ASPETTI DI SICUREZZA IN SISTEMI AVANZATI DI DISTRIBUZIONE DI CHIAVI CRITTOGRAFICHE

Tesi di Laurea Specialistica in Informatica di

Alessio Merlo

RELATRICE DISI: Prof. Gabriella Dodero

RELATRICE ELSAG: Dott.ssa Anna Maria Colla

CORRELATRICE: Prof. Vittoria Gianuzzi

Con la collaborazione di:

UNIVERSITÀ DI GENOVA - DISI
DIPARTIMENTO DI INFORMATICA E SCIENZE DELL'INFORMAZIONE

ELSAG s.p.a.

ISICT
Istituto Superiore di Studi in Tecnologie dell'Informazione e della Comunicazione

SETTEMBRE 2005

Ai miei, per tutto.

Indice

Elenco delle figure

Ringraziamenti

I miei primi ringraziamenti vanno alla Prof. Gabriella Dodero, la mia relatrice del DISI, ed alla Dott.ssa Anna Maria Colla, la mia corrispondente presso Elsag s.p.a., per avermi seguito con molta costanza, pazienza (con me ce ne vuole molta, in verità!) e perenne disponibilità, permettendomi di lavorare in ogni momento sotto le migliori condizioni possibili e facendo in modo che mi appassionassi a questa tesi come a nessun altro lavoro prima d'ora.

Colgo inoltre l'occasione per ringraziare la Dott.ssa Anna Martinoli, l'Ing. Giuseppe Ghiorzi e l'Ing Emanuele Aonzo di Elsag s.p.a., per il grande supporto e la consulenza prontamente fornita nelle fasi "clou" di questo studio.

Voglio inoltre ringraziare l'Ing. Sebastiano B. Serpico, presidente dell'ISICT, ed il Prof. Maurizio Martelli per avermi incoraggiato ad intraprendere questa avventura presso Elsag s.p.a., alla quale va, inoltre, la mia più sentita riconoscenza per avermi garantito tutti i servizi necessari per operare con profitto.

Un pensiero affettuoso va anche al DISI, che in questi anni mi ha alacremente formato come informatico e come uomo.

Tutti i miei migliori pensieri e sentimenti vanno poi a quelle poche persone che, in tanti anni, mi sono sempre state vicine, nel bene e nel male, e che hanno apprezzato i miei pregi e sopportato di buon grado i miei difetti. Non ho l'intenzione né la necessità di

nominarli tutti, in quanto sono sicuro che coloro che si trovano in questa ristretta cerchia siano perfettamente coscienti di appartenervi.

Naturalmente, sono e sarò eternamente grato ai miei genitori, a cui questa tesi è dedicata, per l'amore, il supporto e la comprensione che ho sempre ricevuto: senza di loro questo lavoro non sarebbe mai esistito. Poiché tutte le parole del mondo non basterebbero per descrivere la mia graditudine, mi limiterò alla più semplice e significativa parola che conosco: grazie.

Genova, Alessio Merlo

7 settembre 2005.

Introduzione

La crittografia è un'arte che si è sviluppata lentamente nei secoli e si è migliorata continuamente, imparando dai propri errori e cercando di superare di volta in volta i propri limiti.

Il successo di questa arte è da ricercarsi nella potenza con cui permette di nascondere informazioni delle quali il mondo intero, o almeno la maggior parte di esso, non deve essere a conoscenza: applicare un processo crittografico ad un'informazione significa costruire attorno ad essa una cassaforte incorporea ma estremamente resistente, che può essere aperta solo da quella piccola parte di mondo che è abilitata a farlo e ne conosce la combinazione.

La crittografia è un universo a sé stante, con i buoni e con i cattivi, dove ognuno, come in ogni realtà che si rispetti, ha un ruolo specifico, un obiettivo, punti di forza e punti di debolezza.

Il mondo della crittografia, però, non è complesso da capire, tutt'altro: per rappesentarlo e descriverlo non occorre una metafora ricercata, una orazione di Cicerone, un romanzo di Joyce od un quadro di Picasso ma è sufficiente... una favola.

Infatti, nessun racconto ha una migliore analogia con la crittografia di "Alì Babà e i quaranta ladroni". In questa fiaba, il tesoro (*l'informazione*) viene mantenuto al sicuro dalla montagna (*sistema crittografico*) che può essere aperta, permettendo l'accesso al tesoro, solo con la frase "Apriti, Sesamo!" (*il segreto*) di cui sono a conoscenza solo i quaranta ladroni (*gli utenti legali*). Tuttavia, il tesoro è al sicuro solo se il segreto rimane nelle mani dei ladroni: nel momento in cui Alì Babà (l'intruso) viene a conoscenza, in qualche modo,

del segreto, può facilmente aprire la montagna ed impossessarsi del tesoro ad insaputa dei ladroni. Mirabile dictu, in questa situazione i ladroni non sono i cattivi, ma sono i buoni che cercano di proteggere il proprio tesoro, mentre Alì Babà è il "*malvagio*" che vuole aprire la montagna senza esserne autorizzato, e per farlo si impossessa del segreto.

Sebbene questa sia una visione decisamente semplicistica, tuttavia fornisce un'immagine chiara del fulcro su cui si basa l'intero mondo crittografico, cioè la determinazione e la protezione di un segreto, dalla cui riservatezza dipende la sopravvivenza di tutto un sistema di crittografia.

La crittografia, nella sua evoluzione, ha portato a metodi sempre più sicuri per rendere i segreti difficili da individuare a chi non ne sia a conoscenza, e per fare in modo che il loro utilizzo non ne comporti la dispersione ad estranei. Nonostante i passi da gigante fatti in questo senso, benché oggi si possa mettere gli utenti in condizione di determinare ed usare un segreto in maniera abbastanza sicura, un problema molto aperto e dibattuto, la cui importanza è aumentata negli anni in modo direttamente proporzionale allo sviluppo di Internet, è il problema della distribuzione della chiave (segreto) tra gli utenti autorizzati ad accedere all'informazione protetta dal sistema crittografico.

L'aumento delle dimensioni della rete pubblica, infatti, ha reso sempre maggiori le distanze tra coloro che condividono il segreto: i quaranta ladroni, oggi, non sarebbero più seduti sotto una palma a decidere assieme la "chiave" per far aprire la montagna, ma sarebbero ognuno in una città diversa del mondo ad aspettare che il loro capo la decida e la comunichi agli altri, tramite la rete, in modo tale che nessun altro, tranne loro, ne possa a venire a conoscenza.

Questo problema è di importanza fondamentale, in quanto una distribuzione non sicura di una chiave viola la sicurezza di tutto il sistema crittografico, poiché se il segreto viene carpito da terzi, la sicurezza di tutto il sistema decade.

Tutti i sistemi di distribuzione delle chiavi, che si sono studiati nel tempo, non riuscivano

a garantire, nemmeno a livello teorico, una sicurezza incondizionata, ma prevedevano sempre un piccolo margine di insicurezza, una piccola probabilità che comunque il segreto potesse cadere in mani sbagliate, poiché non vi era la capacità di determinare o meno l'eventuale accesso alla chiave da parte di terzi.

Solo recentemente, la nascita della crittografia quantistica ha permesso di realizzare, almeno in linea teorica, sistemi di distribuzione di chiave tra utenti distanti in maniera sicura, ovvero tale che ogni tentativo di carpire la chiave scambiata possa venire prontamente scoperto.

Lo scopo di questa tesi è, in primis, di valutare i limiti della crittografia classica e di esaminare la crittografia quantistica come possibile risposta a tali limiti per la realizzazione di sistemi crittografici e di distribuzione che abbiano una sicurezza incondizionata, cioè totale.

Tuttavia, il problema fondamentale di tali sistemi quantistici risiede nel fatto che, sebbene promettano un livello di sicurezza della distribuzione della chiave sicuro al 100% in linea teorica, nel momento in cui vengono implementati danno origine ad una serie di problematiche di cui occorre tenere presente e che vanno risolte opportunamente, per fare in modo che il livello di sicurezza e di affidabilità del sistema implementato non risulti inferiore a quello ipotizzato nella teoria.

Più specificamente, si è partito dal valutare l'implementazione di un sistema di crittografia quantistica di ultima generazione (QCRYPT, implementato presso Elsag s.p.a.) per determinare quali aspetti di sicurezza debbano essere garantiti su un sistema destinato a "farlo girare".

Le problematiche più significative che si sono riscontrate riguardano il problema di garantire la sicurezza incondizionata dagli attacchi da rete di uno storage di dati sensibili ed il problema dell'identificazione dell'utente.

Il primo problema si è manifestato in fase di testing del sistema quantistico ed ha messo

in luce il problema pratico di dover salvare le chiavi crittografiche in possesso delle parti autorizzate, dopo la distribuzione, su sistemi che fossero sicuri incondizionatamente, tanto quanto il sistema di distribuzione stesso.

Il secondo problema è decisamente di più ampio respiro e benché una corretta identificazione fosse anche un'ipotesi fondamentale per il corretto funzionamento del sistema quantistico analizzato, questo è un problema comune e mai totalmente risolto di qualunque sistema crittografico, dalla distribuzione delle chiavi all'accesso ad un sistema protetto tramite segreto. Pertanto, si è scelto di trattarlo come un argomento a sé stante e nello stesso tempo di proporre soluzioni ottimali anche per il sistema QCRYPT, dal cui studio questa tesi ha avuto origine.

Per entrambi questi problemi, comunque, si è cercato di dare risposte che esulassero dal contesto specifico del sistema di partenza, cercando di determinare architetture, algoritmi e protocolli che fossero soluzioni il più possibile generali, destinati quindi ad ambiti anche diversi da quello crittografico o di distribuzione delle chiavi, ma che anzi potessero essere adottati nei più svariati contesti di sicurezza.

Il fine ultimo di queste pagine è pertanto di riuscire a proporre sistemi che possano essere usati da soli, o insieme ad altri metodi, per fare in modo che la nostra favola da "Mille e una notte", oggi, possa essere riscritta diversamente, con buona pace di tutti gli Alì Babà del nostro tempo.

Capitolo 1

Dalla crittografia classica al QCRYPT

1.1 I limiti della crittografia classica

La crittografia classica nasce dalla necessità di permettere a due parti (dette convenzionalmente Alice e Bob) di effettuare comunicazioni segrete (ovvero tali che i non autorizzati non possano ottenerle) su canali cosiddetti *insicuri*, cioè pubblici, aperti a tutti e pertanto possibili sedi di attaccanti (detti invece Mallory o Eve) che vogliano carpire le informazioni scambiate. L'idea base risiede nella condivisione di un segreto noto solo alle parti comunicanti che viene utilizzato come *chiave*, ovvero come mezzo per alterare i dati, in modo tale che solo chi conosce tale segreto sia nelle condizioni di poter accedere al messaggio.

1.1.1 La crittografia a chiave simmetrica

La prima forma di crittografia, che storicamente ha origini molto antiche (Codice di Cesare, 60 A.C.) si basa sull'idea di avere un'unica chiave da utilizzare sia per cifrare il messaggio in chiaro (sia C) da parte del mittente (di solito noto come Alice) che per decifrare il messaggio cifrato (sia E) da parte del destinatario (di solito, Bob). Per realizzare un sistema di questo tipo occorre utilizzare funzioni che siano *reversibili*. In fig. a pag. 6 si può vedere il meccanismo di cifratura: in pratica il mittente cifra con la chiave il messaggio in chiaro e

lo invia sul canale insicuro all'altra parte; il ricevente applica sul messaggio cifrato la stessa chiave usata dal mittente e ottiene nuovamente il messaggio in chiaro.

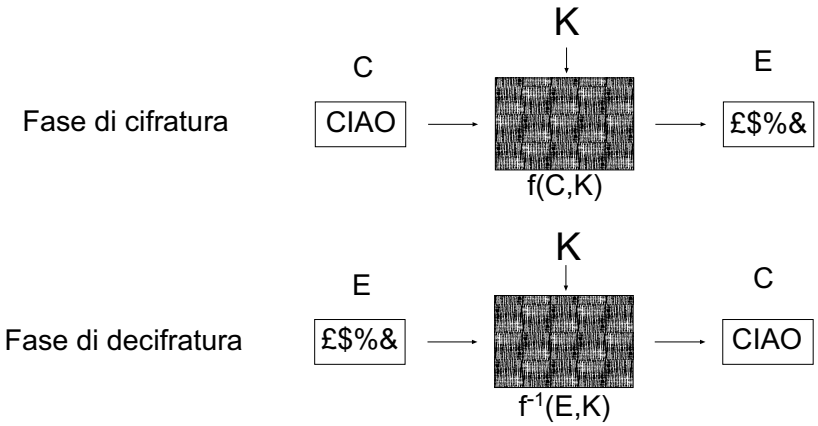


Figura 1.1: Cifratura e decifratura a chiave simmetrica

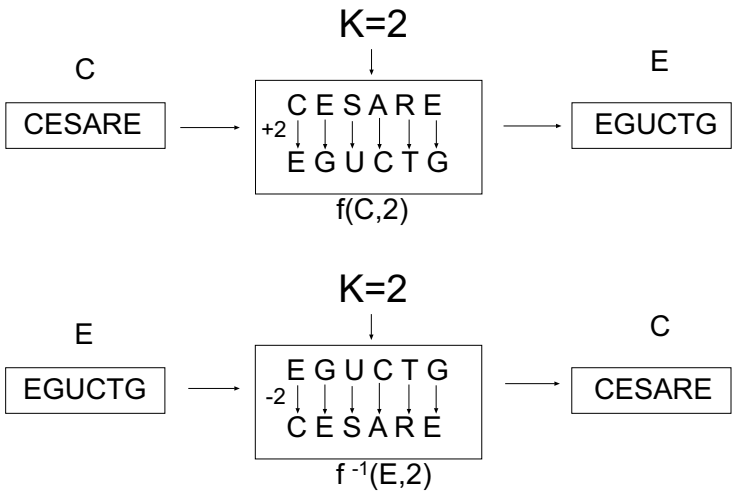


Figura 1.2: Il Codice di Cesare

A titolo di esempio, il Codice di Cesare (fig. a pag. 6) citato in precedenza prevedeva la cifratura di un messaggio attraverso la sostituzione di ogni lettera del messaggio con la lettera distante k posizioni a destra nell'alfabeto usato. A questo punto per ottenere nuovamente il messaggio in chiaro bastava, noto lo spostamento k (che è la "chiave"), sostituire ogni lettera del messaggio cifrato con la lettera che si trovava k posizioni a sinistra

nello stesso alfabeto [1].

È chiaro però che tale algoritmo non è adatto per un'applicazione crittografica reale in quanto prevede un numero molto limitato di chiavi possibili (i.e. tante quante le lettere dell'alfabeto meno uno) ed è quindi facilmente attaccabile con un approccio *a forza bruta*, semplicemente provando tutte le chiavi possibili in breve tempo.

Gli algoritmi di cifratura a chiave simmetrica sono divisi in gruppi e catalogati in base al modo di applicare la cifratura sul testo in chiaro.

Si distinguono pertanto due famiglie principali di algoritmi:

- **stream cipher** dove l'algoritmo di cifratura viene applicato ad un bit o ad un byte alla volta.
- **block cipher** se l'algoritmo viene applicato a gruppi di bit o byte (*blocchi*) di dimensione fissa.

I cifrari a blocchi sono di gran lunga i più diffusi e si dividono a loro volta in sottogruppi che dipendono dal tipo di operazioni che vengono eseguiti sui blocchi per cifrarli.

- **ECB - Electronic CodeBook** è la categoria che raccoglie la cifratura a blocchi più semplice ma anche meno affidabile: ogni blocco viene cifrato sempre con la stessa chiave in successione. A questa categoria appartengono lo stragrande maggioranza degli algoritmi di cifratura utilizzati ad oggi. Sono poco affidabili poiché a blocchi in chiaro uguali corrispondono blocchi cifrati uguali, con il rischio di poter essere analizzati per ricavare informazioni sulla chiave
- **CBC - Cipher Block Chaining** i metodi di cifratura che appartengono a questa categoria collegano i blocchi cifrati ai precedenti nel seguente modo: il blocco cifrato

[1] Un esempio famoso e recente dell'applicazione di tale algoritmo si trova nel film "2001, Odissea nello spazio" di Kubrik dove il supercomputer 'HAL' non è altro che il nome 'IBM' cifrato con K=25 (l'alfabeto inglese è di 26 lettere)

corrente viene ottenuto dalla cifratura del blocco di testo in chiaro considerato in XOR con il blocco cifrato precedente

- **CFB - Cipher FeedBack** anche gli algoritmi appartenenti a questa categoria collegano i blocchi con i precedenti ma in questo caso il blocco cifrato corrente viene ottenuto dallo XOR di parte del blocco in chiaro considerato con il blocco cifrato precedente (l'idea è quella di elaborare i dati non appena sono disponibili anziché aspettare che sia completata l'elaborazione di un intero blocco)

- **OFB - Output FeedBack** a questa categoria appartengono le codifiche a blocchi più veloci: i blocchi sono sempre collegati con i precedenti, ma il collegamento avviene tra l'output del passo precedente ed il blocco considerato. Viene utilizzato nelle comunicazioni ad alta velocità (come quelle dei satelliti)

La debolezza maggiore dei cifrari a chiave simmetrica risiede paradossalmente proprio nel loro punto di forza: la sicurezza offerta risiede nel fatto che solo chi possiede la chiave possa accedere alle informazioni; d'altro canto però occorre un modo efficiente e soprattutto sicuro per fare in modo che le due parti (in genere considerevolmente distanti tra loro) riescano a condividere la chiave: tale processo non è banale e comunque non si riesce a garantire al 100% che nessun intruso possa averla carpita in qualche modo.

