli interi $a \geq b > 0$ (per ogni $k = 1, \cdots, t$). Possiamo scrivere:

$$r_k = (-1)^k x_k a + (-1)^{k+1} y_k b \qquad k = 0, 1, \cdots, t,$$

dove r_k è il k-simo resto nell'algoritmo euclideo, dato da $r_{k-2} = r_{k-1} q_{k-1} + r_k$ per ogni $k = 2, 3, \cdots, t$. (Al solito $r_0 = a$, $r_1 = b$ ed $r_t = d = \gcd(a, b)$.)

Dimostrazione. Per induzione su k. Il caso base ($k = 0$ e $k = 1$) è verificato, in quanto:

$$\begin{aligned} k = 0 \quad &\Rightarrow \quad r_0 = (-1)^0 x_0 a + (-1)^1 y_0 b = a \\ k = 1 \quad &\Rightarrow \quad r_1 = (-1)^1 x_1 a + (-1)^2 y_1 b = b. \end{aligned}$$

Supponiamo ora che l'asserto sia valido per ogni intero $< k$ e mostriamo che questo ne implica la validità per gli interi $\geq k$. Usando l'ipotesi induttiva possiamo concludere:

$$\begin{aligned} r_k &= r_{k-2} - r_{k-1} q_{k-1} \\ &= (-1)^{k-2} x_{k-2} a + (-1)^{k-1} y_{k-2} b - q_{k-1} \left[(-1)^{k-1} x_{k-1} a + (-1)^k y_{k-1} b \right] \\ &= (-1)^k a (x_{k-2} + q_{k-1} x_{k-1}) + (-1)^{k+1} b (y_{k-2} + q_{k-1} y_{k-1}) \\ &= (-1)^k x_k a + (-1)^{k+1} y_k b. \end{aligned}$$

□

Siccome in particolare

$$r_t = \gcd(a, b) = (-1)^t x_t a + (-1)^{t+1} y_t b,$$

segue che $x = (-1)^t x_t$ ed $y = (-1)^{t+1} y_t$ sono soluzioni di $aX + bY = \gcd(a, b)$.

Supponiamo ad esempio di voler risolvere $185X + 40Y = 5$. Applichiamo prima l'algoritmo di Euclide ad $a = 185$ e $b = 40$, ottenendo le relazioni

$$\begin{aligned} 185 = 40 \cdot 4 + 25 \quad &\Rightarrow \quad q_1 = 4, r_2 = 25 \\ 40 = 25 \cdot 1 + 15 \quad &\Rightarrow \quad q_2 = 1, r_3 = 15 \\ 25 = 15 \cdot 1 + 10 \quad &\Rightarrow \quad q_3 = 1, r_4 = 10 \\ 15 = 10 \cdot 1 + 5 \quad &\Rightarrow \quad q_4 = 1, r_5 = 5 \\ 5 = 10 \cdot 2 + 0 \quad &\Rightarrow \quad q_5 = 2, r_6 = 0. \end{aligned}$$

Quindi $t = 5$ e $\gcd(a, b) = r_t = 5$. È banale verificare, inoltre, che $x_5 = 3$ ed $y_5 = 14$. Pertanto:

$$\gcd(185, 40) = r_t = 5 = (-1)^5 \cdot 3 \cdot 185 + (-1)^6 \cdot 14 \cdot 40,$$

il che implica $x = -3$ ed $y = 14$. Non è complesso mostrare che anche la versione estesa ha complessità polinomiale.

B.2 L'aritmetica dell'orologio

In questo paragrafo studieremo i concetti di base dell'aritmetica modulare (detta anche aritmetica dell'orologio poiché su tale principi si basa il calcolo delle ore in cicli di 12 o 24). Per fare ciò ci occorrono alcune nozioni basilari di algebra, in particolare gruppi, anelli e campi.

Definizione B.2 (Gruppo). Un gruppo $(\mathbb{G}, +)$ è un insieme di elementi con un'operazione, per i quali sono verificati i seguenti assiomi:

- *Chiusura.* $\forall g_1, g_2 \in \mathbb{G}$ si ha $g_1 + g_2 \in \mathbb{G}$.
- *Associatività.* $\forall g_1, g_2, g_3 \in \mathbb{G}$ si ha $g_1 + (g_2 + g_3) = (g_1 + g_2) + g_3$.
- *Commutatività.* $\forall g_1, g_2 \in \mathbb{G}$ si ha $g_1 + g_2 = g_2 + g_1$.
- *Elemento neutro.* Esiste un elemento neutro $e \in \mathbb{G}$, tale che $\forall g \in \mathbb{G}$ si ha $g + e = g$.
- *Elemento inverso.* $\forall g \in \mathbb{G}$ esiste ed è unico un elemento inverso $g' \in \mathbb{G}$ tale che $g + g' = e$. ■

Quando vale la proprietà commutativa, il gruppo[91] si dice *abeliano* (in onore del matematico norvegese Niels Henrik Abel). Si definisce *ordine* del gruppo la sua cardinalità $\#\mathbb{G}$. Inoltre diremo che $(\mathbb{H}, +)$ è un *sottogruppo* di $(\mathbb{G}, +)$ se $\mathbb{H}$ è contenuto in $\mathbb{G}$ e se $(\mathbb{H}, +)$ è esso stesso un gruppo.

Definizione B.3 (Anello). Un anello $(\mathbb{A}, +, \cdot)$ è un insieme di elementi con due operazioni, per i quali sono verificati i seguenti assiomi:

- $(\mathbb{A}, +)$ è un gruppo abeliano con elemento neutro e.

[91] Solitamente quando l'operazione è $+$ l'elemento neutro è indicato con 0, l'inverso di g con $-g$ e si scrive ng per $g + \ldots + g$ (n volte).

- *Associatività del* $\cdot$. $\forall a_1, a_2, a_3 \in \mathbb{A}$ si ha $a_1 \cdot (a_2 \cdot a_3) = (a_1 \cdot a_2) \cdot a_3$.
- *Distributività del* $\cdot$ *rispetto al* $+$. $\forall a_1, a_2, a_3 \in \mathbb{A}$ si ha $a_1 \cdot (a_2 + a_3) = a_1 \cdot a_2 + a_1 \cdot a_3$. ■

Quando l'operazione $\cdot$ soddisfa anche la proprietà commutativa, si parla di *anello commutativo*, mentre diremo che $\mathbb{A}$ è *unitario* se esiste l'elemento 1 tale che $\forall a \in \mathbb{A}$ si abbia $a \cdot 1 = a$. Se infine l'anello è commutativo e non esistono divisori dello zero — ovvero se $a \cdot b = 0$ implica che $a = 0$ oppure $b = 0$ — si parla di *dominio d'integrità*.

Definizione B.4 (Campo). Un campo $(\mathbb{K}, +, \cdot)$ è un insieme di elementi con due operazioni, per i quali sono verificati i seguenti assiomi:

- $(\mathbb{K}, +, \cdot)$ è un anello unitario commutativo, con elemento neutro 1.
- *Inverso del* $\cdot$. $\forall x \in \mathbb{K}$ esiste ed è unico $x^{-1} \in \mathbb{K}$ tale che $x \cdot x^{-1} = 1$. ■

Ad esempio l'insieme $\mathbb{Z}$ degli interi è un gruppo rispetto alla somma, ma non lo è rispetto al prodotto (in quanto non sempre c'è un inverso). L'insieme $\mathbb{R} \setminus \{0\}$ di numeri reali (escluso lo 0) è un gruppo rispetto al prodotto.

Il seguente risultato, fondamentale in algebra, è dovuto a Lagrange.

Teorema B.8 (Teorema di Lagrange). *Sia* $\mathbb{H}$ *un sottogruppo di un gruppo finito* $\mathbb{G}$. *Allora* $\#\mathbb{H}$ *divide* $\#\mathbb{G}$.

Si veda ad esempio [Lan02, Proposizione 2.2] per una dimostrazione.

Congruenze e classi di residui. Possiamo ora dare la nozione di *congruenza*. Diremo che gli interi a e b sono congruenti modulo n — indicato con $a \equiv b \pmod{n}$ — se n divide $a - b$. Ad esempio $-2 \equiv 19 \pmod{21}$, $6 \equiv 0 \pmod{2}$ e $2^{340} \equiv 1 \pmod{341}$.

Le congruenze modulo n definiscono una *relazione d'equivalenza*[92] su $\mathbb{Z}$. In questo modo $\mathbb{Z}$ è ripartito in *classi d'equivalenza* contenenti gli interi tra loro

[92] Una relazione ϱ tra elementi di un insieme $\mathcal{S}$ è detta di equivalenza se è:

- *Riflessiva.* Per ogni elemento s di $\mathcal{S}$ si ha $s \varrho s$.
- *Simmetrica.* Per ogni coppia (s_1, s_2) di $\mathcal{S}$ si ha $s_1 \varrho s_2 \Rightarrow s_2 \varrho s_1$.
- *Transitiva.* Per ogni terna (s_1, s_2, s_3) di $\mathcal{S}$ si ha $s_1 \varrho s_2$ e $s_2 \varrho s_3 \Rightarrow s_1 \varrho s_3$.

Un sottoinsieme di $\mathcal{S}$ che contiene tutti e soli gli elementi equivalenti a un qualche elemento s di $\mathcal{S}$ prende il nome di classe di equivalenza di s. L'insieme delle classi di equivalenza su $\mathcal{S}$ si chiama insieme quoziente di $\mathcal{S}$ per la relazione ϱ e viene talvolta indicato con l'espressione $\mathcal{S}/\varrho$.

congruenti modulo n, ovvero gli interi che hanno lo stesso resto quando sono divisi per n. Ciò definisce l'insieme di residui modulo n, indicati con $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$. Ad esempio per $n = 2$, abbiamo $\mathbb{Z}_2 = \{0, 1\}$, in quanto tutti gli interi sono congrui a 0 (se sono pari) oppure ad 1 (se sono dispari) modulo 2. (In altri termini le due classi d'equivalenza sono qui i numeri pari e i numeri dispari.) Quando invece $n = 3$, abbiamo $\mathbb{Z}_3 = \{0, 1, 2\}$ in quanto tutti gli interi n sono congrui a 0 (se $n/3$ ha resto 0), oppure ad 1 (se $n/3$ ha resto 1) oppure a 2 (se $n/3$ ha resto 2), modulo 3.

È facile verificare che $(\mathbb{Z}_n, +)$ è un gruppo rispetto all'operazione di somma modulo n; inoltre $(\mathbb{Z}_n, +, \cdot)$ è un anello commutativo. In generale invece $\mathbb{Z}_n$ non è un campo, in quanto non tutti gli elementi $a \in \mathbb{Z}_n$ sono invertibili:

Lemma B.9 (Elementi invertibili in $\mathbb{Z}_n$). *Se* $\gcd(a, n) > 1$ *allora* $a \in \mathbb{Z}_n$ *non è invertibile modulo* n.

Dimostrazione. Supponiamo per assurdo che a sia invertibile, ovvero esiste $b \in \mathbb{Z}_n$ con $ab \equiv 1 \pmod{n}$. Allora:

$$ab = 1 + nk \quad \Rightarrow \quad ab - nk = 1,$$

per qualche intero k. Evidentemente $\gcd(a, n)$ divide $ab - nk$ e quindi deve anche dividere 1, da cui $\gcd(a, n) = 1$ contro l'ipotesi. □

Si definisce con $\mathbb{Z}_n^*$ l'insieme dei residui invertibili modulo n. Applicando il Lemma B.9, sappiamo che:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \ \gcd(a, n) = 1\}.$$

Notare che la cardinalità di $\mathbb{Z}_n^*$ è data dal numero di elementi minori di n e coprimi con esso.

Definizione B.5 (Funzione toziente di Eulero). Si definisce la funzione toziente di Eulero $\varphi(n)$ come il numero di elementi minori di n e coprimi con esso. ■

Pertanto $\#\mathbb{Z}_n^* = \varphi(n)$. Osserviamo che se $n = p$ è un primo, allora $\varphi(p) = p - 1$; si può dimostrare invece che se $n = p_1 \cdot p_2$, allora $\varphi(n) = \varphi(p_1) \cdot \varphi(p_2)$.

È immediato verificare che $(\mathbb{Z}_n, +, \cdot)$ è un campo se e solo se n è primo. Infatti se n è primo, ogni elemento tranne lo zero è invertibile. D'altra parte se $\mathbb{Z}_n$ è un campo, ogni elemento $< n$ è coprimo con n e quindi n deve essere primo. L'inverso di $a \in \mathbb{Z}_n^*$ può essere calcolato come soluzione dell'equazione $aX + nY = 1$ usando l'algoritmo di Euclide esteso (cf. Lemma B.7).

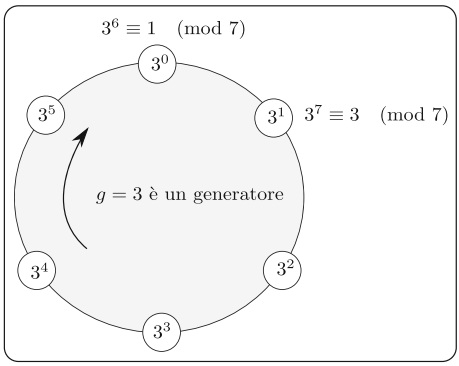


Fig. B.1. Il gruppo ciclico $\mathbb{Z}_7^*$ con il generatore $g = 3$

Sia ora p un primo e consideriamo il *gruppo unitario*:

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\} \quad \text{con} \quad \#\mathbb{Z}_p^* = \varphi(p) = p - 1,$$

ovvero il sottogruppo moltiplicativo di $\mathbb{Z}_p$ contenente tutti gli elementi invertibili modulo p (cioè tutti tranne lo zero). A partire da $g \in \mathbb{Z}_p^*$ calcoliamo le potenze successive di g: $g^0, g^1, g^2, \ldots$; siccome $\mathbb{Z}_p^*$ è finito esistono i, j (con $i \neq j$, diciamo $j < i$) tali che $g^i \equiv g^j \pmod p$. Pertanto $g^{i-j} \equiv 1 \pmod p$, ovvero esiste q tale che $g^q \equiv 1 \pmod p$. Si definisce *ordine* di g *il più piccolo* intero $\mathrm{ord}(g)$ tale che $g^{\mathrm{ord}(g)} \equiv 1 \pmod p$. Un elemento $g \in \mathbb{Z}_p^*$ di ordine $\mathrm{ord}(g) = k$ genera un sottogruppo $\mathbb{G} = \{g^0, g^1, g^2, \ldots, g^{k-1}\}$ di $\mathbb{Z}_p^*$, quindi possiamo concludere $\mathrm{ord}(g) \leq p - 1$. L'elemento g è detto *generatore* del sottogruppo $\mathbb{G}$.