1.1.2 La crittografia a chiave asimmetrica

La cifratura a chiave asimmetrica utilizza invece due chiavi diverse: una per la cifratura dell'informazione e l'altra per la decifratura.
Le informazioni cifrate con una delle due chiavi possono essere decifrate solo con l'altra. Le due chiavi prendono generalmente il nome di *chiave pubblica* (public key) e *chiave privata* (secret key) dal momento che una delle due chiavi (quella pubblica) viene liberamente

diffusa, al fine di poter essere usata per cifrare messaggi destinati al proprietario della stessa, mentre l'altra (quella privata) deve essere tenuta segreta dallo stesso proprietario della coppia di chiavi. Si noti che è totalmente indifferente quale delle due chiavi si scelga come pubblica e quale come privata; l'unica cosa di cui occorre tenere presente è che una volta scelte la chiave pubblica e quella privata, queste non sono intercambiabili successivamente. Tale metodo, per contrapposizione al precedente è anche detto "*a chiave pubblica*".

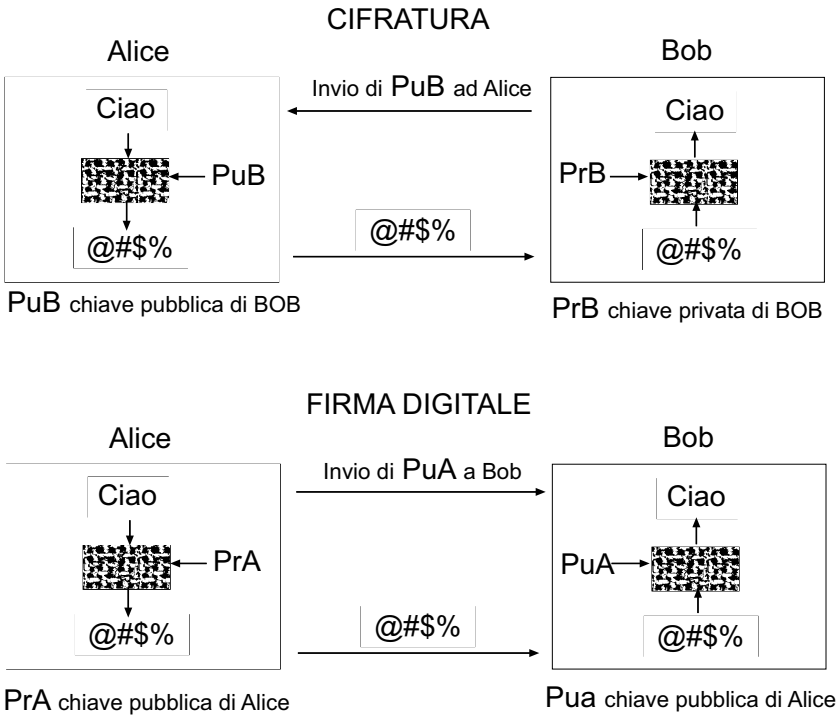


Figura 1.3: Cifratura e firma digitale con chiave asimmetrica

Per la **cifratura** di un messaggio, questo viene cifrato dal mittente per mezzo della chiave pubblica del destinatario ed inviato al destinatario stesso, il quale è in grado di decifrarlo, poiché è a conoscenza della sua chiave privata.

In questo caso lo scambio della chiave pubblica può avvenire anche attraverso un canale insicuro, poiché chi reperisce la chiave pubblica relativa ad un'entità può soltanto inviarle

informazioni cifrate, senza poter decifrare, in quanto non dispone della chiave privata.
La crittografia a chiave pubblica permette inoltre un altro uso dell'algoritmo di cifratura che non ha finalità di segretezza ma di *autenticazione* della provenienza del messaggio: la **firma digitale**.
La firma digitale si realizza cifrando un messaggio con la chiave privata anzichè quella pubblica. I dati cifrati in questo modo possono essere decifrati da tutti (con la chiave pubblica corrispondente), pertanto non si garantisce protezione dei dati, ma si attesta che i dati in questione provengono effettivamente da una certa fonte, l'unica che può aver cifrato il messaggio con quella chiave privata.
La sicurezza del meccanismo di cifratura deriva dalla difficoltà di derivare una delle due chiavi conoscendo l'altra. Gli algoritmi di cifratura si basano sulla difficoltà di effettuare in tempi brevi la fattorizzazione di numeri primi molto grandi. In sostanza, considerando numeri primi molto grandi e facendone il prodotto, trovare i due numeri primi nei quali è scomponibile il risultato ha un'elevata complessità computazionale.

Un esempio di cifratura a chiave pubblica - RSA

RSA è stato di gran lunga uno dei più famosi ed usati algoritmi a chiave pubblica. Questo algoritmo, creato nel 1977 da R. Rivest, A. Shamir e L. Adelman (tre docenti del MIT), prende proprio il suo nome dalle iniziali dei suoi inventori.
Nell'RSA la generazione delle chiavi viene effettuata attraverso i seguenti passi:

1. Si scelgono due numeri primi p e q abbastanza grandi e se ne calcola il prodotto $n = p \ \ q$, detto modulo
2. Si sceglie un numero d tale che $d > 1$ e che non abbia fattori a comune con $(p-1)(q-1)$, cioè d e $(p-1)(q-1)$ siano primi tra loro
3. Calcolare e per il quale risulti $ed \ mod \ (p-1)(q-1) = 1$, ovvero $(ed-1)$ sia divisibile per $(p-1)(q-1)$

La chiave pubblica è rappresentata da (n, e) e quella privata da (n, d).

La cifratura dei messaggi avviene seguendo i passi qui riportati:

1. Codificare il messaggio M, in maniera nota e reversibile, in un numero $m : m < n$
2. Calcolare $c = me \ mod \ n$, dove (n, e) è la chiave pubblica del destinatario

Il messaggio cifrato è c.

La decifratura di un messaggio c avviene invece calcolando $m = cd \ mod \ n = med \ mod \ n = m \ mod \ n = m \ (m < n)$, dove (n, d) è la chiave privata del destinatario di c.
La lunghezza della chiave si riferisce generalmente al numero di bit necessari alla rappresentazione del valore n (il modulo). I due numeri primi p e q il cui prodotto è n, dovrebbero essere dello stesso ordine di grandezza, ma non troppo "vicini" tra loro (la loro differenza non deve essere un numero "piccolo"), in modo da rendere più difficile la fattorizzazione di n. La lunghezza della chiave dipende dalla sicurezza che si desidera ottenere con questo meccanismo: quanto sono importanti le informazioni trasmesse e per quanto tempo devono rimanere "segrete" agli altri. Più lunga è la chiave, maggiore è la sicurezza del sistema, poiché è maggiore il tempo che occorre per decifrare i messaggi con un approccio a forza bruta.
Il limite di RSA, come di tutti gli altri algoritmi a chiave asimmetrica (e.g. El Gamal e DSA) risiede nel fatto che la fattorizzazione dei numeri è destinata a diventare "computazionalmente" sempre meno complessa, con la conseguente diminuzione del livello di sicurezza offerto dal sistema. Negli anni, infatti, i tentativi di violare RSA in tempi brevi sono stati molti.
Ad esempio, un tentativo per la rottura del sistema RSA è stato effettuato nel 1999 con una lunghezza della chiave di 512 bit. La chiave è stata fattorizzata con successo in 7 mesi.

Tutto ciò ha portato a non poter ritenere più sicure le chiavi di 512 bit da quel momento in poi.

1.1.3 Vulnerabilità dei sistemi crittografici classici

Quanto visto finora permette di arrivare alla conclusione che la sicurezza di un sistema crittografico simmetrico o asimmetrico sia inversamente proporzionale al passare del tempo, in quanto lo sviluppo congiunto di algoritmi e nuove possibilità di calcolo computazionale riducono di fatto il livello di affidabilità ed il periodo di validità di una chiave sia pubblica che privata. Inoltre, la sicurezza del sistema è inversamente proporzionale anche alla quantità di dati che vengono cifrati con una certa chiave: maggiore è la quantità di dati scambiati, più alta la probabilità che un eventuale attaccante, confrontando i diversi messaggi cifrati che passano sul canale, possa riconoscere patterns comuni nei messaggi (i.e.con analisi statistiche di ricorrenza delle lettere più usate) ed arrivare a ricostruire la chiave condivisa.

Il periodo di validità della chiave

Attraverso la tecnica di attacco vista (detta *crittoanalisi*) un attaccante può rendere totalmente insicuro il sistema: per questo motivo, risulta chiaro quanto la validità di un sistema crittografico sia dipendente dal tempo. Più una chiave è usata maggiori sono le probabilità che venga decifrata e minore dunque la sicurezza globale del sistema. Inoltre se un attaccante non facesse della crittoanalisi sui messaggi ma cercasse anche solo di provare tutte le chiavi possibili, prima o poi riuscirebbe ad ottenere la chiave.
Questa semplice equazione permette di giungere alla conclusione che una chiave è di fatto sicura solo per un lasso di tempo minore del tempo che un potenziale attaccante *onnipotente* (nello studio di problemi di sicurezza si ragiona sempre nel caso peggiore, ovvero supponendo di avere a che fare con entità maliziose con un potere computazionale illimitato) impiegherebbe per risalire alla stessa. Questo porterebbe ad una gestione dei segreti condivisi molto dinamica e difficile poiché i continui cambi richiederebbero uno sforzo notevole

per le due parti comunicanti, che dovrebbero avere la capacità di cambiare (e non divulgare!) una nuova chiave segreta ogni volta, per poi utilizzarla solo per brevi periodi prima di doverla nuovamente sostituire. Inoltre il periodo di validità di una chiave è direttamente proporzionale al livello di sicurezza richiesto dalle due parti: più le informazioni scambiate devono rimanere segrete, prima è consigliabile cambiare la chiave (anche prima del tempo medio stimato per scoprirla); d'altro canto, se le informazioni hanno una richiesta di segretezza più bassa, si può mantenere la chiave anche per periodi più lunghi (oltre il limite temporale di sicurezza). Occorre inoltre tenere presente, in quest'ottica, che la durata del periodo di sicurezza di una chiave si restringe col passare degli anni, poiché la potenza di calcolo degli elaboratori aumenta notevolmente (secondo la legge di Moore, almeno fino ad oggi).

Infine bisogna considerare un ultimo aspetto importante: molti algoritmi di cifratura classici a chiave asimmetrica (e.g. RSA) hanno il loro punto di forza nella difficoltà di fattorizzare il prodotto di numeri primi. Da un punto di vista prettamente teorico, è vero che il problema della fattorizzazione è NP-completo, e quindi difficile da risolvere, ma sono in via di sviluppo algoritmi sempre migliori per la fattorizzazione e, addirittura, ad oggi non ci sono dimostrazioni che escludano l'esistenza di un algoritmo polinomiale che ancora non è stato scoperto!

Inoltre bisogna tenere presente che esiste già un algoritmo "quantistico", *l'algoritmo di Shor*, che risolve la fattorizzazione in tempo polinomiale. Ciò significa che nel momento in cui il calcolatore quantistico riuscisse a diventare realtà, la sicurezza di tutti questi sistemi crittografici, che sono diventati standard "*de facto*", cadrebbe facilmente e potrebbero non essere più utilizzabili.

L'idea stessa di cambiare la password ogni volta dopo un certo periodo di tempo, benché al momento efficace, è comunque una soluzione estemporanea che non riesce a superare il limite intrinseco della validità limitata di tutte le chiavi e password che sono utilizzate negli

algoritmi crittografici attuali.

L'attacco passivo

Come detto, il grande limite della crittografia a chiave simmetrica è nella necessità di cambiare spesso la chiave, che impone ad Alice e Bob di determinarla ed in qualche modo comunicarsela, in modo tale che il nuovo segreto sia condiviso da ambo le parti. Questa fase può essere davvero critica in quanto costringe le due parti a comunicare un segreto, o parte di esso, sul canale insicuro su cui un attaccante può realizzare un'attività di *sniffing* (attacco passivo in cui l'attaccante si limita ad ascoltare il traffico sulla rete tra le due parti) attraverso la quale può carpire tutta o parte della chiave. In realtà esistono metodi che permettono di fronteggiare questo problema, ma non sono totalmente sicuri e complicano la complessità dell'algoritmo di cifratura.

Il problema maggiore dell'attacco passivo risiede comunque nel fatto che un intervento di un attacante è destinato a passare inosservato, mentre né Alice né Bob riescono ad accorgersi se si sta verificando una situazione di questo tipo: non si può sapere se le informazioni scambiate sono lette da qualcun altro, poiché la lettura passiva non porta alcuna perturbazione sulle informazioni che viaggiano sul canale.

1.1.4 One Time Pad

Nonostante quanto detto sugli algoritmi crittografici odierni sia vero per la quasi totalità dei cifrari, esiste tuttavia un unico algoritmo crittografico a chiave *simmetrica* la cui sicurezza è totalmente *incondizionata*[2]:l'**One Time Pad**.

Questo algoritmo, proposto originariamente da Gilbert Vernam, prevede di utilizzare una nuova sequenza random per ogni messaggio, lunga quanto il messaggio stesso. Il sistema

[2]Con il termine "incondizionata" in questo caso si sottolinea il fatto che le chiavi utilizzate abbiano un periodo di validità potenzialmente infinito nel tempo. Tale proprietà è stata dimostrata matematicamente da Claude Shannon.

di cifratura è semplice e prevede di eseguire uno *xor* bit a bit tra i bit del messaggio e quelli della chiave.

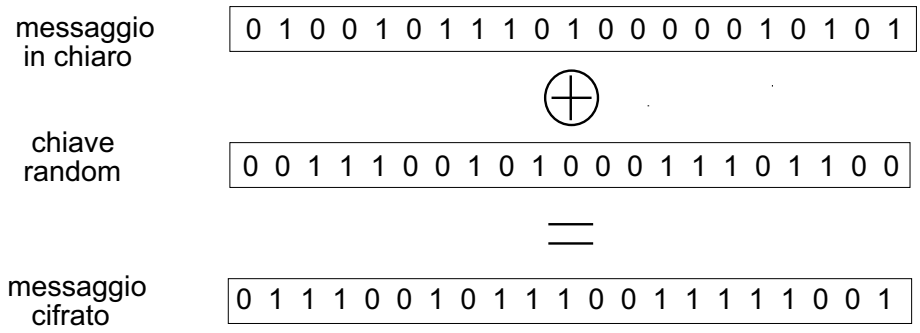


Figura 1.4: La cifratura tramite One Time Pad

Applicabilità dell'One Time Pad in un contesto reale

L'applicazione di un simile algoritmo in un contesto crittografico reale presuppone di disporre di un'immensa quantità di chiavi, che devono essere random e soprattutto devono essere contemporaneamente condivise dalle parti autorizzate a comunicare. Per quanto concerne il primo aspetto, la generazione delle chiavi necessarie può essere realizzata tramite un generatore pseudorandom che, a partire da un seme, genera una serie di sequenze pseudocasuali che possono essere utilizzate con l'One Time Pad.

Il problema di fondo è che i generatori pseudorandom tendono a ricreare sempre le stesse sequenze di numeri a partire dallo stesso valore di seme (le combinazioni non sono tutte equiprobabili): questo costituisce di per sé un punto di debolezza che può essere sfruttato da un attaccante e che potrebbe ridurre la sicurezza del sistema.

Alternativamente ai generatori pseudorandom, esistono generatori di chiavi che vengono definiti a tutti gli effetti (ed erroneamente) "random" ma sono decisamente più costosi e devono essere sottoposti ad un certo numero di test prima di poter essere usati, al fine

di determinare se la natura delle sequenze da essi generati sia effettivamente casuale[3] e lo permanga nel tempo. Queste dimostrazioni però non risultano esaustive, in quanto garantiscono che una certo sottoinsieme delle chiavi (benché questo possa essere molto grande) sia random ma non possono assicurare che questo valga comunque per tutte le sequenze generate: l'utilizzo di una componente elettronica per la generazione di sequenze casuali non permette di assumere che questa sia randomica sempre e al 100%. In pratica, quindi, anche i generatori "random" migliori presenti sul mercato non possono garantire una randomicità totale: questo limite non permette quindi un'applicabilità dell'One Time Pad che garantisca una sicurezza incondizionata ma farebbe permanere nell'utilizzo dell'algoritmo una percentuale di "insicurezza" dovuta alla natura delle chiavi usate[4].

Oltre al problema della generazione, un altro aspetto da tenere in seria considerazione è l'invio delle chiavi all'altra parte, in modo tale che questa possa applicare sul messaggio cifrato il processo di decifratura (identico xor della fase di cifratura, ma tra il messaggio crittato e la chiave). Il problema è che ogni informazione che viaggia sul canale per raggiungere l'altra parte può essere, allo stesso modo ed in maniera totalmente impercettibile, catturata da uno sniffer (sia Eve). È chiaro quindi come diventi problematico condividere il segreto tra due entità distanti in maniera sicura, tanto più in questo caso specifico in cui la dimensione del segreto condiviso è così grande (tanto quanto il messaggio da cifrare!).

In realtà si potrebbe utilizzare una comunicazione sicura con tanto di password precondivisa e qualche metodo crittografico a chiave privata per cifrare le chiavi random, ma si ricadrebbe nel caso in cui non si può garantire comunque una sicurezza totale ed incondizionata, a causa dei limiti già visti in precedenza per questo tipo di algoritmi.
I due problemi evidenziati rendono quindi di fatto non giustificato e non applicabile l'uso

[3]Tra i test più interessanti vi è la compressione della chiave: si fornisce in input la chiave ad un compressore di file (i.e. gzip) e se la compressione è nulla o quasi la chiave è random poiché il compressore non ha trovato patterns ricorrenti.

[4]La dimostrazione di Shannon garantisce la totale sicurezza dell'One Time Pad solo sotto l'ipotesi di chiavi totalmente random, cosa che con i metodi di generazione tradizionali non può essere sempre garantita.