Un elemento di ordine $p-1$ genera tutto $\mathbb{Z}_p^*$ ed è detto *primitivo*. Osserviamo che se g è un elemento primitivo allora $\mathbb{Z}_p^* = \{g^0, g^1, \ldots, g^{p-2}\}$; un gruppo generato dalle potenze successive di un suo elemento è detto *ciclico*. Si può dimostrare che $\mathbb{Z}_p^*$ è sempre ciclico, con $\varphi(p-1)$ elementi primitivi. Ad esempio l'insieme $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$ è ciclico con $\varphi(6) = \varphi(3) \cdot \varphi(2) = 6$ elementi primitivi (cf. Fig. B.1).

A volte è utile calcolare l'ordine di un elemento g appartenente ad un gruppo $\mathbb{G}$ con $\#\mathbb{G} = q$ elementi. Sia $q = p_1^{e_1} \cdot \cdots \cdot p_k^{e_k}$ la scomposizione in fattori primi di q. Per ogni divisore primo p_i di q, sia f_i il massimo intero tale che $g^{q/p_i^{f_i}} = 1$. Allora, l'ordine di g è:

$$\mathrm{ord}(g) = p_1^{e_1 - f_1} \cdot \cdots \cdot p_k^{e_k - f_k}.$$

Se ad esempio prendiamo $\mathbb{Z}_{101}^*$ e $g = 2$, si ha $q = 100 = 2^2 \cdot 5^2$. Poiché $2^{50} \equiv -1 \pmod{101}$ e $2^{20} \equiv -6 \pmod{101}$ si ha $f_1 = 0$ ed $f_2 = 0$; quindi $\mathrm{ord}(2) = 100$ in $\mathbb{Z}_{101}^*$ (ovvero $\mathbb{Z}_p^*$ è ciclico e 2 è un elemento primitivo).

È anche facile verificare quando un certo intero k è l'ordine di un elemento $g \in \mathbb{G}$. Ciò si verifica quando $g^k \equiv 1$ e $g^{k/p} \neq 1$ in $\mathbb{G}$, per ogni divisore primo p di k. Ad esempio 25 è l'ordine di 5 in $\mathbb{Z}_{101}^*$, poiché $5^{25} \equiv 1 \pmod{101}$, ma $5^5 \not\equiv -6 \pmod{101}$.

Alcuni teoremi fondamentali. Concludiamo il paragrafo con la dimostrazione di alcuni teoremi fondamentali nelle applicazioni crittografiche della teoria dei numeri.

Teorema B.10 (Teorema di Eulero). *Sia n un intero positivo. Allora per ogni $a \in \mathbb{Z}_n^*$ si ha $a^{\varphi(n)} \equiv 1 \pmod{n}$.*

Dimostrazione. Consideriamo l'insieme $\mathbb{Z}_n^* = \{x_1, \ldots, x_{\varphi(n)}\}$. Definiamo gli elementi $y_i = a \cdot x_i \bmod n$; è facile mostrare che questi non sono altro che gli elementi di $\mathbb{Z}_n^*$ in ordine diverso (ovvero permutati). Sia inoltre $M = y_1 \cdot \cdots \cdot y_{\varphi(n)}$; notare che M è coprimo con n, quindi è invertibile modulo n. Pertanto

$$\begin{aligned} M \equiv y_1 \cdot \cdots \cdot y_{\varphi(n)} = a x_1 \cdot \cdots \cdot a x_{\varphi(n)} = a^{\varphi(n)} M \pmod{n} \\ \Rightarrow \quad a^{\varphi(n)} \equiv 1 \pmod{n}. \end{aligned}$$

□

Un corollario del teorema di Eulero è il piccolo teorema di Fermat, dimostrato indipendentemente da Fermat.

Teorema B.11 (Piccolo teorema di Fermat). *Sia p un primo. Allora per ogni $a \in \mathbb{Z}_p^*$ si ha $a^{p-1} \equiv 1 \pmod{p}$.*

Dimostrazione. Banale se sostituiamo $n = p$ (per un primo p) nel Teorema di Eulero. □

Il teorema del resto cinese (*Chinese Remainder Theorem*, CRT) studia alcuni sistemi di congruenze.

Teorema B.12 (Teorema del resto cinese). *Siano $m_1, \ldots, m_k$ interi a due a due coprimi (ovvero $\gcd(m_i, m_j) = 1$ per ogni $i \neq j$). Allora, per ogni insieme di valori $(a_1, \ldots, a_k)$, esiste un intero x soluzione del sistema di congruenze:*

$$x \equiv a_i \pmod{m_i} \quad per \quad i = 1, \ldots, k.$$

Tale soluzione è unica modulo $\prod_{i=1}^{k} m_i$.

Dimostrazione. Sia $M_i = \prod_{j \neq i} m_j$. Siccome gli m_i sono a due a due coprimi, $\gcd(M_i, m_i) = 1$. Usando l'algoritmo di Euclide esteso possiamo quindi calcolare gli interi (x_i, y_i) tali che:

$$y_i M_i + x_i m_i \equiv \gcd(M_i, m_i) = 1 \pmod{m_i}.$$

Per tali valori risulta ovviamente $y_i M_i \equiv 1 \pmod{m_i}$. Sia allora:

$$x \equiv \sum_{i=1}^{k} a_i y_i M_i \pmod{m} \quad \text{con} \quad m = \prod_{i=1}^{k} m_i.$$

È facile verificare che tale x è il valore cercato. Infatti $x \equiv a_i \pmod{m_i}$ in quanto $a_i y_i M_i \equiv a_i \pmod{m_i}$ ed $a_j y_j M_j \equiv 0 \pmod{m_i}$ per ogni $j \neq i$. Tale soluzione è inoltre unica; siano infatti x ed x' due soluzioni. Allora $x \equiv x' \pmod{m_i}$, per ogni $i = 1, 2, \ldots k$. Ma poiché gli m_i sono coprimi, $x - x'$ deve essere multiplo di $m = \prod_{i=1}^{k} m_i$, ovvero $x \equiv x' \pmod{m}$. □

Consideriamo ad esempio le seguenti congruenze:

$$x \equiv 2 \pmod 4 \quad x \equiv 1 \pmod 3 \quad x \equiv 0 \pmod 5.$$

Abbiamo $m_1 = 4$, $m_2 = 3$, $m_3 = 5$, $a_1 = 2$, $a_2 = 1$ ed $a_3 = 0$. Quindi $m = 60$, $M_1 = 15$, $M_2 = 20$, $M_3 = 12$. È facile verificare che le soluzioni delle congruenze $y_1 M_1 \equiv 1 \pmod{m_1}$, $y_2 M_2 \equiv 1 \pmod{m_2}$ ed $y_3 M_3 \equiv 1 \pmod{m_3}$, sono (rispettivamente) $y_1 = 3$, $y_2 = 2$ ed $y_3 = 3$. Pertanto

$$x \equiv (2 \cdot 3 \cdot 15 + 1 \cdot 2 \cdot 20 + 0 \cdot 3 \cdot 12) \equiv 130 \equiv 10 \pmod{60}.$$

Notare che il costo computazionale è $O\left(\log^2 m\right)$, l'occupazione di memoria $O(\log m)$.

Nella sua forma più generale il CRT caratterizza la struttura di alcuni anelli commutativi attraverso isomorfismi. Ricordiamo che un isomorfismo è una

mappa biettiva ψ, tale che sia ψ che ψ^{-1} sono omomorfismi. Consideriamo gli anelli $(\mathbb{A}_1, +, \cdot)$ ed $(\mathbb{A}_2, \boxplus, \boxdot)$. La mappa

$$\begin{aligned} \psi :\ & \mathbb{A}_1 \longrightarrow \mathbb{A}_2 \\ & x \in \mathbb{A}_1 \mapsto \psi(x) \in \mathbb{A}_2, \end{aligned}$$

è un omomorfismo di anelli se preserva la struttura, ovvero se soddisfa

$$\begin{aligned} \psi(x_1 + x_2) &= \psi(x_1) \boxplus \psi(x_2) \\ \psi(x_1 \cdot x_2) &= \psi(x_1) \boxdot \psi(x_2). \end{aligned}$$

(Una definizione identica vale per i gruppi e si parla di omomorfismo di gruppi.) Quando $\mathbb{A}_1$ ed $\mathbb{A}_2$ sono isomorfi, si scrive $\mathbb{A}_1 \simeq \mathbb{A}_2$.

Teorema B.13 (CRT generalizzato). *Siano $p_1, \ldots, p_n$ numeri primi tali che $\gcd(p_i, p_j) = 1$ per ogni $i \neq j$. Sia inoltre $n = \prod_{i=1}^{n} p_i$. Allora la mappa*

$$\begin{aligned} f :\ & \mathbb{Z}_n \longrightarrow \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_n} \\ & x \mapsto (x \bmod p_1, \ldots, x \bmod p_n)\,, \end{aligned}$$

e la mappa

$$\begin{aligned} g :\ & \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_n}^* \\ & x \mapsto (x \bmod p_1, \ldots, x \bmod p_n)\,, \end{aligned}$$

sono isomorfismi.

B.3 Residui quadratici

In questo paragrafo ci concentreremo sulle proprietà dei residui quadratici, definiti di seguito.

Residui quadratici modulo p. Diremo che $a \in \mathbb{Z}_p^*$ è un residuo quadratico (*quadratic residue*) modulo p, se esiste $b \in \mathbb{Z}_p^*$ tale che $a \equiv b^2 \pmod{p}$. L'insieme dei residui quadratici modulo p si indica con

$$\mathbb{QR}_p = \left\{ a \in \mathbb{Z}_p^* :\ a \equiv b^2 \bmod p, \text{ per qualche } b \in \mathbb{Z}_p^* \right\}.$$

Quante radici quadrate ha un residuo quadratico modulo p?

Lemma B.14 (Radici quadrate in $\mathbb{QR}_p$). *Ogni elemento $a \in \mathbb{QR}_p$ ha esattamente due radici quadrate modulo p.*

Dimostrazione. Sia $a \in \mathbb{QR}_p$. Allora $a \equiv b^2 \pmod{p}$ per qualche $b \in \mathbb{Z}_p^*$. Ovviamente $(-b)^2 = b^2 \equiv a \pmod{p}$. Inoltre $b \not\equiv -b \pmod{p}$ e quindi gli elementi $\pm b$ sono due radici quadrate di a e possiamo concludere che a ha *almeno* due radici quadrate modulo p.

Per vedere che queste sono uniche, sia b' un'altra radice di a. Allora $b^2 = a \equiv (b')^2 \pmod{p}$, da cui $b^2 - (b')^2 \equiv 0 \pmod{p}$. Fattorizzando il primo membro si ottiene $(b-b')(b+b') \equiv 0 \pmod{p}$, così che una delle due seguenti condizioni è verificata: (i) $p \mid (b-b')$, oppure (ii) $p \mid (b+b')$. Nel primo caso $b \equiv b' \pmod{p}$ e nel secondo caso $-b \equiv b' \pmod{p}$; pertanto $\pm b$ sono le uniche radici di a modulo p. □

Siccome l'elemento neutro del prodotto è $1 \in \mathbb{QR}_p$, e siccome quando $a, a' \in \mathbb{QR}_p$ anche $a \cdot a' \in \mathbb{QR}_p$, l'insieme dei residui quadratici modulo p è un sottogruppo di $(\mathbb{Z}_p^*, \cdot)$. L'insieme $\mathbb{QNR}_p = \mathbb{Z}_p^* \setminus \mathbb{QR}_p$ è detto insieme dei non-residui quadratici di $\mathbb{Z}_p^*$.

Consideriamo, ad esempio, $\mathbb{Z}_{11}^* = \{0, 1, 2, \ldots, 10\}$. È facile verificare che $\mathbb{QR}_{11} = \{1 = 1^2 = 10^2, 4 = 2^2 = 9^2, 9 = 3^2 = 8^2, 5 = 4^2 = 7^2, 3 = 5^2 = 6^2\}$ (cf. Fig. B.2).

Il seguente lemma determina la cardinalità di $\mathbb{QR}_p$.

Lemma B.15 (Cardinalità di $\mathbb{QR}_p$). *Per ogni primo p, si ha $\#\mathbb{QR}_p = (p-1)/2$.*

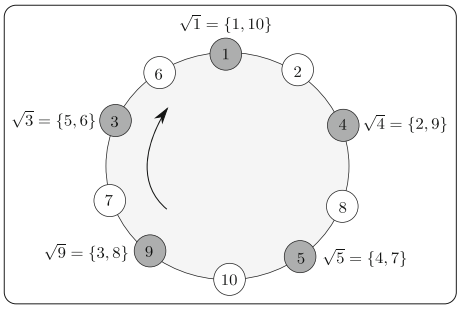


Fig. B.2. Il gruppo $\mathbb{QR}_{11}$

Dimostrazione. Sia g un generatore di $\mathbb{Z}_p^*$; abbiamo $\mathbb{QR}_p = \{g^2, g^4, \ldots, g^{p-1}\}$. Infatti, siccome ogni $x \in \mathbb{Z}_p^*$ è uguale a g^i per qualche i, possiamo concludere:

$$x^2 \equiv g^{2i} \pmod{p-1} \equiv g^j \pmod{p},$$

per un qualche j pari. □

Una conseguenza immediata è che $\#\mathbb{QNR}_p = p - (p-1)/2 = (p-1)/2$. Quindi esattamente metà degli elementi di $\mathbb{Z}_p^*$ è un residuo quadratico e l'altra metà è un non-residuo quadratico.

Possiamo anche verificare velocemente se un dato elemento è un residuo quadratico modulo p o meno.

Lemma B.16 (Criterio di Eulero). *Per ogni primo p, abbiamo che $a \in \mathbb{QR}_p$ se e solo se $a^{(p-1)/2} \equiv 1 \pmod{p}$.*

Dimostrazione. ($\Rightarrow$) Se $a \in \mathbb{QR}_p$ allora $a \equiv g^{2i} \pmod{p}$ per qualche i. Pertanto:

$$a^{\frac{p-1}{2}} \equiv \left(g^{2i}\right)^{\frac{p-1}{2}} \equiv g^{i(p-1)} \equiv 1 \pmod{p}.$$

($\Leftarrow$) Per assurdo. Supponiamo che $a \notin \mathbb{QR}_p$ con $a^{(p-1)/2} \equiv 1 \pmod{p}$. Dunque a è della forma $a \equiv g^{2i+1} \pmod{p}$ per qualche i. Pertanto:

$$a^{\frac{p-1}{2}} \equiv \left(g^{2i+1}\right)^{\frac{p-1}{2}} \equiv g^{i(p-1)} g^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \not\equiv 1 \pmod{p},$$

(essendo g un generatore) contro l'ipotesi. □

Per distinguere i residui quadratici dai non-residui quadratici, si introduce il simbolo di Legendre.

Definizione B.6 (Simbolo di Legendre). Sia $a \in \mathbb{Z}_p^*$. Il simbolo di Legendre $J_p(a)$ è definito come:

$$J_p(a) = \begin{cases} +1 & \text{se } a \in \mathbb{QR}_p \\ -1 & \text{se } a \in \mathbb{QNR}_p. \end{cases}$$

■

Il criterio di Eulero può essere quindi riformulato come $J_p(a) \equiv a^{(p-1)/2} \pmod{p}$ per ogni $a \in \mathbb{Z}_p^*$.

Il simbolo di Legendre soddisfa la seguente proprietà moltiplicativa:

Lemma B.17 (Proprietà moltiplicativa del simbolo di Legendre). *Sia $p > 2$ un primo ed $a, b \in \mathbb{Z}_p^*$. Allora:*

$$J_p(a \cdot b) = J_p(a) \cdot J_p(b).$$

Dimostrazione. Usando il criterio di Eulero, possiamo scrivere:

$$J_p(a \cdot b) = (a \cdot b)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} \cdot b^{\frac{p-1}{2}} = J_p(a) \cdot J_p(b) \pmod{p}.$$

Siccome $J_p(a \cdot b), J_p(a), J_p(b) \in \{\pm 1\}$, l'uguaglianza vale anche senza modulo. □

Un semplice corollario del Lemma B.17 è che se $a, a' \in \mathbb{QR}_p$ e $b, b' \in \mathbb{QNR}_p$, allora:

$$a \cdot a' \bmod p \in \mathbb{QR}_p \quad b \cdot b' \bmod p \in \mathbb{QR}_p \quad a \cdot b \bmod p \in \mathbb{QNR}_p.$$

A volte non vogliamo solamente poter verificare se un dato elemento è un residuo quadratico, ma (in caso affermativo) calcolarne esplicitamente una radice quadrata. In $\mathbb{QR}_p$ ciò è sempre possibile in modo efficiente. Ci limiteremo comunque a discutere il caso semplice in cui $p \equiv 3 \pmod 4$. Sia dunque p della forma $p = 4k + 3$, ne segue che l'ordine di $\mathbb{QR}_p$ è $(p-1)/2 = 2k+1$. È facile verificare allora che $\sqrt{a} \equiv a^{k+1} \pmod p$; infatti:

$$\left(a^{k+1}\right)^2 \equiv a^{2(k+1)} \equiv a^{2k+1} a \equiv a \pmod{p}.$$

Residui quadratici modulo $N = p \cdot q$. Passiamo ora a discutere il caso di $\mathbb{QR}_N$, quando $N = p \cdot q$ è il prodotto di due primi distinti; in questo caso, i residui quadratici in $\mathbb{Z}_N^*$ sono tutti gli elementi $a \in \mathbb{Z}_N^*$ della forma $a \equiv b^2 \pmod N$. Per caratterizzare la struttura di $\mathbb{QR}_N$ useremo il teorema del resto cinese (cf. Teorema B.13) ovvero il fatto che $\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*$; in virtù di ciò possiamo rappresentare un elemento $a \in \mathbb{Z}_N^*$ come $a = (a_p, a_q)$, dove $a_p = a \bmod p$ ed $a_q = a \bmod q$. L'osservazione chiave è data dal seguente:

Lemma B.18 (Elementi di $\mathbb{QR}_N$). *Sia $N = p \cdot q$ con p e q primi distinti ed $a = (a_p, a_q)$. Allora $a \in \mathbb{QR}_N$ se e solo se $a_p \in \mathbb{QR}_p$ ed $a_q \in \mathbb{QR}_q$.*

Dimostrazione. ($\Rightarrow$) Se a è in $\mathbb{QR}_N$, allora esiste $b \in \mathbb{Z}_N^*$ tale che $a \equiv b^2 \pmod N$. Sia $b = (b_p, b_q)$; segue:

$$(a_p, a_q) = a = b^2 = (b_p^2 \bmod p, b_q^2 \bmod q).$$

Ma allora:

$$a_p \equiv b_p^2 \pmod{p} \qquad a_q \equiv b_q^2 \pmod{q} \tag{B.1}$$

ed a_p, a_q sono residui quadratici in $\mathbb{QR}_p$ e $\mathbb{QR}_q$ (rispettivamente).

($\Leftarrow$) D'altra parte, se $a = (a_p, a_q)$ ed a_p, a_q sono residui quadratici (rispetto agli opportuni moduli), allora esistono $b_p \in \mathbb{Z}_p^*$ e $b_q \in \mathbb{Z}_q^*$ tali che l'Eq. (B.1) è verificata. Sia $b = (b_p, b_q)$. Percorrendo l'altro verso della dimostrazione al contrario, è immediato verificare che b è la radice quadrata di a modulo N. □

Se esaminiamo con attenzione la dimostrazione precedente, possiamo concludere che ogni residuo quadratico $a \in \mathbb{QR}_N$ ha esattamente 4 radici quadrate. Scriviamo $a = (a_p, a_q)$ e siano $\pm b_p$ e $\pm b_q$ le radici quadrate di a_p ed a_q modulo p e modulo q rispettivamente. (Queste sono le uniche radici quadrate di b modulo p e modulo q, cf. Lemma B.14.) Ne segue che le radici di a sono date dagli elementi di $\mathbb{Z}_N^*$ corrispondenti alle coppie

$$(b_p, b_q) \qquad (-b_p, b_q) \qquad (b_p, -b_q) \qquad (-b_p, -b_q).$$

Tali elementi sono distinti, perché b_p e $-b_p$ (risp. b_q e $-b_q$) sono unici modulo p (risp. modulo q). Pertanto:

$$\frac{\#\mathbb{QR}_N}{\mathbb{Z}_N^*} = \frac{\#\mathbb{QR}_p \cdot \#\mathbb{QR}_q}{\#\mathbb{Z}_N^*} = \frac{\frac{p-1}{2}\frac{q-1}{2}}{(p-1)(q-1)} = \frac{1}{4}.$$

Possiamo anche estendere il simbolo di Legendre al caso $N = p \cdot q$; in questo caso si parla di simbolo di Jacobi. Per ogni a coprimo con N, definiamo

$$J_N(a) = J_p(a) \cdot J_q(a).$$

Indicheremo con $\mathbb{J}_N^{+1}$ l'insieme degli elementi in $\mathbb{Z}_N^*$ con simbolo di Jacobi $+1$ (una definizione simile vale per $\mathbb{J}_n^{-1}$). Già sappiamo (dal Lemma B.18) che se $a \in \mathbb{QR}_N$, allora $J_N(a) = +1$; ciò implica immediatamente che $\mathbb{QR}_N \subseteq \mathbb{J}_N^{+1}$. Osserviamo però che $J_N(a) = +1$ anche quando $J_p(a) = J_q(a) = -1$. In altre parole, $J_N(a) = +1$ anche quando *entrambi* $a \bmod p$ ed $a \bmod q$ *non* sono residui quadratici (e quindi anche se a è un non-residuo quadratico modulo N). Per tenere conto di ciò, si introduce l'insieme di elementi di $\mathbb{Z}_N^*$ che hanno simbolo di Jacobi $+1$ pur essendo non-residui quadratici modulo N:

$$\mathbb{QINR}_N^{+1} = \{a \in \mathbb{Z}_N^* : \ a \in \mathbb{QINR}_N, \text{ ma } J_N(a) = +1\}.$$

Lemma B.19 (Struttura di $\mathbb{Z}_N^*$). *Sia $N = p \cdot q$, con p e q primi distinti. Allora: (i) esattamente metà degli elementi di $\mathbb{Z}_N^*$ sono in $\mathbb{J}_N^{+1}$, (ii) esattamente metà degli elementi di $\mathbb{J}_N^{+1}$ sono in $\mathbb{QR}_N$ (e l'altra metà sono in $\mathbb{QINR}_N$).*

Dimostrazione. (i) Sappiamo che $J_N(a) = +1$ se $J_p(a) = J_q(a) = +1$ oppure se $J_p(a) = J_q(a) = -1$. Inoltre metà degli elementi in $\mathbb{Z}_p^*$ (risp. $\mathbb{Z}_q^*$) ha simbolo di Jacobi $+1$ e l'altra metà -1. Allora:

$$\begin{aligned}\#\mathbb{J}_N^{+1} &= \#(\mathbb{J}_p^{+1} \times \mathbb{J}_q^{+1}) + \#(\mathbb{J}_p^{-1} \times \mathbb{J}_q^{-1}) = \#\mathbb{J}_p^{+1} \cdot \#\mathbb{J}_q^{+1} + \#\mathbb{J}_p^{-1} \cdot \#\mathbb{J}_q^{-1} \\ &= 2\frac{(p-1)(q-1)}{4} = \frac{\varphi(N)}{2} = \frac{\#\mathbb{Z}_N^*}{2}.\end{aligned}$$

(ii) Siccome $a \in \mathbb{QR}_N$ se e solo se $J_p(a) = J_q(a) = +1$, abbiamo:

$$\#\mathbb{QR}_N = \#(\mathbb{J}_p^{+1} \times \mathbb{J}_q^{+1}) = \frac{(p-1)}{2}\frac{(q-1)}{2} = \frac{\varphi(N)}{4},$$

da cui segue $\#\mathbb{QR}_N = \#\mathbb{J}_N^{+1}/2$. Siccome $\mathbb{QR}_N \subseteq \mathbb{J}_N^{+1}$, metà degli elementi di $\mathbb{J}_N^{+1}$ sono in $\mathbb{QR}_N$ (e quindi l'altra metà in $\mathbb{QNR}_N$). □

Il simbolo di Jacobi eredita la proprietà moltiplicativa dal simbolo di Legendre:

Lemma B.20 (Proprietà moltiplicativa del simbolo di Jacobi). *Siano $a, b \in \mathbb{Z}_N^*$. Allora $J_N(a \cdot b) = J_N(a) \cdot J_N(b)$.*

Dimostrazione. Possiamo scrivere:

$$\begin{aligned}J_N(a \cdot b) = J_p(a \cdot b) \cdot J_q(a \cdot b) &\overset{\text{Lemma B.17}}{=} J_p(a) \cdot J_p(b) \cdot J_q(a) \cdot J_q(b) \\ &= J_p(a) \cdot J_q(a) \cdot J_p(b) \cdot J_q(b) = J_N(a) \cdot J_N(b).\end{aligned}$$

□

Una conseguenza delle proprietà elencate è il seguente:

Lemma B.21 ($\mathbb{QR}_N$ e $\mathbb{QNR}^{+1}$). *Siano $a, a' \in \mathbb{QR}_N$ e $b, b' \in \mathbb{QNR}_N^{+1}$. Allora: (i) $a \cdot a' \bmod N \in \mathbb{QR}_N$, (ii) $b \cdot b' \bmod N \in \mathbb{QR}_N$ e (iii) $a \cdot b \bmod N \in \mathbb{QNR}_N^{+1}$.*

Dimostrazione. Dimostriamo solo la (iii), la prova delle altre affermazioni è lasciata come esercizio. Siccome $a \in \mathbb{QR}_N$, abbiamo $J_p(a) = J_q(a) = +1$. D'altra parte, siccome $b \in \mathbb{QNR}_N^{+1}$, abbiamo $J_p(b) = J_q(b) = -1$. Usando il Lemma B.17, possiamo scrivere:

$$J_p(a \cdot b) = J_p(a) \cdot J_p(b) = -1 \qquad \text{e} \qquad J_q(a \cdot b) = J_q(a) \cdot J_q(b) = -1,$$

quindi $J_N(a \cdot b) = +1$ per il Lemma B.20. Ora chiaramente $a \cdot b$ non è un residuo quadratico modulo N, perché $J_p(a \cdot b) = -1$ ovvero $a \cdot b \bmod p$ non è un residuo quadratico modulo p. Di conseguenza, $a \cdot b \in \mathbb{QNR}_N^{+1}$. □

Abbiamo visto che, usando il criterio di Eulero (cf. Lemma B.16), è possibile verificare in modo efficiente se un dato elemento di $\mathbb{Z}_p^*$ è un residuo quadratico oppure no. Analogamente, è semplice verificare se $a \in \mathbb{Z}_N^*$ è un elemento di $\mathbb{QR}_N$: si calcolano $J_p(a), J_q(a)$ e quindi si verifica che $J_p(a) = J_q(a) = +1$; se questo è il caso $a \in \mathbb{QR}_N$, altrimenti $a \in \mathbb{QNR}_N$. La situazione è più complicata quando la fattorizzazione di N *non* è nota: in questo caso non si conosce nessun algoritmo efficiente per decidere se a è un residuo quadratico modulo n oppure no. (Sorprendentemente, però, è noto un algoritmo per calcolare $J_N(a)$ senza conoscere la fattorizzazione di N [ES98; BZ10].)

Interi di Blum. Un primo di Blum è un primo p della forma $p \equiv 3 \pmod 4$. Un intero N è detto di Blum se $N = p \cdot q$ e se sia p che q sono primi di Blum. La principale proprietà degli interi di Blum è che ogni residuo quadratico a ha una radice b che è anch'essa un residuo quadratico. Tale radice quadrata, è anche detta radice quadrata *principale* di a.