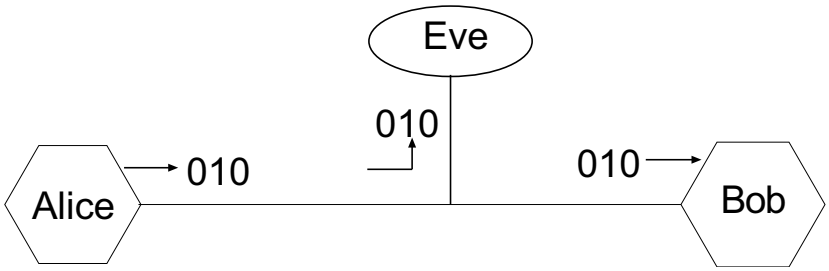


Figura 1.5: Schema di un attacco passivo

dell'One Time Pad: l'utilizzo sarebbe condizionato da una sicurezza non totale, pertanto si ottiene lo stesso risultato adoperando un algoritmo simmetrico meno oneroso in termini di lunghezza della chiave e di conseguenti dati segreti da condividere.

1.2 La crittografia quantistica

1.2.1 I principi della crittografia quantistica

La sicurezza condizionata, non totale o l'inapplicabilità dell'unico strumento di cifratura incondizionatamente sicuro rendono di fatto la crittografia classica deficitaria per l'applicazione in contesti dove un livello di sicurezza molto alto (100%)[5] sia costantemente richiesto.

Nonostante la sostanziale inadeguatezza della crittografia classica per simili scenari, la soluzione oggi esiste e prende il nome di **Crittografia Quantistica (Quantum Cryptography)**.

La crittografia quantistica ha come base teorica i principi della meccanica quantistica e, più in particolare, sfrutta le caratteristiche fisiche di cui godono particelle quali i fotoni per realizzare un sistema di cifratura sicuro a tutti gli effetti.

Le intuizioni della meccanica quantistica, che diventano il punto cardine su cui la crittografia omonima si basa, sono:

[5] È chiaro che una sicurezza di questo tipo si rende necessaria in realtà particolari come ambiti militari e governativi, dove la dispersione anche di parte dell'informazione può portare a gravi conseguenze.

- Non è possibile determinare, simultaneamente e con una precisione arbitrariamente alta, sia la posizione che il momento di una particella.
- Una particella prima di essere misurata non è in uno stato particolare ma nella sovrapposizione di tutti i suoi stati possibili.
- Una misurazione sulla particella la perturba irreversibilmente, facendola decadere in uno dei suoi stati base.
- Non si può duplicare uno stato quantistico sconosciuto (no cloning theorem), cioè non è possibile copiare lo stato di una particella senza prima conoscerlo mediante misurazione (e conseguente perturbazione).

Tali intuizioni si rivelano essere decisamente utili per la causa crittografica perché, utilizzando particelle (o qbit) che viaggiano su un canale quantistico tra Alice e Bob anzichè bit su un canale classico, si può capire se un'Eve stia cercando di carpire informazioni, grazie alla particolare "sensibilità" dei qbit verso ogni forma di misurazione.
Il punto di partenza della crittografia quantistica è l'utilizzo di un canale quantistico anziché classico per permettere ad Alice e Bob di generare chiavi crittografiche random per One Time Pad. Il canale quantistico è di per sé una fibra ottica su cui viaggiano fotoni che rappresentano i bit della chiave random, ma sui quali ogni forma di *eavesdropping* ("ascolto") genera una perturbazione irreversibile sul fotone che può venire rilevata dal ricevente. La caratteristica del segnale quantistico (sotto forma di fotone) che viaggia sul canale è di non essere *replicabile* ovvero non è possibile **copiare** lo stato di un fotone. Un attaccante intenzionato ad ascoltare i dati in viaggio da Alice verso Bob dovrebbe riuscire a copiare il fotone per fare in modo che Bob non si accorga del tentativo di sniffing. Poiché la fisica quantistica garantisce che il fotone non sia copiabile, Eve dovrebbe intervenire attivamente intercettando il qbit di Alice, distruggerlo e sostituirlo con uno identico.

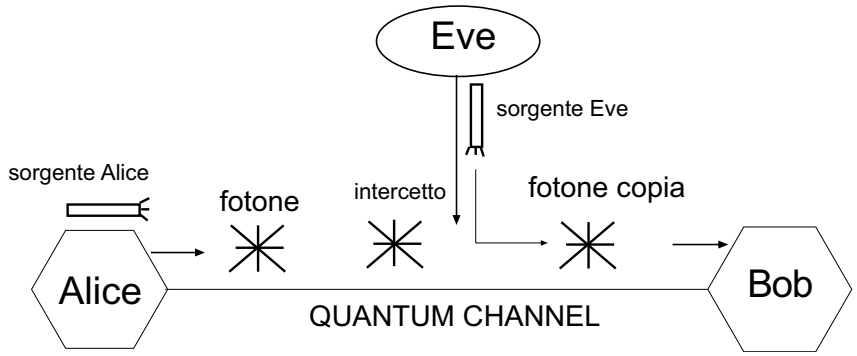


Figura 1.6: Intervento di Eve sul canale quantistico

Oltre al fatto che un tentativo di attacco di questo tipo non è semplice (si vedrà meglio nei paragrafi successivi), il fatto stesso che Eve debba comunque intervenire attivamente per nascondere un'intrusione (cosa che nel caso classico non era necessario) è una situazione che può essere sfruttata dal sistema crittografico per rilevare un attacco.
La capacità di sventare attacchi passivi come lo sniffing semplifica molto il problema della sicurezza della condivisione di un segreto tra due entità distanti, poiché queste possono verificare eventuali differenze sui dati ricevuti dopo lo scambio e decidere di scartarlo nel caso ci sia il sospetto di un intercettamento.
Oltre a questo, un'altra caratteristica garantita dai principi della meccanica quantistica è la capacità di realizzare sequenze randomiche a tutti gli effetti ed in grande quantità.

I principi quantistici quindi permettono di risolvere le due problematiche che hanno reso finora inapplicabile l'One Time Pad e di fatto permettono di rilanciarlo e utilizzarlo a tutti gli effetti come base per realizzare un sistema incondizionatamente sicuro.
Si noti però che l'uso dell'One Time Pad è sì necessario, ma non sufficiente a garantire che il sistema sia totalmente sicuro: affinché questo possa essere assicurato, occorre che ogni parte del sistema possa essere definita sicura, poiché la sicurezza di un sistema complesso è data dalla sicurezza della sua parte più "debole". Pertanto, la sicurezza incondizionata non

è garantita solo dall'uso dell'One Time Pad ma dall'unione tra l'utilizzo di questo algoritmo critttografico e lo sfruttamento delle possibilità offerte dai principi fisici visti.
Nel seguito, si vedrà in primo luogo quali aspetti di sicurezza occorrerà gestire nella fase di generazione delle chiavi per l'One Time Pad, analizzando il primo protocollo crittografico per la crittografia quantistica (il BB84) ed una sua implementazione (il QCRYPT). Successivamente si analizzeranno le caratteristiche che un sistema destinato ad utilizzare tale implementazione deve avere, determinando una serie di requisiti specifici, quantificandoli in termini di importanza e necessità. Infine verranno proposte soluzioni che permettano di rispettare i vincoli di sicurezza evidenziati in precedenza sul suddetto sistema.

1.2.2 Il problema della condivisione della chiave

Da quanto si è visto finora, la garanzia di un sistema crittografico incondizionatamente sicuro deve avere come punto di partenza l'One Time Pad ovvero l'unico algoritmo crittografico intrinsecamente sicuro. Per poter utilizzare l'One Time Pad occorre riuscire a far condividere in maniera sicura le chiavi senza che queste siano note ad estranei. Lo scenario nel quale ci si pone è di avere due parti comunicanti che, in qualche modo, devono generare e condividere delle chiavi random. Poiché è chiaro come in un contesto classico non sia possibile garantire al 100% che i dati sul canale non siano intercettati, occorre sfruttare i principi della meccanica quantistica unitamente ad un canale quantistico come mezzo per condividere la chiave tra le due parti.
Inoltre, si è detto che il principio base della meccanica quantistica da cui si parte è la *non replicabilità dello stato quantistico (no cloning theorem)* cioè il fatto che non sia possibile copiare lo stato quantistico, ad esempio di un fotone in transito su un canale, ma solo misurarlo perturbandolo irreversibilmente. Questa considerazione giustifica l'uso di un canale quantistico per lo scambio delle chiavi per l'One Time Pad dove lo stato di ogni singolo

fotone sottintenda lo stato di un bit della chiave. Tuttavia, non basta solo l'utilizzo di un canale quantistico per poter garantire una sicurezza incondizionata. Come si è evidenziato in precedenza, un attaccante potrebbe intercettare il fotone in transito da Alice, leggerne lo stato e immettere sul canale verso Bob un fotone identico con lo stesso stato, creandolo attraverso una sua sorgente di fotoni. Benché il no cloning sia importante, occorre quindi utilizzare opportuni accorgimenti ed adottare algoritmi che garantiscano che la presenza di un attaccante venga rilevata. Prima di determinare la natura di un algoritmo che sfrutti opportunamente questa proprietà, è necessario dettagliare più precisamente in che modo sia possibile far "trasportare" ad un fotone delle *informazioni*.

1.2.3 La polarizzazione dei fotoni

La meccanica quantistica considera la luce descritta in termini di fotoni cioè come un insieme discreto di particelle indivisibili di massa nulla (quanti di energia), che si propagano a velocità c). L'intensità di un fascio luminoso è direttamente proporzionale al numero di fotoni nel fascio. La luce inoltre ha una doppia natura, ondulatoria e particellare: questa doppia natura appartiene quindi al singolo fotone. Il fotone può, in quanto onda elettromagnetica, possiede una proprietà detta **polarizzazione**. La polarizzazione di un'onda elettromagnetica monocromatica è il piano di oscillazione del campo elettrico dell'onda stessa.

I tipi di polarizzazione

Un fascio di luce è quindi un insieme di fotoni dove in genere ognuno di essi ha una propria polarizzazione[6]. Ai fini della crittografia quantistica, è necessario riuscire ad ottenere, da un fascio di luce non polarizzato, dei fotoni che abbiano un certo tipo di polarizzazione. La polarizzazione che viene utilizzata in ambito crittografico è la polarizzazione lineare,

[6]Un fascio di fotoni di questo tipo viene detto "non polarizzato" a differenza di un fascio "polarizzato" in cui tutti i fotoni del fascio hanno la stessa polarizzazione.

più specificamente le polarizzazioni lineari a 0, 45, 90, e 135 gradi (vedi fig. a pag. 22). Ogni coppia di polarizzazioni ortogonali (0-90 e 45-135) sono modi differenti di rappresentare l'informazione binaria: di conseguenza, queste polarizzazioni del fotone possono essere utilizzate per far trasportare allo stesso un bit della chiave condivisa tra Alice e Bob[7].

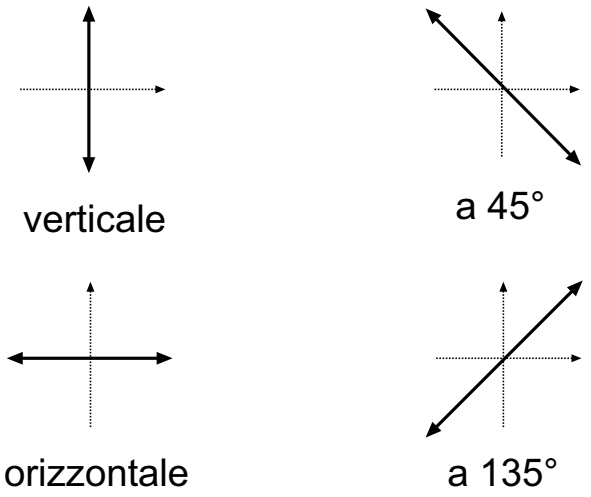


Figura 1.7: Polarizzazioni lineari usate nella crittografia quantistica

Esistono comunque altre possibili polarizzazioni utilizzabili ai fini crittografici, come la polarizzazione circolare o la polarizzazione ellittica, ma di fatto lo standard ad oggi utilizzato è la polarizzazione lineare nelle forme viste.

Processo di polarizzazione e proprietà

A questo punto, il problema è capire in che modo, a partire da un fascio di luce non polarizzato, si possa riuscire ad ottenere fotoni nelle polarizzazioni che abbiamo visto ed in che modo questo sia sfruttabile per creare un sistema quantistico di distribuzione di chiavi sicure, utilizzando i fotoni come "portatori di informazione".

Dato un fascio di luce (un insieme di fotoni con polarizzazioni diverse), si ottengono

[7]Di norma 0 gradi=0, 90 gradi=1, e analogamente 45 gradi=0 e 135 gradi=1.

a partire da esso una serie di fotoni polarizzati in una certa direzione utilizzando uno strumento opportuno detto *polarizzatore*. Il polarizzatore è un filtro che lascia passare i fotoni di un fascio che abbiano la stessa polarizzazione del polarizzatore.

L'esempio più banale di polarizzatore sono gli occhiali da sole. Questi, se indossati correttamente, fanno passare poca luce perché permettono unicamente il passaggio dei fotoni che sono polarizzati verticalmente (cioè con direzione di oscillazione perpendicolare al piano terrestre), i quali sono in quantità minore nei raggi solari che vengono riflessi dagli oggetti della realtà, rispetto a quelli polarizzati verticalmente: di fatto, quindi, fanno passare pochi fotoni sul totale. Gli stessi occhiali da sole, però, se messi in verticale fanno passare molta più luce in quanto permettono il passaggio ai fotoni la cui direzione di polarizzazione è orizzontale, aumentando la quantità di luce anziché diminuirla.

Un tipo molto particolare di polarizzatore sono i *cristalli birifrangenti*: questi cristalli hanno la caratteristica che se bombardati di fotoni lungo una certa direzione (asse ottico del cristallo) suddividono i fotoni in due gruppi. Più precisamente, i fotoni che hanno direzione di polarizzazione perpendicolare all'asse ottico passano senza subire alcuna perturbazione, mentre se hanno una direzione parallela all'asse ottico, vengono deviati di 90 gradi. Ogni fotone che ha una polarizzazione intermedia viene deviato o lasciato passare con una probabilità direttamente proporzionale all'angolo di polarizzazione (i.e. se l'angolo è molto vicino allo zero, cioè parallelo rispetto all'asse ottico, ha molta più probabilità di essere deviato piuttosto che di passare).

I fotoni che hanno un angolo di polarizzazione di ± 45 *gradi hanno una identica probabilità di passare o di essere deviati.* Questo fa intuire in che modo l'utilizzo delle due basi possa essere usato per motivi di sicurezza: se un bit di informazione viene codificato polarizzando il fotone in una certa base, la misurazione di quel fotone sull'altra base (quella ruotata di 45 gradi) darà una misurazione totalmente casuale e priva di significato.

Inoltre, vi è un'altra proprietà dei fotoni polarizzati che va considerata (fig. a pag.

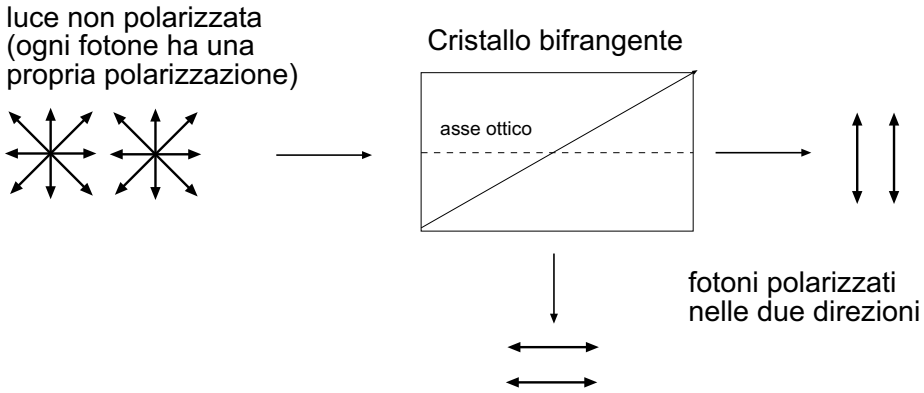


Figura 1.8: Polarizzazione con cristallo birifrangente

24): se un fascio è polarizzato in una certa direzione, l'interazione del fascio con un altro cristallo che ha una polarizzazione diversa, porterà una parte dei fotoni in ingresso a passare mentre una parte verrà riflessa: il numero totale di fotoni passanti sarà dunque minore di quella in ingresso, con la conseguente diminuzione dell'intensità del fascio. Questa proprietà permette quindi di riuscire a monitorare la presenza di un polarizzatore non previsto (che potrebbe essere di Eve), verificando una diminuzione dell'intensità del fascio rispetto alle attese.

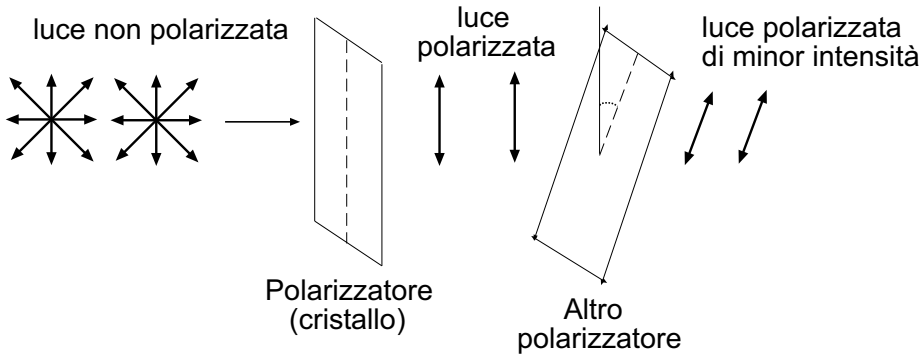


Figura 1.9: Diminuzione dell'intensità del fascio a causa dell'inserimento di un altro polarizzatore

Per capire meglio in che modo queste proprietà fisiche possano essere sfruttate, si rimanda alla sezione successiva riguardante il BB84, un protocollo che sfrutta questo aspetto per creare un metodo di distribuzione di chiavi totalmente sicuro.