Lemma B.22 (Radici in $\mathbb{QR}_p$ quando p è un primo di Blum). *Sia p un primo di Blum ed $a \in \mathbb{QR}_p$. Allora $b = a^{(p+1)/4} \bmod p$ è una radice quadrata principale di b modulo p.*

Dimostrazione. Dobbiamo mostrare che la radice quadrata di b (modulo p), è anch'essa un residuo quadratico. Siccome a è un residuo quadratico per ipotesi, il criterio di Eulero (cf. Lemma B.16) implica $a^{(p-1)/2} \equiv 1 \pmod p$. In effetti:

$$b^2 \equiv \left(a^{(p+1)/4}\right)^2 \equiv a \cdot a^{(p-1)/2} \equiv a \pmod p,$$

e quindi b è una radice quadrata di a modulo p. Per verificare che $b \in \mathbb{QR}_p$ basta applicare nuovamente il criterio di Eulero

$$b^{(p-1)/2} \equiv \left(a^{(p+1)/4}\right)^{(p-1)/2} \equiv \left(a^{(p-1)/2}\right)^{(p+1)/4} \equiv 1^{(p+1)/4} \equiv 1 \pmod p.$$

Segue $b \in \mathbb{QR}_p$ e quindi b è principale. □

Il caso di $\mathbb{QR}_N$, con $N = p \cdot q$, è simile:

Lemma B.23 (Radici in $\mathbb{QR}_N$ quando N è un intero di Blum). *Sia $N = p \cdot q$ un intero di Blum e sia $a \in \mathbb{QR}_N$. Allora a ha quattro radici quadrate modulo N, una sola delle quali è principali.*

Dimostrazione. Abbiamo già mostrato (cf. discussione dopo il Lemma B.18) che ogni $a \in \mathbb{QR}_N$ per $N = p \cdot q$ ha quattro radici modulo N. È un semplice esercizio verificare che di queste una sola è a sua volta un residuo quadratico. □

B.4 Curve ellittiche

Le curve ellittiche sono uno strumento potente in teoria dei numeri con svariate applicazioni in diversi campi della scienza. In crittografia hanno dato vita ad una vera e propria branca, detta appunto crittografia sulle curve ellittiche (*Elliptic Curve Cryptography*, ECC). In questo paragrafo introdurremo (per lo più informalmente) i concetti relativi alle curve ellittiche utili ad alcune applicazioni crittografiche. Una trattazione completa e rigorosa esula dagli scopi del testo. Si vedano ad esempio [Sil86; Was08] per approfondire. Una panoramica si trova anche in [Ven09].

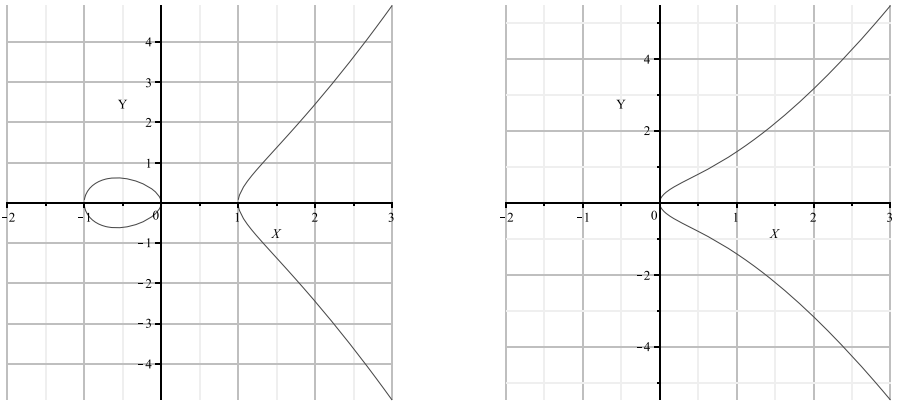
Curve ellittiche su campo $\mathbb{K}$. Sia $(\mathbb{K}, +, \cdot)$ un campo (cf. Definizione B.4) con elemento neutro della somma 0 ed elemento neutro del prodotto 1. Si definisce *caratteristica* di $\mathbb{K}$ (indicata con $\text{char}(\mathbb{K})$), come il minimo numero di volte che bisogna sommare l'elemento 1 per ottenere 0. Se tale valore non esiste, si dice che il campo ha caratteristica nulla, $\text{char}(\mathbb{K}) = 0$. (Ad esempio $\mathbb{Z}_p^*$ ha caratteristica p, mentre l'insieme dei reali $\mathbb{R}$ ha caratteristica 0.) Daremo la definizione di curva ellittica per campi con caratteristica diversa da 2 e 3; anche in questi casi è possibile dare una definizione, ma vanno in un certo senso trattati a parte [Was08, Capitolo 2].

Definizione B.7 (Curva ellittica su un campo $\mathbb{K}$). Sia $\mathbb{K}$ un campo con caratteristica $\text{char}(\mathbb{K}) \neq 2, 3$. Una curva ellittica su $\mathbb{K}$, indicata con $E(\mathbb{K})$, è definita dall'equazione

$$E: \; Y^2 = X^3 + AX + B \quad A, B \in \mathbb{K}. \tag{B.2}$$

■

La curva E è detta non-singolare se non ha zeri doppi, il che si verifica quando il discriminante $\Delta_E = 4A^3 + 27B^2$ è diverso da 0 in $\mathbb{K}$. Se $\mathbb{K} = \mathbb{R}$ possiamo tracciare un grafico della curva, come mostra la Fig. B.3. La curva E è simmetrica (rispetto all'asse delle ascisse) e può avere uno oppure tre zeri reali.



(a) $Y^2 = X^3 - X$ (b) $Y^2 = X^3 + X$

Fig. B.3. Due curve ellittiche su $\mathbb{K} = \mathbb{R}$

Per le applicazioni che ci interessano, è necessario *immergere la curva nel piano proiettivo*. Non presenteremo una definizione formale. Il risultato è che questa operazione consente di aggiungere alla curva un cosiddetto *punto all'infinito* (indicato con ∞) il cui ruolo è quello di dotare l'insieme dei punti di E della struttura di gruppo algebrico. Geometricamente, il punto ∞ può essere pensato come il punto in cui si incontrano due rette parallele.

Più formalmente, sia $Y = \lambda X + \mu$ l'equazione della retta per P e Q. Se calcoliamo l'intersezione con la curva otteniamo:

$$\begin{aligned}
&(\lambda X + \mu)^2 = X^3 + A\ X + B \\
\Rightarrow &X^3 - \lambda^2 X^2 + (A - 2\mu\lambda)X + B - \mu^2 = 0 \\
\Rightarrow &X^3 - \lambda^2 X^2 + (A - 2\mu\lambda)X + B - \mu^2 = (X - x_P)(X - x_Q)(X - x_S) = 0.
\end{aligned}$$

Sviluppando il secondo membro è facile vedere che il coefficiente di X^2 è $-(x_P + x_Q + x_S)$, così che deve risultare $\lambda^2 = x_P + x_Q + x_S$, ovvero

$$x_S = -x_P - x_Q + \lambda^2 = x_R.$$

Inoltre, possiamo sempre scrivere:

$$\begin{aligned}
\lambda = \frac{y_S - y_P}{x_S - x_P} &= \frac{-y_R - y_P}{x_R - x_P} \\
\Rightarrow \quad y_R &= -y_P - (x_R - x_P)\lambda.
\end{aligned}$$

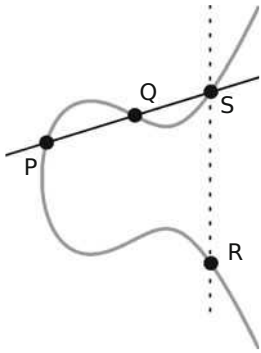


Fig. B.4. Il punto R, somma dei punti P e Q su una curva ellittica

La legge di gruppo. Definiamo ora un'operazione su $E(\mathbb{K})$ tale che dati due punti di $E(\mathbb{K})$, restituisce un altro punto di $E(\mathbb{K})$. Iniziamo con i due punti $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$, e consideriamo la retta passante attraverso essi; come mostra la Fig. B.4, tale retta interseca la curva in un terzo punto $S = (x_S, y_S)$. Sia $R = (x_R, y_R) = (x_S, -y_S)$ il simmetrico del punto S rispetto all'asse delle ascisse. La somma dei punti P e Q è definita come $P + Q = R$. Restano da distinguere tre casi:

1. $P \neq Q$. Abbiamo semplicemente:
$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}.$$
2. $P = Q$. In questo caso λ è il coefficiente angolare della retta tangente alla curva E in $P = Q = (x_P, y_P)$. Dunque:
$$\begin{aligned} & Y = \sqrt{X^3 + Ax + B} \\ \Rightarrow \quad & \frac{dY}{dX} = \frac{3X^2 + A}{2\sqrt{X^3 + AX + B}} = \frac{3X^2 + A}{2Y} \\ \Rightarrow \quad & \lambda = \frac{3x_P^2 + A}{2y_P}. \end{aligned}$$
3. $P = -Q$. In questo caso poniamo $P + Q = P - P = \infty$.

Abbiamo così trovato la legge di gruppo (*group law*):

Definizione B.8 (Legge di gruppo). Sia $E(\mathbb{K})$ una curva ellittica $E : Y^2 = X^3 + AX + B$ su un campo $\mathbb{K}$, e siano $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ due punti di E diversi dal punto all'infinito. Definiamo $P + Q = R = (x_R, y_R)$ come segue:

$$\lambda = \begin{cases} \dfrac{y_Q - y_P}{x_Q - x_P} & \text{se } P \neq Q \\ \dfrac{3x_P^2 + A}{2y_P} & \text{se } P = Q \end{cases}$$
$$x_R = -x_P - x_Q + \lambda^2$$
$$y_R = -y_P - (x_R - x_P)\lambda.$$

Quando $P = -Q$ poniamo $P + Q = \infty$. ■

Consideriamo ad esempio la curva $E : Y^2 = X^3 + X + 1$, con $\mathbb{K} = \mathbb{R}$ e $P = (0, 1)$. Notare che $\Delta_E = 4A^3 + 27B^2 = 31 \neq 0$ e quindi E è non-singolare. In effetti è semplice verificare che P è un punto di E. Calcoliamo $R = P + P$:

$$\lambda = \frac{3x_P^2 + A}{2y_P} = \frac{0+1}{2} = \frac{1}{2}$$
$$x_R = -x_P - x_P + \lambda^2 = \frac{1}{4}$$
$$y_R = -y_P - (x_R - x_P)\lambda = -\frac{9}{8}$$
$$\Rightarrow \quad P + P = R = \left(\frac{1}{4}, -\frac{9}{8}\right).$$

È immediato verificare che R è un punto di $E(\mathbb{R})$.

Il punto chiave, a prima vista sorprendente, è che l'insieme

$$E(\mathbb{K}) = \left\{(x, y) \in \mathbb{K} \times \mathbb{K} : \ y^2 = x^3 + A\, x + B\right\} \cup \{\infty\}$$

dei punti di $E(\mathbb{K})$ più il punto all'infinito, è un gruppo algebrico $(E(\mathbb{K}), +)$ (cf. Definizione B.2) rispetto all'operazione di somma della Definizione B.8.

Curve ellittiche su $\mathbb{Z}_p$. Le principali applicazioni delle curve ellittiche in crittografia hanno a che fare con curve ellittiche definite su un campo finito, ad esempio $\mathbb{K} = \mathbb{Z}_p$. Possiamo adattare facilmente la Definizione B.8 a questo caso particolare.

Definizione B.9 (Curva ellittica su $\mathbb{Z}_p$). Sia $p \neq 2, 3$. Una curva ellittica su $\mathbb{Z}_p$, indicata con $E(\mathbb{Z}_p)$, è definita da un'equazione della forma:

$$E: \; Y^2 \equiv X^3 + AX + B \pmod{p} \qquad A, B \in \mathbb{Z}_p.$$

■

La curva E è detta non-singolare se non ha zeri doppi, ovvero se il discriminante $\Delta_E = 4A^3 + 27B^2 \not\equiv 0 \pmod{p}$. Possiamo anche adattare la legge di gruppo di Definizione B.8:

Definizione B.10 (Legge di gruppo per $E(\mathbb{Z}_p)$). Sia $E(\mathbb{Z}_p)$ una curva ellittica $E : Y^2 \equiv X^3 + AX + B \pmod{p}$ sul campo $\mathbb{Z}_p$, e siano $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ due punti di E diversi dal punto all'infinito. Definiamo $P + Q = R = (x_R, y_R)$ come segue:

$$\begin{aligned} \lambda &\equiv \begin{cases} \dfrac{y_Q - y_P}{x_Q - x_P} \pmod{p} & \text{se } P \neq Q \\ \dfrac{3x_P^2 + A}{2y_P} \pmod{p} & \text{se } P = Q \end{cases} \\ x_R &\equiv -x_P - x_Q + \lambda^2 \pmod{p} \\ y_R &\equiv -y_P - (x_R - x_P)\lambda \pmod{p}. \end{aligned} \tag{B.3}$$

Nel caso in cui $P = -Q$ poniamo $P + Q = \infty$. ■

Anche nel caso $\mathbb{K} = \mathbb{Z}_p$, infine, l'insieme

$$E(\mathbb{Z}_p) = \left\{(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p : y^2 \equiv x^3 + A\ x + B \pmod{p}\right\} \cup \{\infty\},$$

è un gruppo algebrico rispetto all'operazione di somma della Definizione B.10. Inoltre tale gruppo è finito, come mostrato dal seguente:

Teorema B.24 (Teorema di Hasse). *Sia $E : Y^2 = X^3 + AX + B$ una curva ellittica su $\mathbb{Z}_p$, con p un primo. Allora:*

$$\#E(\mathbb{Z}_p) = p + 1 - t,$$

per $|t| < 2\sqrt{p}$.

Il primo algoritmo efficiente per calcolare $\#E(\mathbb{Z}_p)$ è dovuto a Schoof [Sch85].

Concludiamo discutendo brevemente della complessità computazionale associata alla manipolazione e alla rappresentazione di una curva ellittica su un

campo finito. Definire una curva ellittica su $\mathbb{Z}_p$ equivale a scegliere A e B in $\mathbb{Z}_p$, quindi l'occupazione di memoria è $2 \log p = O(\log p)$. Un discorso identico vale per lo spazio necessario a memorizzare un punto P (due coordinate). Il costo dell'addizione è dominato dalla divisione nel calcolo di λ, quindi $O(\log^3 p)$ se ci riferiamo all'algoritmo euclideo (cf. Corollario B.5). Per calcolare un punto P di $E(\mathbb{Z}_p)$ si deve scegliere un valore $x \in \mathbb{Z}_p$, calcolare $\alpha \equiv x^3 + Ax + B \pmod{p}$ e sperare che α sia un residuo quadratico modulo p (il che è facilmente verificabile attraverso il criterio di Eulero, cf. Lemma B.16). Infine, è necessario estrarre la radice, il che può sempre essere fatto efficientemente in $\mathbb{Z}_p$ (cf. Appendice B.3).

Esercizi

Esercizio B.1. Determinare la complessità degli algoritmi elementari di somma, moltiplicazione e divisione in funzione della dimensione degli input.

Esercizio B.2. Usare l'algoritmo di Euclide per calcolare $\gcd(841, 294)$. Quindi, esprimere il risultato come combinazione lineare dei due numeri usando l'algoritmo di Euclide esteso.

Esercizio B.3. Calcolare $\gcd(X^4 - 4X^3 + 6X^2 - 4X + 1, X^3 - X^2 + X - 1)$.

Esercizio B.4. Risolvere la congruenza $3X \equiv 4 \pmod 7$.

Esercizio B.5. Usando il teorema del resto cinese, risolvere il seguente sistema di congruenze:

$$\begin{cases} X \equiv 2 \pmod 3 \\ X \equiv 3 \pmod 5 \\ X \equiv 4 \pmod{11} \\ X \equiv 5 \pmod{16}. \end{cases}$$

Esercizio B.6. Dimostrare che $\mathbb{Z}_{56}$ non è un gruppo rispetto all'operazione di moltiplicazione.

Esercizio B.7. Quanti e quali sono gli elementi di $\mathbb{Z}_{19}^*$? Il gruppo è ciclico? In caso affermativo dire quanti generatori ci sono, altrimenti trovare il sottogruppo di $\mathbb{Z}_{19}^*$ con ordine più grande.

Esercizio B.8. Calcolare 7^{120007} mod 143 a mano.

Esercizio B.9. Calcolare $J_{167}(91)$.

Esercizio B.10. Risolvere $X^2 \equiv 100 \pmod{231}$.

Esercizio B.11. Considerare il gruppo $\mathbb{Z}_{35}^*$.

1. Quanti elementi ci sono? Elencarli.
2. Per ogni elemento del gruppo, decidere se ha simbolo di Jacobi $+1$ oppure -1. Quanti sono gli elementi con simbolo di Jacobi $+1$.
3. Per ogni elemento con simbolo di Jacobi $+1$, determinare se l'elemento è un residuo quadratico oppure no. Quanti elementi con simbolo di Jacobi $+1$ sono residui quadratici?
4. Per ogni residuo quadratico, trovare le corrispondenti radici quadrate.
5. Supponiamo di utilizzare RSA in $\mathbb{Z}_{35}$ con $e = 7$. Qual è il valore di d?

Esercizio B.12. Sia $E : Y^2 = X^3 + 1$ su $\mathbb{Z}_5$. Verificare che si tratta di una curva ellittica. Determinare l'ordine del punto $(2, 2)$ nel gruppo $E(\mathbb{Z}_5)$.

Esercizio B.13. Consideriamo la curva $E : Y^2 \equiv X^3 + X + 2 \pmod{p}$ per $p = 5$. Elencare i punti di $E(\mathbb{Z}_5)$. Sia $P = (1, 2)$; calcoliamo $P + P = 2P$. Infine, mostrare che $E(\mathbb{Z}_5)$ è ciclico e che P è un generatore.

Letture consigliate

[BZ10] Richard P. Brent e Paul Zimmermann. "An $O(M(n)\log n)$ Algorithm for the Jacobi Symbol". In: *ANTS*. 2010, pp. 83–95.

[ES98] Shawna Meyer Eikenberry e Jonathan Sorenson. "Efficient Algorithms for Computing the Jacobi Symbol". In: *J. Symb. Comput.* 26.4 (1998), pp. 509–523.

[JJ05] Gareth A. Jones e Josephine M. Jones. *Elementary Number Theory*. Springer-Verlang, 2005.

[Lan02] Serge Lang. *Algebra*. Springer-Verlang Berlin, 2002.

[Sch85] René Schoof. "Elliptic Curves over Finite Fields and the Computation of Square Roots mod$p$". In: *Math. Comp. 44(170)* (1985), pp. 483–494.

[Sil86] Joseph H. Silverman. *Arithmetic of Elliptic Curves*. Springer-Verlang, 1986.

[Sin97] Simon Singh. *Fermat's Last Theorem: The Story of a Riddle that Confounded the World's Greatest Minds*. Fourth Estate, 1997.

[Ven09] Daniele Venturi. *Introduction to Algorithmic Number Theory*. Rapp. tecn. ECCC TR09-62. SAPIENZA University of Rome, 2009. URL: `http://eccc.hpi-web.de/report/2009/062/`.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Seconda edizione. Chapman & Hall (CRC Press), 2008.

[Wil95] Andrew Wiles. "Modular elliptic curves and Fermat's Last Theorem". In: *Annals of Mathematics* 142 (1995), pp. 443–551.

APPENDICE

C

Problemi computazionali

«Quanto fa uno più uno più uno più uno più uno più uno più uno più uno più uno più uno?». «Non saprei», disse Alice. «Ho perso il conto». «Non è in grado di fare l'addizione», disse la Regina Rossa.

Lewis Carroll, Attraverso lo Specchio e quel che Alice vi trovò [Car71]

Come abbiamo avuto modo di vedere in diverse occasioni, l'ipotesi di base su cui si regge tutta la teoria della crittografia moderna è l'esistenza di alcuni problemi computazionali ritenuti in qualche modo "difficili" da risolvere (in determinate condizioni). Questa appendice offre una panoramica su alcuni di questi problemi, discutendo la loro origine e studiando alcuni degli algoritmi più efficienti conosciuti per attaccarli.

Guida per il lettore. Il primo problema computazionale di cui ci occuperemo (nel Paragrafo C.1) è quello di determinare se un dato numero è primo oppure no. Ciò è molto importante nel contesto di alcuni crittosistemi a chiave pubblica, ad esempio il cifrario RSA (cf. Paragrafo 6.2). Come vedremo, in realtà, questo problema non è un problema difficile, in quanto è sempre risolvibile in tempo polinomiale.

Nel Paragrafo C.2 passeremo a studiare alcuni algoritmi per fattorizzare interi della forma $p \cdot q$, dove p e q sono primi di grande dimensione (diciamo $\approx$ 50 cifre decimali).

Nel Paragrafo C.3, infine, studieremo il problema del logaritmo discreto, che richiede, dato un elemento g^x in un gruppo finito $\mathbb{G}$, di calcolare l'esponente x.

Per una trattazione esaustiva si rimanda a [CP01]. Una panoramica si trova anche in [Sch08; Ste08; Ven09].

Venturi D.: Crittografia nel Paese delle Meraviglie
DOI 10.1007/978-88-470-2481-6_17, © Springer-Verlag Italia 2012

C.1 Test di primalità

Un numero intero $p > 1$ è detto *primo* se p ha come divisori solamente 1 e sé stesso. Lo studio della differenza tra numeri primi e numeri composti (ovvero numeri che sono esprimibili come prodotto di primi) sembra avere origini molto antiche. Euclide è stato il primo a mostrare che ogni numeri intero positivo è esprimibile come prodotto di primi, oppure è primo esso stesso.

Teorema C.1 (Euclide, Libro VII Proposizioni 31 e 32). *Ogni numero intero maggiore di* 1 *è esprimibile come prodotto di primi oppure è esso stesso un primo.*

Dimostrazione. Per induzione. Supponiamo che l'asserto sia vero per tutti gli interi inferiori ad n. Se n è primo non c'è niente da dimostrare. Se n non è primo, esistono due interi n_1, n_2 tali che $n = n_1 \cdot n_2$, con $n_1, n_2 < n$. Ma per l'ipotesi induttiva n_1 e n_2 devono essere esprimibili come prodotto di primi (oppure essere loro stessi primi); sostituendo tali espressioni, otteniamo che anche n è esprimibile come prodotto di primi. □

Quanti sono i numeri primi? La risposta a questa domanda è stata data ancora da Euclide.

Teorema C.2 (Euclide, Libro IX Proposizione 20). *Ci sono infiniti numeri primi.*

Dimostrazione. Per assurdo supponiamo che l'insieme dei primi sia finito e pari a $\{p_1, \ldots, p_k\}$. Consideriamo il numero:

$$n = \prod_{i=1}^{k} p_i + 1.$$

Tale numero non è divisibile per nessuno dei numeri $p_1, \ldots, p_k$. D'altro canto n deve essere divisibile per qualche primo oppure essere primo a sua volta (per il Teorema C.1). Dunque esiste un primo p_{k+1} diverso da $p_1, \ldots, p_k$ e ci sono infiniti primi. □

Gauss è stato il primo a mostrare che la scomposizione in fattori primi di un numero intero è unica.

Teorema C.3 (Teorema fondamentale dell'aritmetica). *Ciascun numero naturale è esprimibile in uno ed un sol modo come prodotto di primi.*

Dimostrazione. Per induzione. Supponiamo che l'asserto sia valido per ogni numero naturale strettamente minore di n. Se n è esso stesso primo non c'è niente da dimostrare. Sia allora n composto e supponiamo che esso ammetta due scomposizioni distinte in fattori primi, vale a dire:

$$n = p \cdot q \cdot r \cdots = p' \cdot q' \cdot r' \cdots,$$

dove $p, q, r, \ldots$ e $p', q', r', \ldots$ sono tutti primi. Osserviamo che nessun primo può comparire in entrambe le scomposizioni in quanto, se così fosse, potrebbe essere cancellato generando due scomposizioni distinte di un numero minore di n, contro l'ipotesi induttiva.

Senza perdita di generalità supponiamo che p e p' siano i primi più piccoli delle due scomposizioni. Siccome n è composto, abbiamo $n \geq p^2$ ed $n \geq (p')^2$; ma p e p' sono distinti e quindi una delle due uguaglianze deve essere vera in senso stretto, ovvero $p \cdot p' < n$. Consideriamo ora il numero $n - p \cdot p'$; siccome tale numero è minore di n, deve ammettere un'unica scomposizione in fattori primi. È evidente d'altra parte che sia p che p' dividono $n - p \cdot p'$, in quanto entrambi dividono n e $p \cdot p'$. Di conseguenza:

$$n - p \cdot p' = p \cdot p' \cdot q'' \cdot r'' \cdots,$$

dove $q'', r'', \ldots$ sono tutti primi. Segue che $p \cdot p'$ deve essere un fattore di n, il che è impossibile, in quanto abbiamo già osservato che i primi $p, q, r, \ldots$ e $p', q', r', \ldots$ devono essere distinti. Concludiamo che la fattorizzazione di n è unica. □

Fissiamo un intero n e chiediamoci quanti sono i primi minori o uguali ad n. Il teorema dei numeri primi (*Prime Number Theorem*, PNT) afferma che per n abbastanza grande ci sono circa $n/\ln n$ primi in $[1, n]$.

Teorema C.4 (Teorema dei numeri primi). *Sia $\pi(n)$ il numero di primi strettamente minori di n. Abbiamo:*

$$\lim_{n \to \infty} \frac{\pi(n) \ln n}{n} = 1.$$

La prova di questo fatto è complessa ed esula dagli scopi del testo. In generale domande di questo tipo vengono studiate in teoria analitica dei numeri. Si veda [Apo76] per un'introduzione.

Il compito di stabilire se un dato intero è un primo oppure no, è molto importante in crittografia. Nel seguito discuteremo due algoritmi: il test di

Miller-Rabin ed il test di Agrawal, Kayal e Saxena. Il primo algoritmo è probabilistico; in particolare non è escluso che un numero composto passi il test venendo identificato come primo (tuttavia ciò avviene solo con bassa probabilità). Il secondo algoritmo è stato il primo test di primalità ad essere contemporaneamente deterministico e polinomiale (mostrando così che il problema di stabilire la primalità di un numero è in **P**). Un altro algoritmo molto importante (basato sulle curve ellittiche) è dovuto a Goldwasser e Killian [GK86] ed è stato poi migliorato da Atkin e Morain [AM93]; seppure il test sia deterministico, l'analisi della complessità è probabilistica (ma si congettura che sia polinomiale). Distingueremo test probabilistici (in cui la risposta è corretta con alta probabilità) e test deterministici (in cui la risposta è sempre corretta).

Il test di Miller e Rabin. L'algoritmo di Miller-Rabin è probabilistico: esso è in grado di stabilire con elevata probabilità (ed in tempo polinomiale) se un dato numero n *non* è primo. D'altro canto, se n passa il test, non possiamo essere certi che esso sia primo. È necessario quindi ripetere l'algoritmo diverse volte, rendendo così trascurabile la probabilità che un numero composto non sia riconosciuto come tale. L'idea originale è dovuta ad Artjuhov[Art67]. Rabin[Rab80] ha proposto la versione probabilistica dell'algoritmo; indipendentemente Miller [Mil76] ha mostrato che, sotto alcune ipotesi, il test può essere trasformato in deterministico.

In un certo senso il test di Miller-Rabin può essere visto come una generalizzazione del seguente *test di Fermat*. Per il piccolo teorema di Fermat (cf. Teorema B.11), se p è un primo, allora per ogni $x \in \mathbb{Z}_p^*$, abbiamo $x^{p-1} \equiv 1 \pmod{p}$. Pertanto, dato un numero n da testare, potremmo pensare di scegliere $x \in \mathbb{Z}_n^*$ e controllare se per caso $x^{n-1} \equiv 1 \pmod{n}$. Il problema di questa strategia è che il piccolo teorema di Fermat fornisce solo una condizione necessaria per la primalità di n; esistono infatti numeri — detti *pseudo-primi di Fermat* in base x — tali che $x^{n-1} \equiv 1 \pmod{n}$, anche quando n è composto (cf. Esercizio C.1). In realtà c'è di peggio: esistono numeri — detti *numeri di Carmichael* — che soddisfano la relazione di cui sopra per *qualsiasi* valore di $x \in \mathbb{Z}_n^*$, indipendentemente dal fatto che n sia primo o composto (cf. Esercizio C.2). L'idea di Miller-Rabin è quella di aumentare la probabilità di successo del test di Fermat.