1.3 Il protocollo BB84

Il protocollo BB84 (da Bennett e Brassard, gli autori e 1984 l'anno di pubblicazione) sovrintende lo scambio di chiavi random su un canale quantistico, sfruttando le proprietà fisiche viste finora. Il protocollo garantisce uno scambio efficiente e sicuro delle chiavi tra le due parti, descrivendo le fasi che devono susseguirsi, ma lasciando la possibilità di implementare ognuna di tali fasi in maniera totalmente libera, purchè siano soddisfatti i requisiti di ogni singola fase. Questo aspetto garantisce sia una grande adattabilità del protocollo, che può essere utilizzato in diversi contesti di sicurezza, sia molta libertà dal punto di vista implementativo.

1.3.1 FASE 1: La generazione della chiave

Nella sua prima formulazione, il protocollo BB84 nasce per essere applicato in sistemi di crittografia quantistica di *prima generazione*, detti **a singolo fotone**. L'idea base di questi sistemi era che una delle due parti, sia Alice, generasse la chiave e la inviasse successivamente alla seconda (Bob) attraverso il canale quantistico.
Per poter generare la chiave random si potrebbe pensare di partire da una base (ad esempio quella lineare) ed associare ad una polarizzazione il valore di un bit:
$0 \longrightarrow V$ $1 \longrightarrow H$
Per generare la chiave random, Alice disporrebbe quindi di due polarizzatori, uno per V ed uno per H e dovrebbe semplicemente scegliere, per ogni fotone generato col laser, o un polarizzatore o l'altro in maniera casuale, ottenendo così una sequenza di fotoni polarizzati, che rappresentano una chiave random da inviare successivamente a Bob. Un approccio di

questo tipo è semplice, sfrutta appieno le proprietà quantistiche, ma non ha alcun vantaggio dal punto di vista della sicurezza rispetto ad un approccio classico: in questo caso Eve, nota la base che usano Alice e Bob per comunicare sul canale quantistico, potrebbe, come in fig. a pag. 19, leggere il fotone polarizzato nella base lineare e, in funzione del valore letto, polarizzare un nuovo fotone identico da inviare a Bob.

Un attacco di questo tipo (fig. a pag. 27) passerebbe totalmente inosservato alle due parti esattamente come nel caso classico: di conseguenza non sarebbe giustificabile un simile uso del canale quantistico poiché è costoso e non permette di trarre alcun beneficio rispetto all'approccio tradizionale.

In realtà si può render vincente questo approccio con piccoli accorgimenti e l'utilizzo di entrambe le basi di polarizzazione[8] anziché una sola. Lo schema rimarrebbe identico ma invece di scegliere tra due polarizzazioni di una base, Alice dovrebbe scegliere in primis tra una delle due basi (in maniera ovviamente random) e polarizzare il fotone con uno dei due polarizzatori di quella base; all'atto pratico, quindi, Alice dispone di *quattro* polarizzatori tra cui scegliere casualmente per polarizzare ogni fotone. Alla ricezione dei fotoni inviati da Alice, Bob sceglierà in maniera analoga e sempre casualmente una base di misura per ognuno dei fotoni ricevuti.

Alla fine di questo scambio di dati sul canale quantistico, le due parti sono entrambe in possesso di una prima chiave detta **rough key**: poiché la base di misurazione scelta per ogni singolo fotone da Alice e Bob può essere differente, occorre a questo punto eliminare quei fotoni per cui le due parti abbiano adottato una base diversa. Per determinare quali di questi fotoni siano da eliminare, Bob dichiara su un canale pubblico, la lista delle basi (ma non delle polarizzazioni) che ha usato: Alice, leggendo la lista e confrontandola con quella delle basi da lei usate, può facilmente capire quali deve eliminare. A questo punto le due parti tengono soltanto i dati dei fotoni (cioè i bit di chiave) per i quali hanno scelto una

[8] Si supponga, nella base diagonale, $-45 \longrightarrow 0$ e $+45 \longrightarrow 1$.

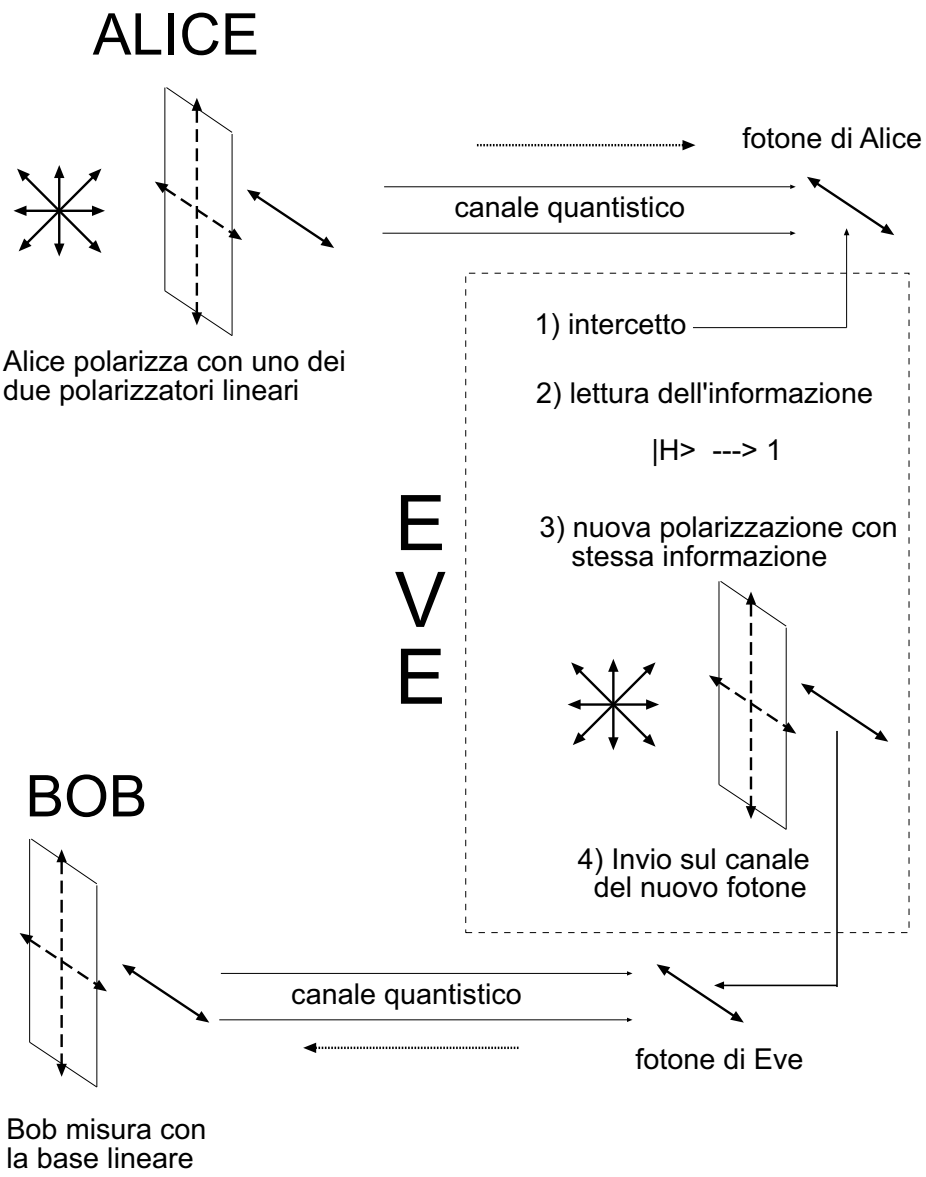


Figura 1.10: Possibile attacco di Eve su polarizzazione ad unica base

base comune.

Trasmissione Quantistica														
Bits Random di Alice.....	0	0	1	1	1	0	1	1	0	0	0	0	1	0
Basi Random di Alice.....	L	D	L	L	D	L	D	D	D	L	D	L	L	L
Fotoni inviati.....	|	\	-	-	/	|	/	/	\	|	\	|	-	|
Basi Random di Bob.....	D	D	L	D	L	L	D	L	L	L	L	D	D	L
Bits come ricevuti da Bob.....	X	0	1	X	X	0	1	X	X	0	X	X	X	0
Discussione su canale pubblico														
Lista delle basi di Bob.....	D	D	L	D	L	L	D	L	L	L	L	D	D	L
Alice dice quali coincidono.....		ok	ok			ok	ok			ok				ok
Informazioni condivise (no Eve).....		0	1			0	1			0				0

Figura 1.11: Esempio di scambio di chiave con due basi di misurazione

Una modifica di questo tipo al sistema precedente, benché semplice, complica molto la vita di Eve il cui tentativo di *sniffing* a questo punto non può più passare inosservato. Con due basi di polarizzazione, Eve non conosce più a priori la base ma è costretta ad *indovinare* per ogni fotone quale base di polarizzazione Alice abbia scelto. Se sceglie la base diversa, il risultato che può ottenere è totalmente casuale e privo di significato (i.e. se Eve misura con la base diagonale un fotone polarizzato in quella lineare può ottenere o un 1 o uno 0 con eguale probabilità, come visto nella sezione "Processo di polarizzazione e proprietà").

Una volta acquisito il risultato "presunto", Eve inoltrerà a Bob un fotone polarizzato sulla base errata. Dal canto suo, alla ricezione di ogni fotone anche Bob sceglierà casualmente una base in cui misurarne la polarizzazione.

Si possono a questo punto fare alcune considerazioni su questo algoritmo di scambio e cercare di determinare in che percentuale la presenza di Eve può essere rilevata:

- Poiché per ogni fotone sia Alice che Bob scelgono tra due basi con eguale probabilità,

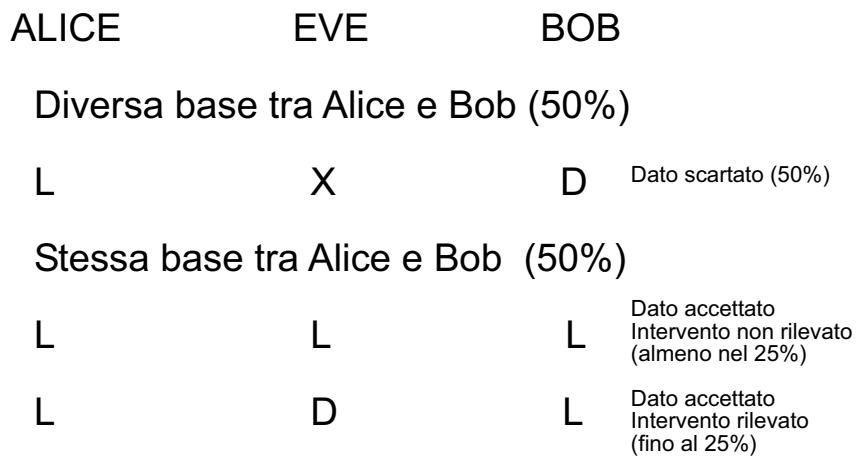


Figura 1.12: Situazioni possibili in un approccio a due basi

su un numero abbastanza grande di fotoni si avrà che le basi coincideranno nel 50% dei casi. Di conseguenza il 50% della rough key viene scartato per incompatibilità delle basi scelte da Alice e Bob. Ogni intervento di Eve sul 50% della rough key che viene scartato non viene considerato poiché quella parte di chiave stessa non verrà mai utilizzata.

- Sulla restante parte di chiave una situazione analoga alla precedente vale per Eve: Eve, esattamente come Bob, deve scegliere una base e si può analogamente supporre che nella metà dei casi la indovini e nell'altra metà la sbagli.

- Nel caso in cui la indovini, è chiaro che, per il processo mostrato prima, l'intervento di Eve passerebbe inosservato, mentre nel caso in cui sbagli, sarebbe possibile notare una differenza di valori di chiave tra Alice e Bob benché le basi da loro scelte siano coincidenti.

- Di conseguenza, si può pensare che in una comunicazione sniffata da un attaccante si notifichino fino al 25% di differenze di valori di chiave tra le due parti.

Note sulle comunicazioni sul canale pubblico

L'uso di un canale pubblico per rendere nota la lista delle basi scelte deve tenere conto di tutti i limiti di sicurezza intrinseci di un canale classico: Eve potrebbe frapporsi tra Alice e Bob secondo un tipico attacco di tipo Man-In-The-Middle e portare avanti due comunicazioni separate con Alice e Bob, gestendo le liste di basi a proprio vantaggio ed in funzione delle basi da essa scelte in fase di sniffing. Un attacco di questo tipo, permetterebbe ad Eve di ottenere due chiavi randomiche diverse per Alice e Bob da utilizzare per comunicare con le due parti. In pratica Eve viene a sostituirsi, per ognuna delle due parti, alla parte con cui questa deve comunicare: ponendosi tra Alice e Bob svolge le funzioni di Bob per Alice e di Alice per Bob.
Un attacco Man-In-The-Middle sulle liste di basi potrebbe seguire le seguenti fasi:

1. Eve intercetta la lista di Bob ed invia ad Alice la lista delle sue basi.

2. Riceve le conferme da Alice sulla sua lista e quindi determina una chiave random da usare con Alice.

3. Per ogni base *concordante* con quella di Bob, dà conferma a Bob ottenendo alla fine una seconda chiave da usare invece con Bob.

ALICE	EVE	BOB
1 L	>no D >no	L
2 D	>ok D >no	L
3 D	>ok D >ok	D
4 L	>ok L >no	D
5 D	>no L >ok	L

Figura 1.13: Attacco di Eve sul canale pubblico

Un attacco di questo tipo sul canale pubblico rischia di minare la sicurezza dell'intero sistema e rende pressochè inutile il processo di scambio delle chiavi sul canale quantistico.

Il punto di forza dell'attacco portato da Eve è nello scambio delle identità, cioè nel fatto che Eve riesce ad ingannare le due parti facendosi identificare per l'altra. L'attacco è però scongiurabile attraverso un meccanismo di **autenticazione** attraverso il quale le due parti ottengano la certificazione della provenienza di tale messaggio. Ogni messaggio deve quindi essere autenticato: l'autenticazione può essere realizzata tramite una **firma digitale** che il mittente del messaggio appone al fondo dello stesso (sia questo la lista delle basi o delle conferme).

Per la realizzazione di tale sistema di firma, si rende necessaria la condivisione di un segreto (una chiave) tra le due parti prima dell'inizio dell'intera sessione, che Eve non potrebbe violare a meno di non conoscere la chiave stessa: finchè il segreto rimane solo appannaggio di Alice e Bob, le due parti possono essere certe della provenienza delle informazioni ricevute. Il metodo da utilizzare per realizzare la firma (algoritmi asimmetrici basati su certificazione oppure simmetrici) ed i tempi di validità di tale segreto sono problemi critici da considerare, poiché la gestione errata o superficiale di questi aspetti rischia di rendere totalmente inutile il meccanismo di difesa visto e lascerebbe via libera ad Eve per l'attacco Man-In-The-Middle.

Il problema del rumore

Finora abbiamo trattato implicitamente il canale quantistico come "ideale", ovvero totalmente esente da errori di trasmissione o di ricezione. Su un canale ideale, gli unici errori e discrepanze tra i bit di chiave delle due parti, per coppie di basi identiche, sono dovuti all'intervento di Eve. In un contesto applicativo reale, però, la trasmissione sul canale quantistico è soggetta ad errori dovuti alla non perfezione dei sistemi di ricezione, dei polarizzatori e ad eventuali ritardi dei fotoni o disturbi sul canale stesso. Inoltre occorre tenere presente che i fotoni sono ottenuti attraverso un laser il cui comportamento è imprevedibile: la loro generazione non è "on-demand", ovvero non si riescono ad ottenere i fotoni esattamente quando si richiedono ma bisogna, al contrario, polarizzarli quando sono disponibili

(nel 90% dei casi le pulsazioni del laser non generano fotoni).

Tutte queste imperfezioni possono portare a misurare un valore di chiave anzichè un altro e di fatto quindi degli errori intrinseci del sistema che devono essere presi in considerazione e corretti (esiste una opportuna fase del protocollo BB84 detta Error Correction - si veda dopo). Da un punto di vista squisitamente teorico, questi errori vengono comunque considerati come causati da Eve, nel rispetto dell'ipotesi di porsi sempre nel caso peggiore.

Benché ogni errore sia considerato come causato da un'intrusione, occorre però saper valutare attentamente il peso dell'errore e poter decidere, a fronte di una certa percentuale di errore sulla chiave **sifted**[9], se quell'errore sia dovuto ad uno sniffing o ad errori di trasmissione.

Questa è un'altra fase di importanza cruciale in quanto la valutazione dell'errore sulla chiave determina l'eliminazione (se si conclude che gli errori siano causati da Eve) o l'accettazione della chiave (se si conclude che gli errori dipendano da errore del canale o da un intervento "intermittente" di Eve, comunque non sufficiente a compromettere il livello di sicurezza richiesto alla chiave).

Di questo aspetto si occupa la fase successiva del protocollo BB84, detta "*stima del QBER*".