Lemma C.5 (Radici dell'unità modulo p). *Sia $p > 2$ un primo. Non esistono radici non-banali di 1 modulo p.*

Dimostrazione. Osserviamo che sia 1 che -1 restituiscono sempre 1 quando sono elevati al quadrato modulo p; queste sono le radici banali dell'unità. Supponiamo che $x \in \mathbb{Z}_p$ sia una radice non-banale dell'unità modulo p. Allora

$$x^2 \equiv 1 \pmod{p} \quad \Rightarrow \quad (x+1)(x-1) \equiv 0 \pmod{p}.$$

Siccome x è non-banale, abbiamo $x \not\equiv \pm 1 \pmod{p}$, vale a dire $x+1$ ed $x-1$ sono coprimi con p, ovvero né $x+1$ né $x-1$ sono divisibili per p. Ma allora p non può neanche dividere il prodotto tra $x+1$ ed $x-1$ e possiamo concludere:

$$(x+1)(x-1) \not\equiv 0 \pmod{p},$$

che è una contraddizione. Concludiamo che le uniche radici dell'unità modulo p sono quelle banali. □

Lemma C.6 (Algoritmo di Miller-Rabin). *Sia n un primo dispari. Scriviamo $n-1 = 2^s d$, con d un intero dispari ed $s \geq 1$. Per ogni $x \in \mathbb{Z}_n^*$ si ha che $x^d \equiv 1 \pmod{n}$ oppure $x^{2^r d} \equiv -1 \pmod{n}$ per qualche $0 \leq r < s$.*

Dimostrazione. Siccome n è primo, il piccolo teorema di Fermat ci dice che $x^{n-1} \equiv 1 \pmod{n}$, ovvero n è congruo all'unità in $\mathbb{Z}_n^*$. Se ora calcoliamo ripetutamente la radice quadrata di x^{n-1}, il Lemma C.5 ci dice che otterremo sempre $+1$ oppure -1. Quando il risultato è -1, la seconda uguaglianza è verificata.

Supponiamo d'altra parte di aver continuato ad estrarre le radici quadrate di x^{n-1} senza che la seconda uguaglianza sia mai verificata; ciò equivale a dire che $x^{2^r d} \not\equiv -1 \pmod{p}$ per ogni $r = 0, 1, \cdots, s-1$. Notare che siccome anche x^d è una radice quadrata dell'unità, deve aversi $x^d \equiv \pm 1 \pmod{n}$. Comunque per $r = 0$ abbiamo ottenuto $x^d \not\equiv -1 \pmod{n}$, quindi se la seconda uguaglianza non vale deve valere la prima. □

Analogamente al caso dei numeri pseudo-primi in base x, possiamo definire i numeri *pseudo-primi forti* in base x come gli interi dispari $n > 3$ per cui la condizione del Lemma C.6 è soddisfatta. Se scegliamo a caso $x \in \mathbb{Z}_n^*$ e la condizione del Lemma C.6 non è verificata, possiamo concludere subito che n è composto. D'altro canto, se tale condizione vale per un dato x, non possiamo concludere che n è primo, in quanto n potrebbe essere uno pseudo-primo forte in base x. La speranza è che stavolta la frazione di numeri composti che passino il test sia più bassa. In effetti posto

$$\mathcal{B} = \left\{ x \in \mathbb{Z}_n^* : \; x^d = 1 \text{ oppure } x^{2^r d} = -1 \text{ per qualche } 0 \leq r < s \right\},$$

si può dimostrare che (per ogni intero dispari composto $n > 9$) si ha $(\#\mathcal{B})/\varphi(n) \leq 1/4$.

Questo fatto può essere usato per costruire il seguente test di primalità probabilistico. Si sceglie un valore a caso $x \in \mathbb{Z}_n^*$ e si verifica se $x \in \mathcal{B}$. Quando ciò accade si conclude che n è composto, altrimenti n è probabilmente primo. In una singola istanza la probabilità che $x \in \mathcal{B}$ per n composto è al più $1/4$; ripetendo il test k volte tale valore può essere ridotto a 4^{-k} e quindi reso piccolo a piacere.

Notare che per verificare che x sia un elemento di $\mathcal{B}$, bisogna elevare $x \in \mathbb{Z}_n^*$ ad un esponente non più grande di n. Ciò richiede $O(\log(n)(\log(n))^{\mu})$, con $\mu = 2$ se si fa riferimento agli algoritmi di moltiplicazione ordinaria in $\mathbb{Z}_n$ e $\mu = 1 + \varepsilon$ usando le tecniche di moltiplicazione veloce [Ber08].

Il test di Agrawal, Kayal e Saxena. L'algoritmo AKS [AKS04] (dalle iniziali dei suoi inventori) ha dimostrato per la prima volta che il compito di stabilire la primalità di un dato intero è in **P**. L'ingrediente chiave è contenuto nel seguente teorema, la cui dimostrazione esula dagli scopi del testo.

Teorema C.7 (Algoritmo AKS). *Sia $n > 0$ un intero dispari ed r un primo. Supponiamo che sia verificato quanto segue:*

(i) n non è divisibile per nessuno dei primi $\leq r$;

(ii) l'ordine di n in $\mathbb{Z}_r^*$ è:

$$\operatorname{ord}(n) \geq \left(\frac{\log(n)}{\log(2)}\right)^2;$$

(iii) per ogni $0 \leq a \leq r$, possiamo scrivere:

$$(X + a)^n = X^n + a \qquad \text{in } \mathbb{Z}_n[X]/\left(\frac{X^r - 1}{X - 1}\right).$$

Allora n è potenza di un primo.

Per un primo r, si definisce l'r-simo *polinomio ciclotomico*[93]:

$$\Phi_r(X) = \frac{X^r - 1}{X - 1} = X^{r-1} + \cdots + X^2 + X + 1.$$

[93] Tali polinomi sono connessi al problema di dividere un cerchio unitario in segmenti uguali, ovvero il problema di iscrivere poligoni regolari in un cerchio di raggio 1.

In questo modo, l'anello

$$\mathbb{Z}_n[X]/\left(\Phi_r(X)\right) = \left\{a_{r-2}X^{r-2} + \cdots + a_1 X + a_0 : \ a_i \in \mathbb{Z}_n\right\}$$

contiene tutti i polinomi di grado al più $r-2$ con coefficienti in $\mathbb{Z}_n$. Pertanto rappresentare un singolo elemento di tale anello richiede $O(r \log n)$ bit.

Osserviamo che quando n è primo i coefficienti del polinomio $(X+a)^n$ sono tutti nulli modulo n, quindi, usando il piccolo teorema di Fermat (cf. Teorema B.11), possiamo concludere:

$$(X+a)^n = \sum_{i=0}^{n} \binom{n}{i} X^{n-i} a^i = X^n + a^n = X^n + a \qquad \text{in } \mathbb{Z}_n[X].$$

Purtroppo valutare $(X+a)^n$ per ogni a è troppo costoso quando n è grande: l'idea di Agrawal, Kayal e Saxena è quella di calcolare $(X+a)^n$ modulo l'r-simo polinomio ciclotomico $\Phi_r(X)$.

Si può trasformare il Teorema C.7 in un test di primalità, come segue:

1. Controlla[94] che n non sia potenza di qualche intero.
2. Provando i valori $r = 2, 3, \ldots$ trova il più piccolo primo r che non divide n e che non divide nessuno degli elementi $n^i - 1$ (con $i = 0, 1, \ldots, \left(\frac{\log(n)}{\log(2)}\right)^2$). In simboli:
$$r \nmid n \cdot \prod_{i=1}^{\left(\frac{\log(n)}{\log(2)}\right)^2} (n^i - 1).$$
3. Controlla che la condizione (iii) del Teorema C.7 sia verificata.
4. Se n passa il Test, concludi che n è primo, altrimenti concludi che n è composto.

Dimostriamo che il test funziona. Banalmente se n è primo, passerà il test per il piccolo teorema di Fermat. Supponiamo d'altra parte che n passi il test.

[94] Tale controllo può essere effettuato alla seguente maniera. Sia n della forma $n = m^d$ con $d > 1$ ed $m \geq 2$, ovvero

$$n = m^d \geq 2^d \quad \Rightarrow \quad d \leq \frac{\log(n)}{\log(2)}.$$

Per $d = 2, 3, \ldots, \frac{\log(n)}{\log(2)}$ approssimiamo, con una certa precisione, il valore $n^{1/d}$ in $\mathbb{R}$ e poniamo $m = \text{round}(n^{1/d}) \in \mathbb{Z}$. Verifichiamo quindi che $m^d \neq n$. Il costo è $O(\log^4(n))$.

È sufficiente controllare che tutte le condizioni del Teorema C.7 siano rispettate per poter concludere che n sia primo (perché abbiamo già controllato nel punto (1) dell'algoritmo che esso non è potenza di un primo). La condizione (iii) è sicuramente verificata, in quanto essa coincide con il punto (3) del test; anche la condizione (i) è verificata, per come abbiamo definito r al punto (2). Sia infine o l'ordine di n modulo r, ovvero $n^o \equiv 1 \pmod{r}$ (il che ha senso in quanto $r \nmid n$); allora $r|(n^o - 1)$. Nel punto (2) dell'algoritmo abbiamo verificato che r non divide gli elementi $n^i - 1$ per $i = 0, 1, \ldots, \left(\frac{\log(n)}{\log(2)}\right)^2$, così che possiamo concludere:

$$o > \left(\frac{\log(n)}{\log(2)}\right)^2,$$

che è esattamente la condizione (ii) del Teorema C.7.

Si può dimostrare che $r \leq O(\log^5 n)$. La parte più costosa del punto di vista computazionale è quella in cui dobbiamo calcolare $(X + a)^n$ in $\mathbb{Z}_n[X]/(\Phi_r(X))$ per ogni $a = 0, \ldots, r - 1$. Siccome un elemento nell'anello è rappresentabile con $O(r \log n)$ bit, valutare l'n-sima potenza costa $O(\log n(r^2 \log^2 n))$ se ci si riferisce alle tecniche standard di moltiplicazione. Poiché dobbiamo ripetere per ogni possibile $a = 0, \ldots, r - 1$ la complessità nel caso peggiore sarà $O(r^3 \log^3 n) = O(\log^{18} n)$. Tale valore si abbassa a $O(\log^{12+\epsilon} n)$ se si fa riferimento alle tecniche di moltiplicazione veloce [Ber08].

C.2 Fattorizzazione di interi

Come abbiamo visto, il teorema fondamentale dell'aritmetica assicura che ogni numero intero sia scomponibile in fattori primi in modo univoco. È possibile in qualche modo trovare tale scomposizione? Quando il numero da fattorizzare è molto grande, non esiste (ad oggi) alcun algoritmo efficiente per risolvere il problema. Questo è il motivo per cui alcuni crittosistemi come RSA (cf. Paragrafo 6.2) sono basati sull'ipotesi che fattorizzare alcuni interi sia difficile.

Per testare la robustezza del cifrario RSA, vengono pubblicate periodicamente delle "sfide" in cui si richiede di fattorizzare un intero di certe dimensioni. L'ultimo numero fattorizzato [Kle+10] — della forma $p \cdot q$ con 232 cifre — ha richiesto circa due anni usando alcune tra le reti di super-computer più potenti al mondo (distribuite tra la Svizzera, il Giappone ed i Paesi Bassi).

Sia $p|n$ il più piccolo primo divisore di n (il numero che vogliamo fattorizzare). È ovvio che $p \leq \sqrt{n}$. L'idea più semplice è quella di scegliere un valore casuale $d < n$ e sperare che $\gcd(d, n) > 1$. Siccome la probabilità che il numero scelto

sia divisibile per p è $1/p$, e siccome il costo associato al calcolo del massimo comun divisore è $O(\log^3 p)$ (cf. Corollario B.5), possiamo concludere che la complessità è $O(p \log^3 n)$ che nel caso peggiore (ovvero quando $p \approx \sqrt{n}$) è già esponenziale in n.

Un'altra idea semplice è quella di cercare p semplicemente dividendo n per tutti i primi minori uguali della $\sqrt{n}$. Siccome dobbiamo eseguire tante divisioni quanti sono i primi minori o uguali a p — ovvero $\approx p/\log p$ per il teorema dei numeri primi (cf. Teorema C.4) — abbiamo una complessità $O(\log^2 n \cdot p/\log p)$ che è ancora esponenziale quando $p \approx \sqrt{n}$.

Nel resto di questo paragrafo discutiamo qualche algoritmo più sofisticato: l'algoritmo di Pollard $p-1$ e la sua estensione sulle curve ellittiche (detta ECM) dovuta a Lenstra.

Pollard $p-1$. L'idea alla base dell'algoritmo di Pollard è abbastanza semplice. Purtroppo come vedremo, esso consente di trovare un fattore non banale di n solo in casi molto "fortunati". Nonostante ciò la sua versione sulle curve ellittiche ha dato vita ad uno degli algoritmi più efficienti noti.

Sia n l'intero da fattorizzare. Fissiamo una costante B che rappresenti un limite per il numero di passi che siamo disposti a compiere. Calcoliamo

$$M = \prod_{\substack{p_i \leq B \\ p_i \text{ primo} \\ p_i^{e_i} \leq B}} p_i^{e_i}, \tag{C.1}$$

avendo indicato con e_i l'esponente del fattore primo p_i nella scomposizione di B. Ad esempio se $B = 20$, abbiamo $M = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$.

Una conseguenza del teorema dei numeri primi (cf. Teorema C.4) è che $M \approx e^B$. Fissiamo un valore casuale $x \in \mathbb{Z}_n^*$ e calcoliamo $y = x^M \bmod n$ e $\gcd(y - 1, n) = d$. La speranza è che $1 < d < n$, così che abbiamo trovato un divisore non banale d di n.

Sia ad esempio $n = 10001$ e $B = 10$, così che $M = 2^3 \cdot 3^2 \cdot 5 \cdot 7 = 2520$. Per $x = 2 \in \mathbb{Z}_{10001}^*$, abbiamo:

$$\begin{aligned} y = x^M = 2^{2520} &\equiv 3579 \pmod{10001} \\ \gcd(3579 - 1, 10001) &= 73. \end{aligned}$$

In effetti $10001 = 73 \cdot 137$.

Indichiamo con p il divisore primo minimale di n; l'algoritmo riesce a calcolare p se $p | \gcd(y - 1, n)$, ovvero se $p | (y - 1)$ poiché p è per definizione un

divisore di n. Quando ciò avviene

$$p|(y-1) \quad \Rightarrow \quad y \equiv 1 \pmod{p} \quad \Rightarrow \quad x^M \equiv 1 \pmod{p}.$$

Per il piccolo teorema di Fermat (cf. Teorema B.11), quindi, è sufficiente che M sia un multiplo di $p-1$, ovvero $p-1$ deve dividere M. Ciò è possibile solo se *tutti* i divisori primi di $p-1$ sono più piccoli di B (per costruzione di M), ovvero se $p-1$ è B-liscio (B-*smooth*).

Definizione C.1 (Numeri B-lisci). Sia $B > 0$ un numero reale e $n > 0$ un intero. Diremo che n è B-liscio, se tutti i divisori primi di n sono più piccoli di B. ■

Ad esempio 100 è 10-liscio, poiché $100 = 5^2 \cdot 2^2$ e $2, 5 < 10$.

Possiamo pertanto concludere che l'algoritmo $p-1$ di Pollard ha successo solo se $p-1$ è B-liscio. Il costo computazionale è dato dal calcolo di $y = x^M \bmod n$ e del massimo comun divisore, ovvero:

$$O\left(\log(M)\log^2(n) + \log^3(n)\right) = O\left(B\log^2(n)\right).$$

In pratica, se $B \approx n^{1/10}$, si può vedere che la probabilità che n abbia un divisore primo p tale che $p-1$ è B-liscio, è molto bassa; pertanto l'algoritmo funziona bene solo in casi fortunati. Ovviamente, più i valori di B sono grandi, maggiore è la probabilità che un numero sia B-liscio; d'altro canto la complessità è però esponenziale in $\log(B)$.

Più in generale, l'algoritmo funziona se $x^M \equiv 1 \pmod{p}$, mentre $x^M \not\equiv 1 \pmod{q}$ per ogni altro divisore primo q di n.

Il metodo delle curve ellittiche. Sebbene l'algoritmo Pollard $p-1$ sia poco efficiente, esso costituisce il cuore dell'idea di Lenstra [Len87], che può essere vista come una traduzione nel linguaggio delle curve ellittiche dell'algoritmo di Pollard. L'algoritmo è detto metedo delle curve ellittiche (*Elliptic Curve Method*, ECM).

Come abbiamo visto, l'algoritmo di Pollard funziona quando $p-1$ è B-liscio, dove $p-1 = \#\mathbb{Z}_p^*$. Nell'algoritmo ECM sostituiremo $\mathbb{Z}_p$ con $E(\mathbb{Z}_p)$, essendo E una curva ellittica non-singolare (cf. Definizione B.9). Analogamente a quanto accade nell'algoritmo di Pollard, il valore di p non è noto e tutti i calcoli saranno quindi eseguiti in $E(\mathbb{Z}_n)$ (al posto di $\mathbb{Z}_n^*$).

Supponiamo per semplicità che $n = pq$, dove p e q sono primi distinti maggiori di 3. Il teorema del resto cinese (cf. Teorema B.13) implica che

$$\begin{aligned} \mathbb{Z}_n &\simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \text{(come anelli)} \\ \mathbb{Z}_n^* &\simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^* \quad \text{(come gruppi)} \\ E(\mathbb{Z}_n) &\simeq E(\mathbb{Z}_p) \times E(\mathbb{Z}_q) \quad \text{(come gruppi)}. \end{aligned}$$

Pertanto l'insieme $E(\mathbb{Z}_n)$ eredita la struttura di gruppo abeliano: la maggior parte dei punti in $E(\mathbb{Z}_n)$ può essere addizionata usando le formule di Definizione B.10. Infatti la formula fallisce solo quando alcune quantità calcolate in $E(\mathbb{Z}_n)$ sono zero modulo p e diverse da zero modulo q (o viceversa). Quando ciò accade n, sarà fattorizzato.

Traduciamo l'algoritmo di Pollard usando il linguaggio delle curve ellittiche. Consideriamo la curva ellittica di equazione

$$E: \; Y^2 = X^3 + AX + B \qquad \gcd(n, 6) = 1 \text{ e } \gcd(\Delta_E, n) = 1,$$

con $\Delta_E = 4A^3 + 27B^2 \neq 0$. Notare che abbiamo richiesto che il discriminante Δ_E della curva non sia divisibile per nessuno dei divisori primi di n, ovvero che la curva E sia non-singolare[95] su $\mathbb{Z}_q$, per ogni divisore primo q di n. (Osserviamo che questo test preliminare è polinomiale, in quanto richiede solamente il calcolo di un massimo comun divisore. Inoltre se per caso $\gcd(\Delta_E, n) \neq 1$ abbiamo già trovato un fattore non banale di n.)

Scegliamo una costante B e calcoliamo il valore M in Eq. (C.1). Fissiamo quindi un punto a caso su $E(\mathbb{Z}_n)$ e proviamo a calcolare $[M]P = P + \cdots + P$ (M volte) in $E(\mathbb{Z}_n)$, il che corrisponde a calcolare $y = x^M$ nell'algoritmo di Pollard. Se ad un certo punto la formula per calcolare il coefficiente angolare λ in Eq. (B.3) fallisce (vale a dire se troviamo un valore non invertibile), allora abbiamo trovato un fattore di n. Ciò accade solo se

$$\begin{aligned} & [M]P = \infty \text{ in } E(\mathbb{Z}_p) \\ & [M]P \neq \infty \text{ in } E(\mathbb{Z}_q) \; \forall \text{ divisore primo } q \neq p \text{ di } n \\ \Leftrightarrow \quad & \#E(\mathbb{Z}_p) | M \quad \text{i. e. } \#E(\mathbb{Z}_p) \text{ è } B\text{-liscio}. \end{aligned}$$

[95] Siccome si può dimostrare che estrarre radici quadrate modulo n (quando n è composto) è equivalente a fattorizzare n (cf. Teorema 6.4), è necessario scegliere il punto P prima della curva E. Quindi, si sceglie prima $P = (x_P, y_P)$ ed un valore casuale per A; poi si calcola:

$$B = y_P^2 - x_P^3 - Ax_P.$$

Infine, si verifica che $\gcd(\Delta_E, n) = 1$.

Un esempio chiarisce immediatamente le idee. Sia $n = 35$ e supponiamo di usare $E : Y^2 = X^3 - X - 2$. È facile verificare che $\gcd(\Delta_E, 35) = 1$. Sia $M = 3$ e $P \in E(\mathbb{Z}_n)$, diciamo $P = (2, 2)$. Dobbiamo calcolare $[M]P$ in $E(\mathbb{Z}_n)$ e sperare che $[M]P = \infty$ in $E(\mathbb{Z}_p)$, ma $[M]P \neq \infty$ in $E(\mathbb{Z}_q)$ per ogni divisore primo $q \neq p$ di n. Ovviamente non conosciamo p, ma, se la condizione di cui sopra si verifica, è sufficiente calcolare $[M]P = [3]P = P + P + P$ in $E(\mathbb{Z}_n)$ per trovare un fattore di n. Cominciamo calcolando $[2]P$, usando le equazioni della Definizione B.10; abbiamo:

$$\lambda = \frac{3x_P^2 + A}{2y_P} = \frac{11}{4} = \frac{11 \cdot 9}{4 \cdot 9} \equiv 99 \equiv -6 \pmod{35}.$$

Quindi,

$$\begin{aligned} x_{2P} &= -x_P - x_P + \lambda^2 \equiv -3 \pmod{35} \\ y_{2P} &= -y_P - \lambda(x_{2P} - x_P) \equiv 3 \pmod{35}. \end{aligned}$$

Pertanto $[2]P = (-3, 3)$. Se ora tentiamo di calcolare $[3]P = 2P + P = (-3, 3) + (2, 2)$, otteniamo:

$$\lambda = \frac{3 - 2}{-3 - 2} = -\frac{1}{5},$$

e 5 non è invertibile in $\mathbb{Z}_{35}$. Abbiamo così trovato un fattore di $n = 5 \cdot 7$. È facile verificare che in effetti

$$\begin{aligned} [3]P &= (2, -2) + (2, 2) = \infty \text{ in } E(\mathbb{Z}_5) \\ [3]P &= (-3, 3) + (2, 2) \neq \infty \text{ in } E(\mathbb{Z}_7), \end{aligned}$$

e che $E(\mathbb{Z}_{35}) \simeq E(\mathbb{Z}_7) \times E(\mathbb{Z}_5)$.

Finora abbiamo solamente tradotto l'algoritmo di Pollard usando le curve ellittiche. Il punto di forza dell'algoritmo ECM sta nel fatto che ora però abbiamo un grado di libertà in più: se l'algoritmo fallisce per una data curva, possiamo semplicemente cambiare curva (senza incrementare B) e riprovare! (Ciò non è possibile in Pollard $p - 1$, in quanto $\mathbb{Z}_p^*$ è unico; se $p - 1$ non è B-liscio non resta altro da fare che aumentare B.)

Per stimare la complessità computazionale dell'algoritmo, bisogna stimare il numero di curve E che è necessario provare prima di avere successo. Ciò è possibile conoscendo: (i) la distribuzione del numero di curve ellittiche su $\mathbb{Z}_p$ in funzione del loro numero di punti e (ii) quanti sono i numeri B-lisci per un dato B. Si può dimostrare che questa analisi, nel caso in cui $p \approx \sqrt{n}$ fornisce una complessità $O(e^{\sqrt{\log(n) \log\log(n)}})$.

Tab. C.1. Algoritmi di fattorizzazione a confronto

Algoritmo	Complessità	Numero massimo di cifre
Divisione	$O\left(e^{\frac{1}{2}\log(n)}\right)$	1-15
ECM	$O\left(e^{\sqrt{\log(n)\log\log(n)}}\right)$	10-70
QS	$O\left(e^{\sqrt{\log(n)\log\log(n)}}\right)$	60-120
NFS	$O\left(e^{\log^{\frac{1}{3}}(n)(\log\log(n))^{\frac{2}{3}}}\right)$	120-200

Metodi di crivello. Gli algoritmi più sofisticati sono basati sui cosiddetti *metodi di crivello* che si basano su una vecchia idea già nota ai tempi di Fermat: si può sperare di calcolare un fattore di n se si riesce a scrivere n come differenza di due quadrati:

$$n = u^2 - v^2 = (u+v)(u-v).$$

Il crivello quadratico (*Quadratic Sieve*, QS) ed il crivello dei campi di numeri (*Number Field Sieve*, NFS) sono basati esattamente su questa osservazione. La Tab. C.1 mette a confronto gli algoritmi di fattorizzazione più importanti.

C.3 Il logaritmo discreto

Un altro problema centrale in crittografia è quello del logaritmo discreto. In questo paragrafo daremo una definizione formale del problema. Quindi studieremo un algoritmo semplice (ma non molto efficiente) per il calcolo di logaritmi discreti in $\mathbb{Z}_p^*$.

Logaritmi in $\mathbb{Z}_p^*$. Sia p un primo. Come abbiamo visto nell'Appendice B.2 il gruppo $\mathbb{Z}_p^*$ è ciclico e si chiama generatore un elemento g che ha ordine $p-1$ in $\mathbb{Z}_p^*$. Dato un tale generatore, ogni elemento $y \in \mathbb{Z}_p^*$ può essere espresso come $y = g^x$ per un qualche intero x.

Definizione C.2 (Logaritmo discreto in $\mathbb{Z}_p^*$). Con la notazione sopra introdotta, diremo che x è il logaritmo discreto di y rispetto alla base g e scriveremo $x = \log_g y$. ■

Osserviamo che per il piccolo teorema di Fermat (cf. Teorema B.11) possiamo sempre aggiungere multipli di $p-1$ all'esponente, ovvero:

$$y = g^x \equiv g^{x+k(p-1)} \pmod{p} \qquad \forall\ k \in \mathbb{Z}.$$

Segue che l'esponente $x = \log_g(y)$ è sempre un elemento di $\mathbb{Z}_{p-1}$.

È facile convincersi che il logaritmo discreto soddisfa le stesse proprietà del logaritmo canonico sui reali. In particolare se $y_1 = g^{x_1}$ ed $y_2 = g^{x_2}$, abbiamo $y_1 \cdot y_2 = g^{x_1+x_2}$ e quindi

$$\log_g(y_1 \cdot y_2) = x_1 + x_2 = \log_g(y_1) + \log_g(y_2).$$

D'altra parte, sia $h \neq g$ un altro generatore di $\mathbb{Z}_p^*$. Possiamo sempre scrivere $g = h^c$ per qualche $c \in \mathbb{Z}_{p-1}$. Dato allora un elemento $y = g^x = h^{cx}$, si ha:

$$\log_h(y) = cx = c \log_g(y) \quad \Rightarrow \quad \frac{\log_h(y)}{\log_g(y)} = c \in \mathbb{Z}_{p-1}^*,$$

ovvero è possibile cambiare la base del logaritmo da g ad h moltiplicando per una costante c che dipende solo da g ed h. Osserviamo infine che se g è un generatore di $\mathbb{Z}_p^*$, allora necessariamente $\log_g(-1) = \frac{p-1}{2}$, poiché g è un non-residuo quadratico modulo p.

Passi-nani e passi-giganti. Sia p un primo e g un generatore di $\mathbb{Z}_p^*$; dato un elemento $y = g^x \in \mathbb{Z}_p^*$ vogliamo calcolare $x \in \mathbb{Z}_{p-1}$. Il modo banale di fare ciò è l'attacco a forza bruta: calcoliamo le potenze successive di g — vale a dire $g^2, g^3, \ldots$ — fino a che non ritroviamo y e quindi x. La complessità è ovviamente $O(e^{\log p})$, esponenziale nella dimensione di p.

Il seguente algoritmo — detto algoritmo passi-nani passi-giganti (*Baby-Step Giant-Step*, BSGS) e dovuto a Shanks — consente di migliorare le prestazioni. (Lo stesso algoritmo può essere utilizzato per calcolare il numero di punti di una curva ellittica su $\mathbb{Z}_p^*$.) Si sceglie una costante $B = \lfloor \sqrt{p} \rfloor + 1$ e si scrive l'espansione in base B di x, ovvero $x = c_0 + c_1 B$. Siccome $x < B^2$ per costruzione, deve aversi $0 \leq c_0, c_1 \leq B$. Si eseguono quindi i passi-nani, calcolando i valori

$$y, yg^{-1}, \ldots, yg^{-B},$$

e memorizzando il risultato in una lista. Si eseguono poi i passi-giganti, calcolando i valori

$$g^B, (g^B)^2, \ldots, (g^B)^B,$$

e memorizzando il risultato in una seconda lista.