1.3.2 FASE 2: La stima del QBER

A questo punto entrambe le parti hanno una chiave sifted condivisa, ma non è detto che entrambe le copie siano identiche: al contrario, possono avere delle differenze sulle cui cause si è parlato in precedenza. Al fine di poter avere una chiave applicabile per One Time Pad, occorre, in primis, verificare che la chiave condivisa non sia "compromessa", ovvero non sia stata in qualche modo intercettata; se la chiave viene ritenuta valida occorre poi correggere le eventuali differenze per arrivare ad avere una chiave unica per entrambe le parti, in modo tale che sia utilizzabile per cifrare messaggi. Per poter realizzare questi passi, occorre

[9] La chiave SIFTED è la chiave rimasta dopo il confronto delle basi, eliminando i bit di chiave per i quali le due parti hanno scelto basi diverse.

studiare il QBER. QBER è l'acronimo di *Quantum Bit Error Rate* ed indica la percentuale di bit di chiave sifted differenti tra Alice e Bob. Per determinare il QBER occorre sacrificare una parte della chiave sifted e inviarla sul canale pubblico all'altra parte (ovviamente sempre con l'autenticazione già usata per l'invio della lista delle basi): tale parte di chiave è sacrificata perché resa pubblica e inutilizzabile quindi per cifrare, ma permette all'altra parte di confrontarla con la medesima parte di chiave in suo possesso e determinare il numero delle differenze. Il QBER quindi viene calcolato come il rapporto tra i bit differenti e la lunghezza del pezzo di chiave usato per la stima:

$K = parte\ della\ chiave\ usata\ per\ la\ stima$

$N = numero\ di\ differenze\ tra\ Alice\ e\ Bob\ su\ K$

$QBER = \frac{N}{|K|}$

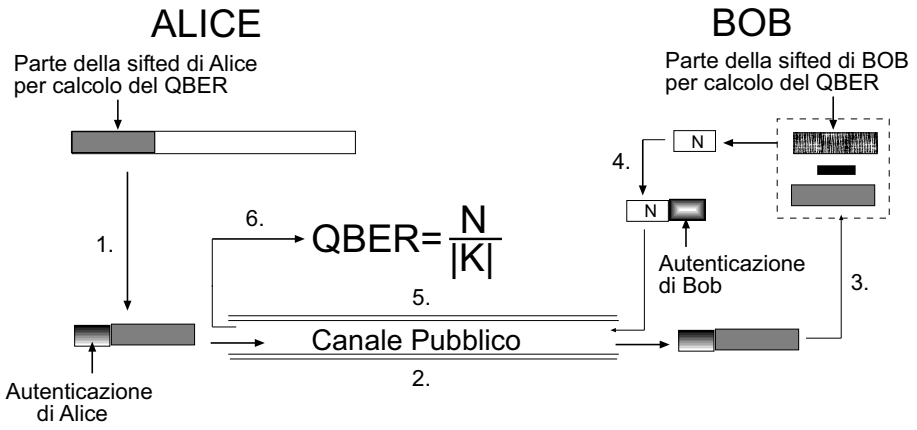


Figura 1.14: Algoritmo per la stima del QBER

La stima del QBER è un processo molto delicato poiché la si calcola su una parte di chiave K ed il valore stimato deve essere "buona" per tutto la parte di chiave restante, ovvero il comportamento degli errori nella chiave residua deve rispecchiare l'andamento evidenziato dalla stima, pena l'errata correzione degli errori in essa contenuti. Il QBER

quindi non deve essere *sottostimato* né *sovrastimato*.

La sovrastima del QBER porta a considerare più errori di quelli che effettivamente sono presenti, con il rischio, qualche volta, di ritenere la chiave insicura anche quando non lo è veramente. Anche nel caso in cui ciò non avvenisse, supporre di avere più errori comporta sacrifici non necessari di parti di chiave nelle fasi successive di Error Correction e Privacy Amplification, con la conseguenza di perdere bit utili per la cifratura.

D'altro canto, la sottostima del QBER porta invece a considerare meno errori di quelli che effettivamente sono presenti nella chiave sifted, con una conseguente presenza di discrepanze tra la chiave di Alice e Bob. Il throughput della chiave (cioè il numero dei bit di chiave condivisa prodotta per secondo) è superiore ma la "qualità" della chiave è decisamente inferiore.

Occorre quindi che la stima sia accurata ed è consigliabile verificare, per ogni blocco di chiave finale che si ottiene col BB84, che tale stima continui ad essere valida ed eventualmente modificarla opportunamente attraverso studi statistici (varianza, scarto quadratico medio), al fine di mantenerla consistente con l'effettiva presenza di errori nella chiave.

Una buona percentuale di chiave da sacrificare potrebbe essere sul 10%, anche se, in generale, dipende dal contesto e dall'ambiente in cui il sistema crittografico deve funzionare.

1.3.3 FASE 3: Error Correction - Correzione degli Errori della chiave

La stima del QBER permette di determinare la quantità di errori che si dovranno correggere nelle chiavi in possesso di Alice e Bob, al fine di renderle uguali. Il passo successivo consiste nella correzione vera e propria di tali errori: la rilevazione richiede ad Alice e Bob di sfruttare nuovamente il canale pubblico per scambiarsi informazioni riguardo la natura della chiave. È chiaro che la pubblicazione di tali informazioni (sempre con autenticazione del messaggio) fornisce ad Eve dati importanti per risalire alla chiave stessa. Per poter garantire la sicurezza, alla fase di Error Correction seguirà pertanto una fase di Privacy

Amplification, che, al costo di altri bit di chiave sacrificati, farà in modo che le informazioni carpite da Eve durante la fase di Error Correction le siano pressochè inutili per risalire alla chiave.

Per capire come può funzionare un processo di correzione e come questo possa dare informazioni all'attaccante, si vedrà il metodo (Parity Check) proposto dagli stessi Bennett e Brassard, il quale, benché sia teoricamente corretto, non ha una applicazione in qualche implementazione di sistemi di crittografia quantistica, in quanto "svela" troppe informazioni e lascia molti aspetti pratici non definiti.

Correzione attraverso Parity Check

L'idea è di dividere in blocchi di m bit la chiave e calcolare su ognuno di questi la parità (cioè si conta il numero di bit a 1: se il numero è dispari la parità è 1 altrimenti 0) ed inviarla *su canale pubblico* all'altra parte. La parte che riceve il valore della parità lo confronta con la parità del medesimo blocco sulla sua parte di chiave. A questo punto si possono verificare due situazioni:

- La parità coincide. In questo caso si passa al blocco successivo.
- La parità non coincide. In questo caso il blocco è diviso in due sottoblocchi di $\frac{m}{2}$ bit e l'intero processo è riapplicato su entrambi i sottoblocchi.

Ogni informazione sulla parità può essere osservata da Eve che viene ad ottenere informazioni sulla chiave che le possono permettere di ricavarla almeno in parte.

Questo algoritmo di correzione ha tuttavia una serie di problemi. La parità permette di determinare la presenza di un numero dispari di differenze tra le due chiavi all'interno di ogni singolo blocco, un numero pari di errori non verrebbe riconosciuto: poiché la coincidenza della parità non è sufficiente a garantire l'assenza di errori, in un blocco in cui la parità è verificata, si dovrebbe comunque applicare sullo stesso qualche altro procedimento per verificare la presenza di un numero pari di errori. Tra le proposte, vi è quella di

ALICE	Par. Alice	BOB	Par. Bob
0100110	1	0100110	1
0000110	0	0100110	1
0000010	1	0100110	1

scartare un bit di chiave e ricalcolare la parità sui restanti bit, o di dividere in sottoblocchi anche blocchi pari e ricalcolare su questi la parità: tutti questi procedimenti portano però a perdere una grande quantità di bit di chiave e non è chiaro quando sia lecito interrompere il procedimento ricorsivo di divisione in sottoblocchi; di conseguenza, da un punto di vista implementativo, un algoritmo di correzione di errori di questo tipo risulta inadeguato ed inefficiente.

Gli algoritmi di correzione che vengono utilizzati nella realizzazione pratica di un sistema di crittografia quantistica sono i classici codici a correzione lineari e ciclici.

1.3.4 FASE 4: Privacy Amplification

Dopo la fase di correzione di errore, Eve rischia di avere molte informazioni sulla chiave condivisa, troppe in molti casi per garantire che la chiave possa essere ritenuta ancora sicura. Le informazioni di Eve provengono sia dalla fase di scambio sul canale quantistico che dalle informazioni rivelate dalle due parti in fase di Error Correction: durante la fase di scambio sul canale quantistico, un intervento "attento" di Eve, ad esempio non continuo e per periodi brevi, non verrebbe considerato sufficiente per la violazione della sicurezza della chiave (basso valore del QBER[10]), in quanto le informazioni carpite sarebbero comunque troppo poche. Sebbene questo possa essere vero prima della fase di Error Correction, questo potrebbe non essere più vero se alle informazioni già in possesso di Eve si aggiungono quelle ottenute durante la fase di correzione: a questo punto, l'insieme delle informazioni in possesso di Eve potrebbe diventare troppo *consistente* e la sicurezza dell'intera chiave potrebbe essere di fatto compromessa.

[10]Se il QBER è il 5% Eve potrà conoscere al più quella percentuale della chiave.

La fase di Privacy Amplification si rende quindi necessaria per riportare il livello di conoscenze di Eve sotto una soglia tale da poter garantire la sicurezza della chiave finale, al costo di ulteriori bit di chiave sacrificati.
Più nello specifico, se la chiave dopo la fase di Error Correction è lunga n bit, dopo la fase di Privacy Amplification sarà lunga r ($r < n$): la nuova chiave ottenuta è il risultato dell'applicazione di una opportuna **funzione di compressione** sulla chiave di n bit in input.

La funzione di compressione

La funzione di compressione $g : \{0,1\}^n \longrightarrow \{0,1\}^r$ usata nella Privacy Amplification fa parte di una classe di funzioni detta $UNIVERSALE_2$.

Def. Una classe G di funzioni $A \longrightarrow B$ è detta $UNIVERSALE_2$ se $\forall x_1, x_2$ distinti la probabilità che $g(x_1) = g(x_2)$ è $\leq \frac{1}{|B|}$ con g funzione scelta *casualmente* tra le funzioni in G.

La dimensione della chiave in output (r) deve essere più piccola della differenza tra la dimensione della chiave in input e le informazioni possedute da Eve.
Le informazioni in possesso di Eve sono $k = x + y$ con x i bit di chiave conosciuti con lo scambio di chiavi sul canale quantistico (deriva direttamente dal QBER) ed y i bit di chiave conosciuti con la Error Correction (si può fare una stima pessimistica in funzione del codice di correzione adottato).
Occorre inoltre tenere presente (ed anche questo determina il valore di r) del livello di sicurezza che si richiede, sia s ($0 < s < n - k$), che determina direttamente quanto può essere il numero massimo dei bit di chiave che Eve conosce alla fine del processo di Privacy Amplification ($\leq \frac{2^{-s}}{ln(2)}$).

La dimensione della chiave finale è data da $r = n - k - s$. In pratica l'applicazione della funzione g non equivale ad altro che a calcolare una nuova *parità* di r-bit sulla chiave, che però adesso rimane segreta e non viene comunicata su alcun canale. La percentuale di bit persi con la Privacy Amplification è data da $\frac{s+k}{n}$.
La chiave in output alla fase di Privacy Amplification è la chiave "buona" che verrà usata per cifrare con l'One Time Pad.

1.3.5 Considerazioni finali sul protocollo

Le quattro fasi del protocollo BB84 non sono tutte uguali, ma mentre le prime due sono molto economiche dal punto di vista computazionale, le ultime due, a causa dei calcoli e dei controlli svolti, sono decisamente onerose. Inoltre, attraverso le fasi, la chiave subisce una progressiva diminuzione della dimensione dalla sifted key a quella finale. Nonostante questo, il BB84 è un buon algoritmo per lo scambio di chiavi quantistiche: quello che è importante non è cercare un algoritmo migliore che porti meno riduzioni, ma è compito di chi implementa un sistema quantistico di "configurare" il BB84 in modo tale che il *throughput*, cioè il numero di bit di chiave finale prodotti dall'algoritmo nell'unità di tempo, sia soddisfacente per i requisiti del sistema da realizzare. Ad esempio, adottando un buon algoritmo di correzione degli errori o determinando un campione il più piccolo possibile per la stima del QBER, si può ottimizzare il BB84 in modo tale che le perdite siano irrilevanti ed il risultato che verrà fornito sia comunque soddisfacente. Nella sezione successiva, analizzando il Q-CRYPT, si vedrà un esempio di come si possono personalizzare le diverse fasi e di quali parametri entrano effettivamente in gioco.

1.4 Il Q-CRYPT

Il protocollo BB84 nasce per i cosiddetti sistemi di crittografia quantistica di prima generazione, cioè quelli *a singolo fotone* presentati finora, dove Alice polarizza e Bob legge i fotoni

polarizzati. Tuttavia, nel 1991 Ekert avanzò l'ipotesi che si potessero usare gli stati ENTANGLED idealizzati da Einstein, Podolsky e Rosen (detti anche stati EPR) per realizzare un sistema di crittografia quantistica la cui sicurezza sia garantita dalla *disuguaglianza* di Bell.

1.4.1 Entanglement e crittografia quantistica

Il concetto base intorno a cui ruota l'idea del Q-CRYPT è quello di **entanglement**. L'entanglement è un fenomeno quantistico per il quale non esiste un analogo nel caso classico, in cui ogni "stato" (quantistico) di un insieme di due o più sistemi, dipende dallo stato degli stessi anche se sono spazialmente distanti, ovvero se si trovano dislocati nello spazio e non sono a contatto. Uno stato *entangled* implica delle correlazioni tra le osservabili dei sistemi coinvolti.

Un esempio potrebbe essere un sistema di due particelle: lo stato entangled del sistema delle due particelle implica che la misurazione dello spin di una delle due, determini univocamente lo spin della seconda particella, anche se le due particelle si trovano in luoghi differenti. È chiaro quindi che in un sistema entangled la misura di un sottosistema influenza anche lo stato degli altri sottosistemi.

I sistemi entangled che interessano in chiave crittografica sono sistemi di due fotoni legati da un rapporto di entanglement. Un coppia di fotoni entangled può essere generata sfruttando le caratteristiche di un *cristallo non lineare* (tipo il β-Borato di Bario): un fotone che urta un cristallo di questo tipo, decade in *due* fotoni di lunghezza d'onda maggiore (cioè con frequenza -e quindi energia- minore) la cui energia totale è pari a quella del fotone originale. I due fotoni che escono dal cristallo sono tali che uno è allineato in un cono di luce polarizzato orizzontalmente ($\leftrightarrow$) e l'altro in un cono di luce polarizzato verticalmente($\updownarrow$).

Se uno dei due fotoni incontra un polarizzatore verticale e passa, l'altro non deve passare, perché polarizzato necessariamente in maniera opposta secondo i principi della meccanica

quantistica: si noti che la grandezza di tale proprietà sta nel fatto che si garantisce che se uno viene misurato in un certo stato di polazizzazione, l'altro abbia lo stato opposto in qualunque posto esso sia[11].

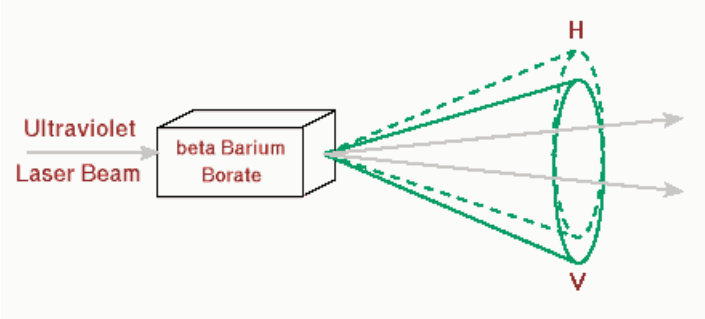


Figura 1.15: Processo di generazione di coppie entangled

Il processo di generazione dei fotoni avviene in sostanza attraverso i seguenti passi:

1. Un laser *ultravioletto* invia un singolo fotone verso il β-Borato di Bario.
2. Appena il fotone passa attraverso il cristallo, c'è una possibilità che decada.
3. Se decade, due fotoni escono dal cristallo.
4. I fotoni sono considerati **entangled**.

È importante notare che si usa un laser ultravioletto perché ha una frequenza molto alta, e più grande è la frequenza, maggiore la probabilità che decada il fotone.

I due fotoni che escono, come si è detto, appartengono uno al cono di polarizzazione orizzontale (colore rosso), ed uno a quella verticale (colore blu).
Il cristallo può essere modificato in modo tale che i due coni di polarizzazione si *intersechino*:

[11] Questa relazione di dipendenza non legata alla località spaziale portò Einstein a non essere convinto della bontà della teoria quantistica.

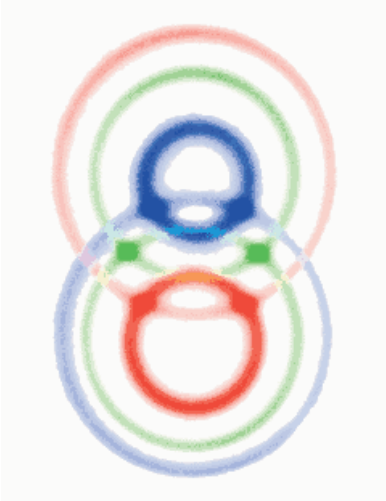


Figura 1.16: Esempio di coni in uscita dal cristallo di Bario

il risultato che si ottiene è di avere coppie di fotoni che possono trovarsi nell'intersezione dei due coni (fig. a pag. 41).

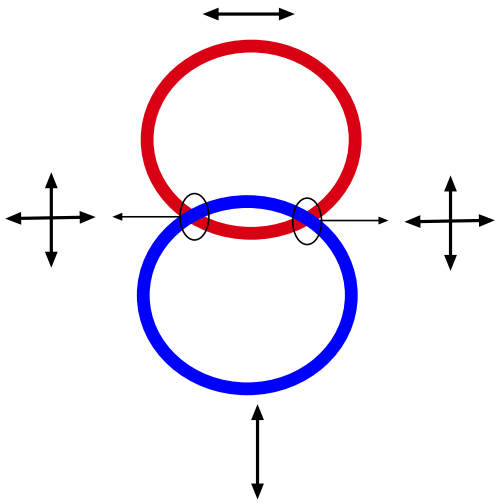


Figura 1.17: Coni di uscita intersecati

Tali coppie hanno **una polarizzazione indefinita**, ma continuano a mantenere la

caratteristica che la misurazione della polarizzazione di uno dei due fotoni determina univocamente anche la polarizzazione dell'altro (*opposta* a quella del primo), in qualunque posto esso si trovi: ad esempio se in fase di misurazione la polarizzazione del primo risulterà essere orizzontale, quella del secondo sarà necessariamente verticale. Anche queste coppie di fotoni, quindi, sono da considerarsi a tutti gli effetti *entangled*. Nel sistema Q-CRYPT vengono utilizzati proprio queste coppie di fotoni per permettere lo scambio dei bit di chiave sul canale quantistico: il cristallo di Bario viene bombardato continuamente ma vengono utilizzate solamente le coppie di fotoni polarizzati nell'intersezione dei due coni.