Siccome $y = g^x = g^{c_0} \cdot g^{c_1 B}$, basta cercare l'elemento c_0 nella prima lista e l'elemento c_1 nella seconda lista. In altre parole, siccome

$$yg^{-c_0} = (g^B)^{c_1} \qquad 0 \leq c_0, c_1 \leq B,$$

dobbiamo semplicemente cercare un elemento che sia *uguale nelle due liste*.

Il modo migliore per eseguire questa ricerca è quello di organizzare le liste in modo efficiente, ad esempio usando una tabella hash (*hash table* in inglese).[96] In questo modo la ricerca nelle due liste richiede tempo costante $O(1)$ e la complessità è dominata dalla dimensione della costante B:

$$O\left(\sqrt{p}\log^2(p)\right) = O\left(e^{\frac{1}{2}\log(p)}\log^2(p)\right).$$

La complessità è ancora esponenziale (ma comunque più bassa che nell'attacco a forza bruta).

Il metodo del calcolo dell'indice. Il calcolo dell'indice (*Index Calculus*) è il miglior algoritmo conosciuto per risolvere il problema del logaritmo discreto. Non daremo una descrizione di questo algoritmo. La complessità comunque è $O(e^{\sqrt{3\log(p)\log\log(p)}})$. È un problema aperto [SS98] quello di tradurre l'algoritmo usando il linguaggio delle curve ellittiche (il che potenzialmente potrebbe portare un notevole vantaggio in termini di complessità).

[96] Una tabella hash, è una struttura dati che usa una funzione hash (cf. Capitolo 4) per organizzare i dati in modo efficiente. Questa metodologia consente di ottenere tempo di accesso alla lista costante ed indipendente dalla dimensione della lista stessa [Knu98].

Esercizi

Esercizio C.1. Mostrare che 341 è uno pseudo-primo di Fermat in base 2, ma non in base 3. Mostrare che 91 è uno pseudo-primo di Fermat in base 3, ma non in base 2.

Esercizio C.2. Il seguente esercizio spiega il criterio di Korselt, che caratterizza completamente i numeri di Carmichael: un intero n è un numero di Carmichael se e solo se n è positivo, composto, privo di quadrati (i.e., per ogni primo p divisore di n si ha che p^2 non divide n) e per ogni primo p divisore di n si ha che $p-1|n-1$. Rispondere alle seguenti domande:

1. Usare il criterio di Korselt per dimostrare che non esiste un numero di Carmichael con solamente due fattori primi p_1, p_2.
2. Applicando il criterio di Korselt costruire un numero di Carmichael che abbia tre fattori primi p_1, p_2, p_3.
3. Dimostrare il criterio di Korselt.

Esercizio C.3. Verificare se il numero 341 è primo usando il test di Fermat ed il test di Miller-Rabin.

Esercizio C.4. Fattorizzare il numero 1001, usando il metodo $p-1$ ed il metodo delle curve ellittiche.

Esercizio C.5. Calcolare $\log_{11} 5$ in $\mathbb{Z}_{31}$ e $\log_5 27$ in $\mathbb{Z}_{103}$, usando l'algoritmo di Shanks.

Esercizio C.6. Alice e Bob usano un sistema di scambio chiavi di Diffie-Hellman basato sulla curva ellittica $E: Y^2 \equiv X^3 + 3X + 12 \pmod{23}$ che ha 21 punti. Pubblicano anche il punto $P = (2, 7)$. Successivamente Alice invia il messaggio $A = aP = (7, 10)$ e Bob invia il messaggio $B = bP = (9, 3)$.

1. Elencare i punti della curva.
2. Calcolare l'ordine del gruppo generato dal punto P.
3. Calcolare il numero segreto b usando l'algoritmo di Shanks.
4. Calcolare la chiave di sessione.

Letture consigliate

[AKS04] Manindra Agrawal, Neeraj Kayal e Nitin Saxena. "PRIMES is in **P**". In: *Annals of Mathematics* 160 (2004), pp. 781–793.

[AM93] Arthur O. L. Atkin e Francois Morain. "Elliptic Curves and Primality Proving". In: *Math. Comp. 61* (1993), pp. 29–68.

[Apo76] Tom M. Apostol. *Introduction to Analytic Number Theory*. Springer, 1976.

[Art67] M. M. Artjuhov. "Certain Criteria for Primality of Numbers Connected with the Little Fermat Theorem". In: *Acta Arith. 12* (1967), pp. 355–364.

[Ber08] Daniel J. Bernstein. *Fast Multiplication and its Applications, pp. 325–384 in Surveys in Algorithmic Number Theory*. J. P. Buhler and P. Stevenhagen, Math. Sci. Res. Inst. Publ. 44. Cambridge University Press, New York, 2008.

[Car71] Lewis Carroll. *Through the Looking-Glass, and What Alice Found There*. Edizione Italiana Einaudi. Traduzione di Alessandro Ceni. Mac Millan & Co., 1871.

[CP01] Richard Crandall e Carl Pomerance. *Prime Numbers: A Computational Perspective*. Springer, 2001.

[GK86] Shafi Goldwasser e Joe Kilian. "Almost All Primes Can Be Quickly Certified". In: *STOC*. 1986, pp. 316–329.

[Kle+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev e Paul Zimmermann. "Factorization of a 768-Bit RSA Modulus". In: *CRYPTO*. 2010, pp. 333–350.

[Knu98] Donald Knuth. *The Art of Computer Programming 3: Sorting and Searching*. Addison-Wesley, 1998.

[Len87] Hendrik W. Lenstra. "Factoring Integers with Elliptic Curves". In: *Annals of Mathematics* 126 (1987), pp. 649–673.

[Mil76] Gary L. Miller. "Riemann's Hypothesis and Tests for Primality". In: *JCSS* 13.3 (1976), pp. 300–317.

[Rab80] Michael O. Rabin. "Probabilistic Algorithms for Testing Primality". In: *J. Number Theory* 12 (1980), pp. 128–138.

[Sch08] René Schoof. *Four Primality Testing Algorithms, pp. 101–126 in Surveys in Algorithmic Number Theory*. J. P. Buhler and P. Stevenhagen, Math. Sci. Res. Inst. Publ. 44. Cambridge University Press, New York, 2008.

[SS98] Joseph H. Silverman e Joe Suzuki. "Elliptic Curve Discrete Logarithms and the Index Calculus". In: *ASIACRYPT*. 1998, pp. 110–125.

[Ste08] Peter Stevenhagen. *The Number Field Sieve, pp. 83–100 in Surveys in Algorithmic Number Theory*. J. P. Buhler and P. Stevenhagen, Math. Sci. Res. Inst. Publ. 44. Cambridge University Press, New York, 2008.

[Ven09] Daniele Venturi. *Introduction to Algorithmic Number Theory*. Rapp. tecn. ECCC TR09-62. SAPIENZA University of Rome, 2009. URL: `http://eccc.hpi-web.de/report/2009/062/`.

Glossario

B-smooth: B-liscio. 482

Auxiliary Input Zero Knowledge: conoscenza nulla con input asiliario. 356

Baby-Step Giant-Step: passi-nani passi-giganti. 486

Bilinear Decisional Diffie-Hellman problem: problema decisionale bilineare di Diffie-Hellman. 267

blind signature: firma cieca. 411

Bounded Decoding Distance: decodifica a distanza limitata. 240

Bounded-Retrieval Model: modello della capacità limitata. 35

Bounded-Storage Model: modello dello spazio di memoria limitato. 35

broadcast: radiodiffusione circolare. 35, 408

Certificate Revocation List: lista di certificati revocati. 261

Certification Authority: autorità di certificazione. 258

chain rule: regola della catena. 437

Chinese Remainder Theorem: teorema del resto cinese. 454

Chosen Chipertext Attack: attacco a crittotesto scelto. 106

Chosen Plaintext Attack: attacco a messaggio scelto. 105

client: cliente. 336

Closest Vector Problem: problema del vettore più vicino. 238

commitment: impegno. 398

Common Reference String: stringa comune di riferimento. 370

Computational Diffie-Hellman problem: problema computazionale di Diffie-Hellman. 161

Decisional Diffie-Hellman problem: problema decisionale di Diffie-Hellman. 161

denial of service: interruzione del servizio. 283

e-cash: denaro digitale. 9

eavesdropping: spionaggio. 282

electronic voting: voto elettronico. 410

Elliptic Curve Cryptography: crittografia sulle curve ellittiche. 463

Elliptic Curve Method: metodo delle curve ellittiche. 482

Full Domain Hash: hash a dominio pieno. 211

fuzzy: sfocato. 272

greatest common divisor: massimo comun divisore. 444

group law: legge di gruppo. 466

hard-core: estremo. 57

hash table: tabella hash. 487

hash: polpettina fatta di avanzi di carne e verdure. 68

Hierarchical Identity-Based Encryption: cifrario su base identità di tipo gerarchico. 272

honest but curious: onesti ma curiosi. 403

Honest Verifier Zero Knowledge: conoscenza nulla con verificatore onesto. 355

Index Calculus: calcolo dell'indice. 487

Information Theory: teoria dell'informazione. 425

Initialization Vector: vettore di inizializzazione. 87

interleaving: interallacciamento. 283

Key Distribution Center: centro di distribuzione delle chiavi. 313

Key Encapsulation Mechanism: meccanismo di incapsulamento della chiave. 151

Key Generation Center: centro di generazione delle chiavi. 262

knapsack: sacco. 246

leakage: perdita. 9

Learning Parity with Noise: imparare la parità in presenza di rumore. 243

Learning With Errors: imparare in presenza di errori. 239

Linear Feedback Shift Register: Registro a scorrimento a retroazione lineare. 138

login: accesso. 293

malware: codice maligno. 35

man-in-the-middle: uomo-nel-mezzo. 283

meet-in-the-middle: incontro nel mezzo. 126

Message Authentication Code: codice autenticatore di messaggio. 184

min-entropy: entropia minima. 439

modification: modifica. 283

MultiParty Computation: computazione a parti multiple. 389

negligible: trascurabile. 12

Non-Interactive Zero Knowledge: conoscenza nulla non-interattiva. 371

Number Field Sieve: crivello dei campi di numeri. 485

Oblivious Transfer: trasferimento immemore. 395

One-Time Pad: blocco monouso. 33

One-Way Function: funzione unidirezionale. 54

One-Way Permutation: permutazione unidirezionale. 63

One-Way Trapdoor Function: funzione unidirezionale con botola. 148

Optimal Asymmetric Encryption Padding: cifrario asimmetrico ottimo con riempimento. 158

padding: riempimento. 86, 132

Password-based Chosen-basis CDH: problema CDH su base password a basi scelte. 338

perfect forward secrecy: sicurezza perfetta in avanti. 292

perfectly binding: pefettamente vincolante. 399

perfectly hiding: pefettamente celante. 399

Permutation boxes: scatole di permutazione. 122

Prime Number Theorem: teorema dei numeri primi. 475

Probabilistic Polynomial Time: probabilistico a tempo polinomiale. 12

Proof of Knowledge: prova di conoscenza. 362

prover: dimostratore. 294

PseudoRandom Function: funzione pseudocasuale. 110

Pseudorandom Generator: generatore pseudocasuale. 49

PseudoRandom Permutation: permutazione pseudocasuale. 116

Public Key Infrastructure: infrastruttura a chiave pubblica. 258

quadratic residue: residuo quadratico. 456

Quadratic Sieve: crivello quadratico. 485

Radio-Frequency IDentification: identificazione a radio-frequenza. 304

random variable: variabile aleatoria. 426

re-play: ri-uso. 283

receipt-freeness: assenza di ricevuta. 411

reflection: riflessione. 283

Registration Authority: autorità di registrazione. 264

round: turno. 119

salt and stretching: sale e stiramento. 329

Secure Hash Algorithm: algoritmo hash sicuro. 90

server: servente. 336

session hijacking: scambio della sessione. 282

Shortest Independent Vectors Problem: problema dei vettori più corti indipendenti. 238

Shortest Vector Problem: problema del vettore più corto. 238

Simulation Soundness: Validità di simulazione. 378

Smallest Integer Solution: soluzione agli interi più piccoli. 245

Special Honest Verifier Zero Knowledge: conoscenza nulla speciale con verificatore onesto. 362

Special Soundness: validità speciale. 362

Strong PseudoRandom Permutation: permutazione pseudocasuale forte. 117

Strong Special Honest Verifier Zero Knowledge: conoscenza nulla speciale in senso forte con verificatore onesto. 362

Subset Sum: somma di sottoinsiemi. 246

Substitution boxes: scatole di sostituzione. 122

threshold secret sharing: condivisione di segreti a soglia. 391

timestamp: francobollo temporale. 186

Trusted Third Party: terza parte fidata. 407

uncoercibility: incoercibilità. 411

Universal Composability: componibilità universale. 409

Weak PseudoRandom Function: funzione pseudocasuale debole. 138

Witness Indistinguishability: indistinguibilità di indizi. 356

Zero Knowledge: conoscenza nulla. 349

Indice analitico

UNITEXT – Collana di Informatica

Amministrazione avanzata di server Linux
Massimo Tartamella, Marco Sajeva, Benedetto Vassallo, Lorenzo Puccio
2004, X, 460 pp., ISBN 978-88-470-0234-0

Algoritmi. Lo spirito dell'informatica
David Harel, Yishai Feldman
2008, XXII, 616 pp., ISBN 978-88-470-0579-2

Qualità dei Dati. Concetti, Metodi e Tecniche
Carlo Batini, Monica Scannapieco
2008, XXII, 280 pp., ISBN 978-88-470-0733-8

Robotica mobile. Un'introduzione pratica
Ulrich Nehmzow; edizione italiana a cura di A. Chella, R. Sorbello
2008, XVI, 272 pp., ISBN 978-88-470-0385-9

Ricerca Operativa
Paolo Serafini
2009, XVI, 536 pp., ISBN 978-88-470-0845-8

Crittografia nel Paese delle Meraviglie
Daniele Venturi
2012, XIV, 500 pp., ISBN 978-88-470-2480-9