1.4.2 Architettura del sistema Q-CRYPT

L'utilizzo di coppie di fotoni entangled permette di definire una nuova architettura (fig. a pag. 42) per garantire ad Alice e Bob di poter generare una chiave random per l'One Time Pad.

La differenza fondamentale di questo sistema, rispetto a quello a singolo fotone, è che la chiave non viene più generata da una delle due parti ed inviata alla seconda ma si utilizza un generatore di fotoni entangled, in un luogo diverso dalle sedi di Alice e Bob, col quale generare coppie da inviare alle parti.

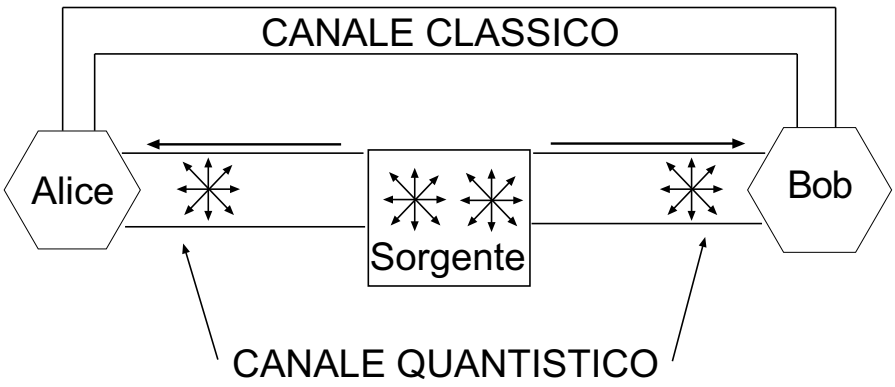


Figura 1.18: Architettura del QCRYPT

Come si può vedere, l'architettura cambia solo nella parte quantistica rispetto all'approccio della precedente generazione: il resto rimane invariato e le due parti continuano ad utilizzare un canale pubblico che anche qui verrà sfruttato per l'implementazione delle varie fasi del protocollo BB84.

Lo schema descritto per lo scambio di fotoni sul canale quantistico permette ad Alice e Bob di avere quindi due sequenze identiche di informazioni come sequenze di fotoni polarizzati: è chiaro però che, se l'entanglement garantisce che i fotoni della coppia siano *anticorrelati*, una delle due parti deve invertire l'informazione che ottiene dal fotone ricevuto (fig. a pag. 43)

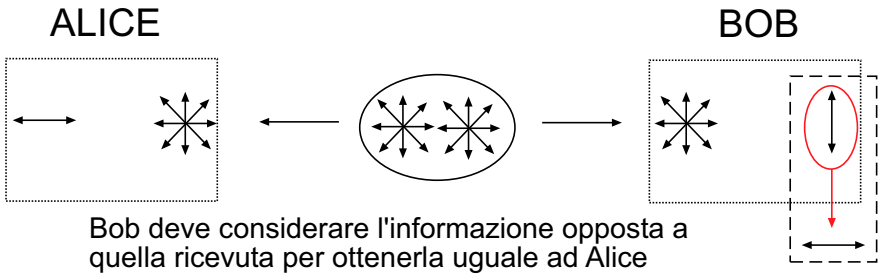


Figura 1.19: Condivisione dell'informazione tra Alice e Bob

Finora si è presentato il sistema QCRYPT ragionando solo sulla base lineare ($\updownarrow\leftrightarrow$), tuttavia per l'applicabilità del protocollo BB84 e per la mancanza di sicurezza nell'usare un'unica base (come mostrato in precedenza), occorre utilizzare sia la base lineare che la base diagonale(± 45). Questo è possibile perché lo stato entangled della coppia gode di una proprietà detta *invarianza rotazionale* che garantisce che l'anticorrelazione tra i fotoni sussista a prescindere dalla base di misurazione.

A questo punto il nuovo sistema ha tutte le caratteristiche per poter implementare il protocollo BB84 per la distillazione della chiave.

1.4.3 Implementazione del BB84 nel QCRYPT

Scopo di questa sezione non è rivisitare il protocollo BB84 ma descrivere le situazioni di cui occorre tenere conto in un contesto applicativo reale. Lo scopo è di determinare, per le diverse fasi, i parametri essenziali di cui poi occorrerà tenere conto e su cui sia necessario riflettere per poter implementare un sistema efficiente.

La distribuzione delle chiavi

La prima fase del protocollo, benché sia concettualmente e computazionalmente abbastanza semplice, richiede di dover tenere conto di diversi aspetti, affinché sia possibile una sua effettiva implementazione.

In primo luogo, in un sistema effettivamente funzionante, Alice e Bob riceverebbero un grande quantità di fotoni entangled che devono riuscire ad identificare: in pratica entrambi devono essere in grado di riconoscere, per ogni fotone ricevuto, la coppia a cui appartiene. Questo aspetto è molto importante poiché permette ad entrambe le parti il confronto dei dati e delle basi.

Per realizzare un riconoscimento di questo tipo si usa un meccanismo di **timestamping** in cui ogni fotone della coppia viene contraddistinto con un'informazione temporale, di modo che sia possibile alle due parti associare il fotone alla coppia. Tale operazione viene detta *sincronizzazione*.

Il metodo di timestamping utilizzato consiste nell'associare ad ogni fotone una stringa di bit che permetta di sincronizzarlo: nel QCRYPT quindi, le informazioni in output dai misuratori di Alice e Bob sono stringhe di 32 bit, con un bit che indica il valore misurato, un bit per la base in cui è stato misurato quel valore, e 30 bit che indicano il gap temporale tra il momento dell'acquisizione del fotone da parte del misuratore e l'inizio della sincronizzazione. Il bit meno significativo vale 2 nsec, quindi è possibile sincronizzare per periodi di al massimo 2 secondi.

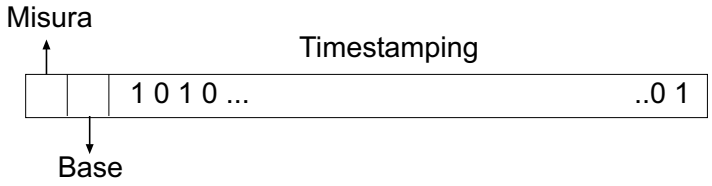


Figura 1.20: Stringa in output dai misuratori

Il timestamping è necessario anche perché molti fotoni in transito subiscono ritardi e molti possono non arrivare a destinazione: in questo modo è facile scartare eventuali fotoni spaiati, ovvero tali che una delle due parti non abbia ricevuto l'accoppiato. Inoltre, occorre tenere presente che nel caso in cui la distanza di Alice e Bob dalla sorgente non sia identica (cosa che verosimilmente accade), i fotoni giungono in tempi diversi alle due parti e che quindi l'identificazione dei fotoni senza un simile meccanismo sarebbe quantomai irrealizzabile.

Oltre al problema del timestamping, un'altra situazione comunque da gestire è la randomicità della scelta dei polarizzatori: Alice e Bob scelgono tra due basi con cui misurare i fotoni in arrivo sul canale quantistico; tale scelta deve essere *random* a tutti gli effetti, pena la perdita di efficacia dell'One Time Pad e, in generale, della possibilità di garantire una sicurezza totalmente incondizionata.

La stima del QBER

Il problema principale della stima del QBER è di determinare la dimensione del campione da sacrificare, onde evitare sovra o sottostime dell'errore. In teoria più è grande il valore, maggiori sono le possibilità che la stima sia accurata; in pratica occorre discernere un limite superiore oltre il quale non ha senso andare, perché l'aumento di bit sacrificati non porterebbe un guadagno apprezzabile in termini di precisione della stima dell'errore. Tale limite potrebbe essere un buon valore di campione.

Il problema è che la determinazione di tale fattore dovrebbe tenere conto di molte variabili, quali la precisione degli strumenti di polarizzazione e di misurazione, la distanza tra Alice e Bob (la percentuale di errore sul mezzo trasmissivo è proporzionale alla lunghezza) e le condizioni ambientali in cui l'intero sistema si trova. Considerare e valutare tutti questi fattori non è semplice e sottolinea come non esista in teoria un valore accettabile in senso assoluto ma si possa determinare solo un range di valori tra cui, statisticamente, conviene scegliere. Sempre nell'ottica di dover determinare un limite superiore, bisogna considerare, oltre che la fallacità degli strumenti, anche il fatto che un campione troppo grande porterebbe a sacrificare una percentuale così grande di bit da rendere praticamente nullo il throughput di chiave finale.

La determinazione di un valore prima dello studio dell'errore, tuttavia, non è sufficiente. Se inizialmente sussistono le condizioni che portano alla determinazione di un certo valore per il campione, è altrettanto vero che nel tempo -e durante la computazione- tale valore potrebbe "perdere" di significato in virtù di una progressiva perdita di precisione degli strumenti. Ma l'errore di stima non tende necessariamente a diventare più grande con il tempo, ma può anche diminuire, ad esempio se le condizioni ambientali migliorano rispetto a quelle iniziali.

È bene quindi tenere sotto costante osservazione la bontà della stima fatta, attraverso operazioni statistiche sui bit di chiave che vengono di volta in volta corretti. Un'idea potrebbe essere di dividere in blocchi la chiave sifted e calcolare, su ognuno di essi, il numero di bit che sono stati effettivamente corretti e calcolarne la varianza o lo scarto quadratico medio con la stima fatta sul campione di chiave attuale: se queste misurazioni portano a sottolineare una significativa discrepanza tra lo stimato e l'effettivamente corretto, si può pensare di modificare run-time la dimensione del campione per i blocchi a seguire, al fine di evitare sovra o sottostime degli errori realmente presenti.

Oltre al problema di determinare un campione di grandezza adeguata, vi è quello di trovare

quale sia la percentuale di errore medio generato dall'imprecisione degli strumenti e dall'ambiente su un certo canale quantistico, in modo tale da giungere ad un QBER massimo dovuto agli errori di trasmissione. I valori di QBER superiori a tale valore potrebbero essere sospetti perché generati, con un'elevata probabilità, da un intervento di Eve. Questo valore limite quindi può essere discriminante per il rifiuto o l'accettazione della chiave stessa.
Il processo per determinarlo viene detto **tuning**; in esso si monitorano le condizioni del canale attraverso test di trasmissione e statistiche sui risultati ottenuti. Questo processo non ha richieste di sicurezza e tutte le informazioni sono scambiate in chiaro poiché, di fatto, non si scambia nessun bit di chiave. Il processo di tuning è obbligatorio ogni volta che si cambia il canale quantistico (se Alice o Bob cambiano locazione), ma può essere richiesto in altre situazioni per vedere se la situazione è in qualche modo cambiata. Ovviamente il tuning non può essere eseguito durante una sessione di scambio di chiave.

1.4.4 La correzione degli errori

La fase di correzione degli errori utilizza algoritmi di correzione classici i cui vantaggi e svantaggi sono noti e la cui efficienza è comunque tale da garantire un'opportuna correzione degli errori nella chiave. Il tipo di codice usato è il BCH (ciclico) che lavora su blocchi di chiave di dimensione fissa. Il parametro più importante da determinare è la distanza di Hamming minima tra due codifiche corrette e dipende direttamente dal numero di errori evidenziati dal QBER: maggiore è la percentuale di errore, maggiore il numero massimo di errori che ci si aspetta di avere nel blocco e, di conseguenza, più grande la distanza di Hamming minima tra due codifiche corrette.
La fase di correzione con BCH, richiede comunque lo scambio di informazioni tra Alice e Bob per verificare lo stato della correzione sul blocco e quindi fornisce informazioni ad Eve, sebbene comunque minori dell'algoritmo di Parity Check.

Capitolo 2

Il problema del KEY STORAGE

2.1 Lo scenario e le possibili minacce

Il processo di generazione della chiave preclude di attivare un sistema complesso il cui utilizzo richiede tempi di "overhead" consistenti dovuti alla preparazione e messa in esecuzione del sistema. Pertanto non è verosimile supporre di attivare il sistema ogni qual volta ci sia bisogno di un pezzo di chiave; viceversa, è più pratico attivare una volta il sistema e produrre una grande quantità di chiavi da conservare a parte e da richiamare poi rapidamente quando serve.

Il problema da gestire a questo punto è di salvare le chiavi in una locazione sicura, protetta contro ogni tentativo di intrusione ed accesso alla chiave ivi contenuta; vanno inoltre garantite l'integrità e la disponibilità della chiave. Per affrontare una situazione di questo tipo in modo tale da giungere ad una soluzione il più possibile *general purpose*, è opportuno porsi nel caso applicativo più generico ed eterogeneo; pertanto, identifichiamo Alice e Bob come due generici host multiutente, non necessariamente dedicati soltanto alla cifratura e allo scambio di chiavi attraverso QKD, su cui è montato il supporto con le chiavi.

Il supporto su cui salvare deve godere di alcune proprietà: tali proprietà verranno elencate di seguito con i vincoli e le conseguenze che comportano:

1. **OFF-LINE**: Il supporto non deve essere né condiviso né accessibile direttamente da

rete in alcun modo poiché questo costituirebbe un rischio inutile e nessun vantaggio per lo scopo prefissato.

Conseguenze:

- Non è direttamente accessibile al di fuori dell'host.
- L'unico modo per accedervi dall'esterno è attraverso il sistema operativo che diviene il filtro fondamentale che non deve essere attaccato con successo, al fine di impedire un accesso indiretto al supporto. Il sistema operativo, infatti, è l'unico che ha accesso diretto al supporto, a questo punto, e che determina se autorizzare o meno un accesso.

2. **SPECIFICO**: Il supporto deve essere dedicato SOLO allo storage delle chiavi e non di altri tipi di dato.

   Conseguenze:

   - Semplicità del protocollo di gestione delle regole di accesso.
   - Si può garantire accesso al supporto solo a quel sottoinsieme di utenti che hanno il diritto ad usare la chiave per cifrare.

     Al contrario, nell'ipotesi di permettere anche storage di altro tipo, si avrebbero da gestire più utenti e con permessi eterogenei, con in aggiunta il problema di partizionare logicamente i dati sul supporto e dover aumentare gli accorgimenti da prendere per garantire la sicurezza e la difesa dagli accessi non autorizzati.

3. **MONOUTENTE**: Al supporto accede solo un utente autorizzato per volta. Gli utenti autorizzati (che possono cifrare usando la chiave) possono essere molteplici ma devono accedere uno per volta ed avere accesso esclusivo al supporto.

   Conseguenze:

- Non occorre gestire fasi di COMMIT.
- Non permettendo più di un accesso alla volta, si riesce a garantire che nel periodo in cui un utente è connesso, ogni altro tentativo di collegamento venga automaticamente abortito dal sistema.

2.1.1 Attacchi possibili e approcci difensivi

È opportuno conservare il supporto con le chiavi, quando possibile ed in periodi di lungo inutilizzo, lontano e disconnesso dalla macchina. Nel momento in cui il supporto è disconnesso, se ne viene garantita la sicurezza fisica, automaticamente risulta essere garantita la sicurezza incondizionata delle chiavi. Nei momenti di attività il supporto deve necessariamente essere connesso alla macchina ed è in questo momento che diventa vulnerabile ad attacchi. Quando connesso, l'interazione con il supporto è gestita dal sistema operativo che pertanto può essere attaccabile con attacchi diretti e volti ad ottenere i permessi di accesso incondizionato al disco (ad es. buffer overflow, ecc). Queste tipologie di attacco puntano a carpire la chiave ma non sono gli unici tipi di attacco che potrebbero interessare, benché di gran lunga le più diffuse in quanto sono quelle che, se a buon fine, forniscono vantaggi maggiori all'attaccante e praticamente violano tutta la sicurezza garantita dal QCRYPT. Un altro tipo di attacco potrebbe infatti essere volto a distruggere le informazioni contenuto sul supporto, facendo perdere la chiave ad una delle due parti, rendendo contemporaneamente inutile la stessa chiave conservata dall'altra parte (a seconda di quanta chiave si salva, la perdita potrebbe essere consistente). Un simile attacco, tuttavia, non da vantaggi diretti all'attaccante ma crea problemi di natura organizzativa alle due parti che devono iniziare nuovamente una distribuzione di chiavi. Esso acquista consistenza solo se è sistematico e va spesso a buon fine, poiché porta a perdite di tempo e dati considerevoli.

Proposte di approcci difensivi

Per gli attacchi volti a carpire la chiave violando il sistema operativo, non ha molto senso preoccuparsi di difendere il sistema operativo, in quanto tali difese dipendono da molti fattori quali il tipo di sistema operativo, la versione, ecc. Pertanto, non è possibile studiare un meccanismo di difesa ad hoc per ogni sistema operativo esistente, senza contare che l'evolversi degli attuali e la nascita di nuovi lascerebbe il problema comunque aperto. Alternativamente, ha più senso mettersi nell'ipotesi che il sistema operativo possa essere violato (si seguirà questo approccio) ed in quel caso determinare il modo in cui difendere comunque il contenuto del supporto, in modo tale che i non autorizzati non riescano ad utilizzare in alcun modo le informazioni carpite o non riescano a ricuperarle.
Nel mondo della sicurezza esistono già molte tecniche volte alla difesa di segreti, che si basano soprattutto su due approcci, ognuno dei quali ha una diversa idea su come fare in modo che dati sensibili non vengano acceduti da terzi non autorizzati:

- **Steganografia**. L'idea base della steganografia è di "nascondere" informazioni sensibili in mezzo ad informazioni irrilevanti e non sensibili in modo tale che la presenza dei dati sensibili passi inosservato. Il numero di informazioni non sensibili deve essere superiore di qualche ordine di grandezza rispetto a quelle sensibili da nascondere. Il vantaggio della steganografia è che per un attaccante risulta molto difficile poter distinguere i bit di chiave dagli altri. Lo svantaggio è che per essere efficace richiede moltissimo spazio, anche 1000 volte la lunghezza della chiave. Se la chiave salvata è molta (come auspicabile) le richieste di spazio sono troppo ingenti ed ingiustificate.

- **Crittografia**. L'idea della crittografia, come detto in precedenza, è di nascondere le informazioni in modo tale che non siano riconoscibili. In un approccio di questo tipo il vantaggio è che l'attaccante ha a che fare con informazioni cifrate e non accessibili direttamente: di conseguenza deve trovare un modo per invertire il processo stesso. Lo

svantaggio sta nel fatto che cifrare un gran numero di dati, come quelli verosimilmente presenti nello storage, richiede molto tempo. Inoltre nel momento in cui un dato sullo storage fosse richiesto, occorrerebe tenere in considerazione un periodo di tempo tra la richiesta dell'informazione e l'ottenimento della stessa, dovuta al processo di decifratura, che può richiedere tempi consistenti.

In linea di principio, nel nostro caso entrambi gli approcci sono da scartare: non ha molto senso proteggere delle chiavi con altre chiavi o con la steganografia in quanto non servirebbe a garantire una sicurezza incondizionata, poiché entrambi i metodi di per sé non sono incondizionatamente sicuri. Tuttavia un uso di queste tecniche può essere usato in un sistema QCRYPT che abbia richieste di sicurezza sì elevate, ma non tali da richiedere un livello del 100%.

Proposte per la sicurezza incondizionata

Sulla base delle considerazioni fatte finora occorre trovare dei meccanismi difensivi che permettano di difendere il dispositivo di storage, sotto le ipotesi fatte finora. Prima di iniziare a definire tali meccanismi, è opportuno chiarire quali siano i passi dell'algoritmo che un utente richiedente una parte di chiave debba eseguire per accedere allo storage:

1. **IDENTIFICAZIONE/AUTORIZZAZIONE** : L'utente si identifica e riceve i permessi per accedere allo storage.
2. **ACCESSO AL PRIMO CLUSTER O SETTORE AVENTE CHIAVE USABILE**: l'utente accede allo storage, iniziando dal primo bit di chiave disponibile estrae gli n bit che gli occorrono.
3. **UTILIZZO DELLA CHIAVE ESTRATTA**: L'utente utilizza i bit estratti per cifrare ed inviare.

4. **ELIMINAZIONE DELLA CHIAVE DALLO STORAGE**: l'utente tramite opportune procedure o primitive elimina (può essere fatto in automatico) i bit della chiave usata, senza lasciare tracce della chiave eliminata.

5. **LOG-OFF**. L'utente si disconnette dallo storage.

Se si garantisce la sicurezza fisica dell'host su cui si trova lo storage (nessun utente malizioso sulla macchina ed il sistema originale è sicuro), è chiaro che il problema degli attacchi proviene necessariamente dalla rete attraverso la quale vengono spediti i messaggi cifrati con la chiave nel dispositivo di storage e da cui provengono i messaggi cifrati da decifrare.

2.2 Sistemi per la sicurezza incondizionata dello storage

2.2.1 Una prima idea: la disconnessione dell'host

La prima osservazione che si può fare è che eventuali attacchi diretti allo storage dalla rete non potrebbero essere perpetrati se nel periodo di connessione dello storage all'host questo sia disconnesso dalla rete pubblica, e lo rimanga durante tutto il periodo di connessione del dispositivo di storage.

Poco prima di connettere fisicamente lo storage (e fino a dopo la disconnessione dello stesso, che si potrebbe pensare avvenga subito dopo che l'utente ha eseguito la fase 3 del protocollo), l'host viene mantenuto isolato dalle attività di rete, attraverso la sospensione, ad esempio, di tutto il traffico sulle porte ethernet. Si noti che l'applicabilità di soluzioni come questa, che hanno grandissimo overhead per le operazioni di preparazione (link, unlink, sospensione del traffico), può essere giustificata solo sotto l'ipotesi, propria di questo contesto, che le operazioni di accesso da parte degli utenti siano relativamente poche. È chiaro che in un contesto classico come l'accesso allo storage di dati di un database condiviso, dove gli accessi possono essere in grande numero, ripetuti in breve tempo da parte dello

stesso utente (alta frequenza) e soprattutto concorrenti, tale approccio non è giustificabile. Nel nostro caso invece può in qualche caso essere preso in considerazione in quanto ci si pone nell'ipotesi verosimile che un utente non debba accedere più volte in periodi brevi (se deve cifrare un certo numero di messaggi, si suppone prenda una volta per tutte la chiave per tutti i messaggi e non acceda ogni volta per ogni messaggio diverso). Inoltre l'accesso concorrente non è considerato e neppure ne sarebbe giustificato un utilizzo: questo infatti sarebbe veramente molto complesso da gestire a causa della fase di eliminazione della chiave.

I vantaggi di un'idea di partenza di questo tipo sono:

- **Semplicità**. Questo metodo risulta essere molto semplice da applicare e l'algoritmo non richiede elevata complessità computazionale.
- **Garanzia di sicurezza elevata**. La disconnessione garantisce che per il periodo di connessione dello storage all'host, non sia possibile che questo subisca attacchi, e quindi garantisce che tutto il protocollo di acquisizione della chiave possa essere fatto solo dall'utente logato e che nessuna fonte esterna possa tentare attacchi in quel periodo.

Come i vantaggi, anche gli svantaggi di una scelta di questo tipo sono evidenti:

- **Scarsa applicabilità**. Seppure abbiamo detto che un approccio di questo tipo possa essere giustificato, è pur vero che ipotizzando una realtà con host con traffico e storage di diverso tipo, la sospensione di tutto il traffico di rete porterebbe all'interruzione improvvisa anche di tutte le altre connessioni e sessioni attive sull'host in molti casi con perdite sostanziali (e.g. si supponga di interrompere un download non resumable, tutta la parte di download fatta fino a quel momento andrebbe perduta e tutto dovrebbe essere rifatto nuovamente).
- **Overhead molto consistente**. Le fasi di disconnessione e riconnessione hanno comunque un costo consistente in termini temporali che comportano periodi significativi

di inattività. Questo fatto limita l'applicabilità in situazioni con accessi frequenti, nei quali questo aspetto determina di fatto l'inadeguatezza del metodo; tuttavia in una realtà come la nostra può essere utilizzato, previo il mantenere la frequenza degli accessi sotto un certo limite.

- **Problema di attacchi preventivi**. Sebbene la disconnessione riesca a garantire che attacchi esterni non possano essere perpetrati dalla rete nel momento in cui lo storage è connesso, allo stesso modo non si può garantire che lo storage non sia passibile di attacchi avvenuti in precedenza. Ad esempio l'installazione di un programma malizioso, residente in memoria prima della disconnessione, potrebbe accedere allo storage durante la disconnessione, carpire informazioni e inviarle all'attaccante al momento della riconnessione dell'host alla rete.

Quanto evidenziato nella fase precedente sottolinea che il metodo in questione risulta maggiormente utile in scenari dove l'host è dedicato, cioè esente da altre attività oltre quella crittografica, e le operazioni sono lunghe e poco frequenti in numero, ovvero con poche connessioni allo storage con prelevamenti di chiave più grandi (e.g. per più messaggi) anziché accessi piccoli e più frequenti.

Il metodo inoltre non è sufficiente a garantire la sicurezza per il problema degli attacchi preventivi: di conseguenza non può essere, di per sé la soluzione unica al prolema ma deve venir utilizzato in combinazione con altre tecniche, per poter garantire un'elevatissima percentuale di sicurezza.

Occorre quindi determinare un protocollo che permetta di sfruttare appieno questa intuizione al fine di garantire una sicurezza incondizionata.

2.3 ATHOS (Asymmetric Two-HOst Storage) Defensive System

L'idea vista in precedenza deve poter essere ottimizzata per evitare attacchi preventivi. Una idea per realizzare un sistema a sicurezza elevata potrebbe essere l'utilizzo di due host opportunamente connessi. L'idea è che uno dei due host sia connesso direttamente alla rete esterna (Internet) mentre un secondo host, su cui è destinato ad essere montato il dispositivo di storage, sia isolato dalla rete tranne che per brevi periodi, all'interno dei quali però né lo storage sia accessibile né ci siano possibilità di installare un programma malizioso all'interno di questo secondo host. Si prevede di utilizzare i due host in modo tale da non permettere traffico in ingresso nel secondo host proveniente dal primo, escluso quello che ci si aspetta di ricevere.

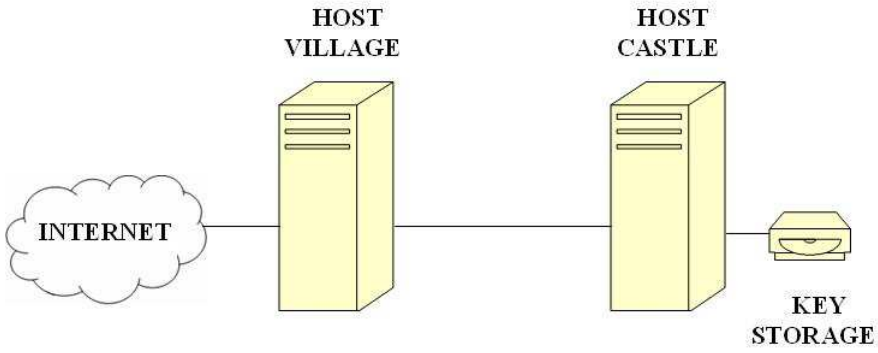


Figura 2.1: Architettura di ATHOS

2.3.1 Caratteristiche delle singole parti

- **HOST VILLAGE**. Host generico multiutente e multifunzione, con traffico di tipo eterogeneo, collegato alla rete pubblica.
- **HOST CASTLE**. Macchina con un hardware essenziale,deve disporre solo di:
  - un dispositivo di output video.

- dispositivi standard di input (tastiera, mouse) per l'utente.
- un dispositivo di input e output dati (lettore floppy, cd-rw, dvd-rw, porta usb per penna o hd esterno) NON ESEGUIBILE per permettere di caricare all'utente il messaggio da cifrare e scrivere il messaggio decifrato.
- un dispositivo di I/O ove montare il dispositivo di storage con le chiavi.
- una memoria primaria che funga da buffer su cui eseguire tutte le operazioni e salvare i dati in I/O.
- Una memoria ROM con un sistema operativo (MOSES) minimale ed altre primitive esenziali.

2.3.2 Il sistema operativo MOSES (Minimal Operating System Enhancing Security)

Il sistema operativo risiedente su ROM ha il compito di sovrintendere tutte le operazioni sull'host CASTLE e di realizzare i diversi passi del protocollo, garantendo che questi vengano eseguiti in modo corretto. Il sistema operativo, che chiamiamo MOSES, deve rispettare alcuni vincoli e regole. In primo luogo il sistema deve essere minimale, cioé deve essere tale da eseguire tutte e sole le operazioni richieste nel minor tempo possibile: altre operazioni non previste non devono essere realizzabili, sia perché questa possibilità diminuirebbe la sicurezza (maggiore dimensione del sistema operativo, maggiore presenza di bachi più difficilmente riscontrabili), sia perché un sistema operativo che realizzi altre funzionalità oltre quelle strettamente richieste per il protocollo ATHOS avrebbe certamente una complessità più elevata, che di fatto si tradurrebbe in un aumento significativo del tempo medio per realizzare ogni singola operazione. Oltre che minimale nella sua struttura interna e nel kernel, MOSES deve essere tale che le sue fasi di boot siano il più rapide possibili, poiché di fatto non viene richiesto che realizzi operazioni sofisticate, al contrario che sia il più possibile efficiente. In pratica (si vedrà meglio in seguito) l'efficienza dell'intero sistema

è direttamente determinata dalla rapidità nella fase di boot dell'host, che dipende a sua volta dalla rapidità di esecuzione delle procedure di avvio del sistema operativo. Il sistema operativo risulta essere dedicato per le operazioni previste da ATHOS e non funziona di fatto come un sistema operativo standard all-purpose. Poiché sull'host CASTLE l'unico sistema operativo ammesso dal protocollo ATHOS è MOSES, è chiaro che l'host può essere utilizzato solo per operazioni di cifratura, decifratura, invio e ricezione di crittogrammi e non per altre operazioni tipiche e più generali di un normale sistema operativo.

Caratteristiche operazionali di MOSES

Il sistema operativo deve realizzare particolari operazioni specifiche e nessun'altra per i vincoli di sicurezza presentati in precedenza:

- **Primitive di cifratura/decifratura OTP**. In primis MOSES deve contenere primitive atte all'esecuzione delle operazioni di cifratura e decifratura dell'One Time Pad attraverso la chiave. Tali primitive devono poter accedere al dispositivo di storage ed al messaggio in chiaro o cifrato (che si suppone si trovi in memoria RAM), realizzare l'operazione di cifratura/decifratura, cancellare i dati intermedi e restituire il messaggio in chiaro o quello cifrato in modo che possa essere acceduto opportunamente.

- **Primitive di attivazione e disattivazione della porta di rete sull'host CASTLE**. MOSES deve contenere delle primitive robuste che permettano di disconnettere l'host CASTLE dall'host VILLAGE in qualunque momento ed in maniera perentoria, ovvero non devono essere soggette a ritardi o malfunzionamenti. La fase di disconnessione della porta è, in particolare, una regione critica dell'intero sistema; pertanto è necessario garantire che l'operazione venga eseguita nel momento desiderato e non sia sensibile a fallimento.

- **Primitive per mount e dismount logico dell'unità di storage e di quella di input del messaggio**. Occorrono primitive che effettuino operazioni di montaggio e

smontaggio logico del dispositivo di storage e dell'unità di input/output su cui salvare il messaggio. In particolare è richiesto che tali primitive siano, allo stesso modo di quelle che gestiscono la connessione con l'host VILLAGE, robuste e che realizzino il loro scopo senza fallire o, in caso di failure, questo venga rilevato e venga ripetuta la procedura fino al corretto conseguimento.

- **Primitive per lettura ed eliminazione della chiave dal supporto di storage**. Occorrono primitive che accedano direttamente allo storage e leggano la chiave per una lunghezza pari a quella necessaria ed occorrono primitive che una volta che la cifratura/decifratura sia andata a buon fine la eliminino permanentemente dallo storage. A proposito dell'eliminazione della chiave, il problema che si pone è di capire "quando" sia più conveniente procedere all'eliminazione della chiave. Le alternative a questo punto possono essere due:

  1. Una prima idea (già accennata) consiste nell'eliminare la chiave solo nel momento in cui la fase di cifratura/decifratura sia andata a buon fine. Questo richiede la conoscenza dello stato della cifratura/decifratura: vi è quindi una "variabile" in più da gestire da parte delle primitive atte a realizzare queste operazioni. A fronte di un errore di elaborazione, il procedimento può quindi venir ripetuto sul messaggio utilizzando la medesima chiave, con un conseguente risparmio di bit di chiave (che quindi non vengono sprecati).
  2. Un'altra alternativa è di procedere all'eliminazione della chiave nel momento stesso in cui questa viene letta per essere utilizzata; tale scelta comporta l'utilizzo di una nuova porzione di chiave a fronte di un errore nella cifratura ma permette di non dover tenere traccia di uno stato dell'operazione eseguita. È chiaro che questa seconda ipotesi è attuabile solo in fase di cifratura: in fase di decifratura occorre utilizzare ad ogni costo la chiave corrispondente a quella scelta dalla parte

che ha cifrato il messaggio; pertanto, nel caso in cui la parte debba decifrare un messaggio ricevuto, l'unico approccio possibile è il precedente. Inoltre, in questo approccio, oltre allo spreco della chiave, occorre realizzare una procedura per informare l'altra parte della parte di chiave sprecata, in modo tale che la stessa provveda ad eliminarla dal suo storage e ad utilizzare la parte di chiave corretta.

La scelta migliore può essere determinata solo in fase di implementazione e dipende dalle scelte implementative e dai test di efficienza sui due approcci, possibili solo dopo una realizzazione pratica dell'intero sistema.

- **Protocolli di comunicazione per l'interazione con l'host VILLAGE**. Il sistema operativo deve prevedere una serie di primitive atte alla comunicazione con l'host VILLAGE. Più in dettaglio, le procedure necessarie sono quelle per inviare al VILLAGE un messaggio cifrato, per ricevere dallo stesso un messaggio, sempre cifrato, ed eventuali altre operazioni di gestione che si potrebbero rendere necessarie in fase di implementazione (e.g. la procedura per indicare un cambio di chiave, richiesta nel secondo approccio evidenziato al punto precedente).

Modalità di avvio di MOSES

Il sistema operativo deve dare la possibilità di bootare in due modi diversi.

- **Modalità TRANSMITTER**. Boot che si conclude mantenendo chiuse le porte di rete. Al momento del riavvio le porte sono chiuse e restano chiuse durante tutta la fase di boot ed anche dopo la conclusione della stessa, fino ad una richiesta di bloccaggio da parte del sistema operativo. Un avvio in questa modalità non permette il traffico né in entrata né in uscita dall'host CASTLE ma garantisce l'impossibilità di attacchi dall'esterno verso l'host stesso. Viene utilizzata per "isolare" l'host in prossimità di fasi cruciali dal punto di vista della sicurezza.

- **Modalità RECEIVER** Boot che alla fine apre le porte di rete. In questa modalità al riavvio le porte sono comunque chiuse e restano tali fino alla conclusione della fase di boot. A questo punto le porte vengono aperte e l'host CASTLE è collegato alla rete esterna attraverso l'host VILLAGE e passibile, in questa configurazione, di un attacco esterno. Questa configurazione è tuttavia necessaria nel momento in cui occorra interagire con l'esterno per inviare o ricevere messaggi cifrati.

Assunzioni su MOSES e ATHOS

Dopo la specifica delle caratteristiche principali e dei vincoli che MOSES deve rispettare ma prima di entrare nei dettagli del protocollo specifico di ATHOS, occorre indicare alcune assunzioni che si danno per scontate nelle pagine a seguire e che devono essere sempre garantite nel momento in cui una futura implementazione del sistema possa essere realizzata. Qualora le assunzioni seguenti non fossero verificate, l'implementazione del sistema non potrebbe dirsi incondizionatamente sicura.

- Si assume che il contenuto della ROM non sia malizioso.
- Nel momento in cui si chiudono le porte ogni protocollo di comunicazione attivo con l'host VILLAGE viene terminato.
- L'accesso al dispositivo di storage deve essere consentito solo mediante le apposite primitive nella ROM: ogni tentativo di accesso allo storage attraverso altre funzioni di lettura/scrittura deve essere bloccato dal kernel del sistema operativo nella ROM, fornendo di fatto una garanzia di sicurezza supplementare (anche se non strettamente necessaria).
- Il protocollo assume inoltre che l'identità degli utenti che interagiscono con l'host CASTLE per cifrare/decifrare sia verificata in qualche modo e che chi interagisce con l'host CASTLE sia a tutti gli effetti abilitato a farlo. Una proposta per la fase

di autenticazione dell'utente che garantisca un'elevata sicurezza la si può vedere in seguito ma è una fase separata e precedente all'utilizzo di ATHOS.

2.3.3 I passi del protocollo

I passi del protocollo sono differenti tra la fase di invio di un messaggio cifrato e la ricezione di un crittogramma per la decifratura. Di conseguenza, trattiamo separatamente i due casi.

2.3.4 La fase di invio

I passi del protocollo che devono essere seguiti per poter permettere il prelevamento e l'uso sicuro della chiave dallo storage da parte del mittente (sia Alice), per cifrare un messaggio in chiaro ed inviarlo sulla rete, sono i seguenti (fig. a pag. 66):

1. **Log-on** dell'utente che deve cifrare un messaggio (possono essere anche più di uno, ma per semplicità di astrazione vediamo l'insieme di tutti i messaggi che Alice può voler inviare in un'unica sequenza come un solo "grande" messaggio) sull'host CASTLE.

2. **Riavvio dell'host CASTLE in modalità TRANSMITTER**. Non ci devono essere dischi o unità scrivibili montate sull'host CASTLE. Il reset quindi formatta la memoria ed elimina eventuali programmi maliziosi che possono essere presenti dopo aver passato le difese precedenti. All'atto del riavvio, l'host CASTLE viene disconnesso dall'host VILLAGE, le porte di rete sono tutte bloccate di default, attraverso le primitive in ROM. Il fatto che si scelga di utilizzare una primitiva nella ROM per bloccare e sbloccare le porte di rete , dà garanzia che il codice di tale primitiva non sia malizioso ne sia stato modificato mediante un attacco mirato, quindi che la chiamata di tale primitiva porti "effettivamente" alla chiusura/apertura delle porte di rete.

3. **Connessione/caricamento del dispositivo contenente il messaggio da cifrare**. Il messaggio da cifrare viene inserito da tastiera oppure viene montato il dispositivo su cui il messaggio si trova.

4. **Collegamento dello storage sull'Host CASTLE**. Appena il sistema operativo è stato riavviato, si procede al collegamento fisico dell'unita di storage all'host CASTLE.

5. **Calcolo del messaggio cifrato** (fasi 2-4 del protocollo utente a pag. 53). L'utente chiama la procedura in ROM per il calcolo del messaggio cifrato passandogli come parametri il puntatore alla prima locazione del dispositivo di storage (il dettaglio su quale debba essere la migliore soluzione a riguardo va analizzata a livello implementativo, si deve fare in modo che l'utente non debba passare, né conoscere esplicitamente il puntatore al primo cluster di chiave utilizzabile sullo storage) dove si trova la chiave da usare ed il puntatore al messaggio nel dispositivo di input. A questo punto la procedura calcola e copia in memoria il messaggio cifrato ed elimina la chiave usata dallo storage man mano che la usa. Infine elimina dalla memoria eventuali residui di dati del calcolo del messaggio cifrato (e.g. eventuali parti temporanee di messaggio in chiaro o chiave).

6. **Smontamento del dispositivo di storage**. Il dispositivo di storage viene smontato. Si noti che il dispositivo non può aver subito tentativi di intrusione se le condizioni e le ipotesi poste in precedenza sono rispettate e se tutti i passi precedenti del protocollo sono state eseguite correttamente.

7. **Smontamento del supporto di input**. Viene smontato il dispositivo col messaggio in chiaro. Allo stesso modo non può aver subito alterazioni neppure questo dispositivo (che comunque viene di default attivato in modalità Read-only, poiché non ci sono esigenze di scrittura).

8. **Collegamento dell'host CASTLE all'host VILLAGE**. L'host CASTLE viene collegato all'host VILLAGE, attraverso una opportuna primitiva nella ROM. Tramite l'host VILLAGE, con metodi tradizionali l'utente (Alice) può inviare un messaggio al destinatario (Bob). Si noti che a questo punto ogni tentativo di attacco alla chiave è

impossibile da parte di intrusi sull'host VILLAGE: la memoria infatti è stata "ripulita" dalla chiave e contiene solo il messaggio cifrato; il dispositivo di storage e quello di input col messaggio in chiaro sono stati disconnessi.

9. **Invio del messaggio**. Il messaggio cifrato viene inviato all'host VILLAGE e quindi, su canali pubblici, a Bob.

10. **Log-off**. L'utente si disconnette dall'host CASTLE. Le porte di rete rimangono aperte.

2.3.5 La fase di ricezione

I passi da seguire in fase di ricezione, nel momento in cui il messaggio cifrato arriva sull'host VILLAGE di Bob, invece sono (fig. a pag. 68):

1. **Segnalazione all'host CASTLE dell'arrivo di un messaggio**. L'host VILLAGE invia un messaggio all'host CASTLE indicante l'arrivo di un nuovo messaggio cifrato dalla rete.

2. **Invio del messaggio cifrato al'Host CASTLE**. L'host CASTLE risponde con un acknowledgement quando è pronto a ricevere il messaggio. Questo viene inviato all'host CASTLE ed eliminato dal buffer dell'host VILLAGE dopo un esito positivo dell'invio.

3. **Chiusura delle porte di rete sull'host CASTLE**. Una volta ricevuto il messaggio l'host CASTLE chiude tutte le porte tramite l'apposita primitiva definita nella ROM.

4. **Collegamento del dispositivo di storage**. Viene collegato il dispositivo di storage.

5. **Collegamento del dispositivo di output**. Viene collegato alla porta I/O il dispositivo su cui salvare il messaggio cifrato.

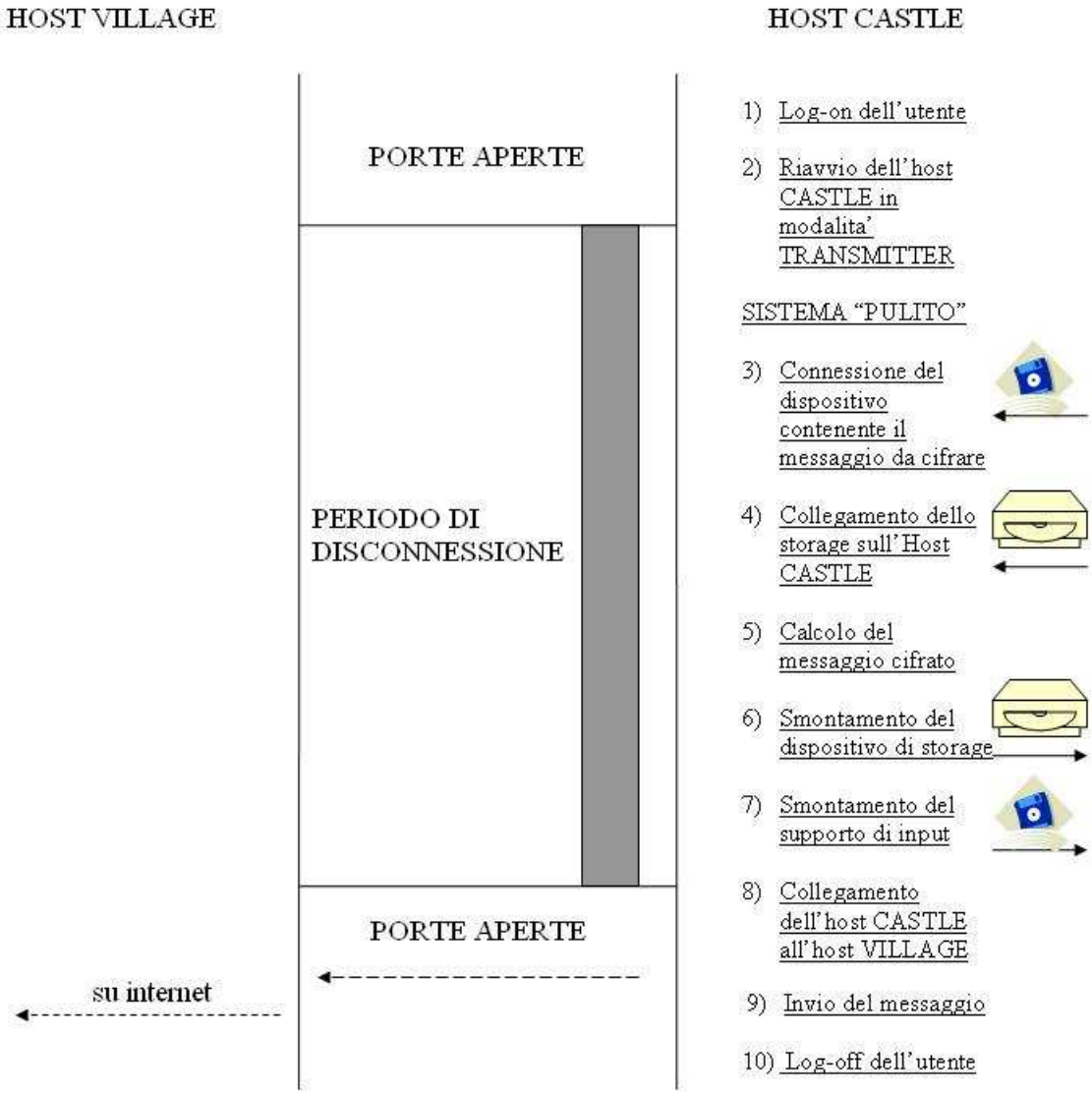


Figura 2.2: Fase di invio

6. **Decifratura e salvataggio del messaggio**. Il messaggio è decifrato leggendo il messaggio in memoria e la chiave nel dispositivo di storage e salvato sul dispositivo apposito. È chiaro che anche in questa fase di decifratura, la primitiva di decifratura provvede all'eliminazione della chiave dal supporto quando la usa.

7. **Eliminazione della chiave usata dal supporto di storage e smontamento**. La chiave letta viene eliminata ed il dispositivo di storage smontato (fasi 2-4 del protocollo utente).

8. **Smontamento del dispositivo di output**. Il dispositivo su cui è stato scritto il messaggio viene smontato.

9. **Riavvio del sistema sull'host CASTLE**. L'utente termina la sessione riavviando il sistema in modalità TRANSMITTER se vuole rispondere al messaggio ricevuto o comunque inviare messaggi (e segue le fasi del protocollo di INVIO) oppure in modalità RECEIVER (porte aperte) se, non dovendo inviare, vuole mettersi in modalità di ricezione nel qual caso ripete questi passi la momento di una nuova ricezione.

2.3.6 Possibili attacchi allo storage

La fase di ricezione inizia con le porte dell'host CASTLE aperte. Sotto questa ipotesi possiamo immaginare la presenza di potenziali attacchi nella memoria volatile (sotto forma di programma e codice malizioso, ecc): il bloccaggio delle porte e il successivo riavvio del sistema garantiscono che nessuna informazione sul messaggio decifrato possa essere sniffata e portata fuori dall'host CASTLE. Il programma malizioso tuttavia rimane in memoria per tutta la fase di decifratura e quindi per un periodo di tempo durante il quale lo storage viene montato sull'host CASTLE: questo potrebbe permettere al programma malizioso di scrivere sullo storage, pertanto è bene valutare i tipi di scritture possibili ed i loro effetti sulla sicurezza e sulla disponibilità della chiave:

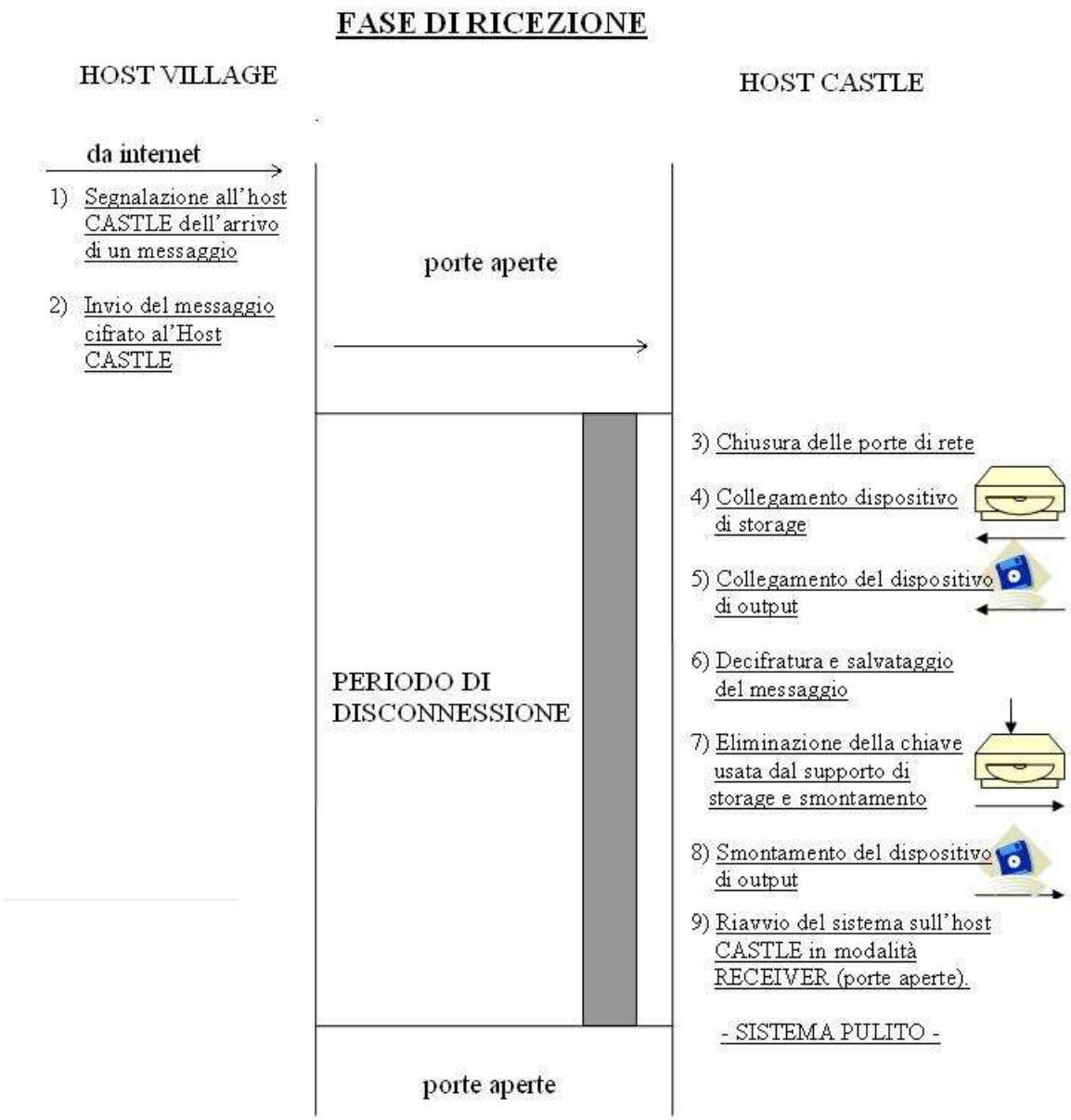


Figura 2.3: Fase di ricezione

1. **SCRITTURA DISTRUTTIVA**. Vista l'impossibilità di inviare dati fuori dall'host CASTLE, un attaccante potrebbe cercare di eliminare la chiave sullo storage creando problemi alla parte in causa e di fatto costringendola ad iniziare una nuova sessione di generazione e scambio di chiave con la controparte corrispondente. Un attacco di questo tipo è tuttavia poco auspicabile in quanto non darebbe alcun vantaggio ad un attaccante (il cui fine, si ricordi, è di carpire informazioni) ma si limiterebbe a creare ritardi nello scambio dei messaggi tra le parti.

2. **COPIATURA DI CODICE MALIZIOSO**. Un programma malizioso potrebbe cercare di copiare del codice che potrebbe essere eseguito a partire dallo storage. Questo attacco sarebbe potenzialmente pericoloso perché permetterebbe di inviare ad insaputa delle due parti dati sensibili come chiave o messaggio ad un attaccante. Lo schema di un possibile attacco di questo tipo, che sfrutta la fase di ricezione e la successiva fase di invio, viene mostrato in fig. a pag. 70:

Un simile attacco, sebbene difficile, va tenuto in considerazione. Le contromisure tuttavia possono essere efficaci e relativamente semplici: in primo luogo è opportuno disabilitare la modalità di esecuzione dallo storage, cioé fare in modo che le procedure del sistema operativo in ROM che leggono e scrivono (cancellazione della chiave) non prevedano la modalità di esecuzione. Più in generale , poiché le attività dell'host CASTLE sono limitate in numero e profondamente standardizzate, si possono definire un numero limitato di processi, generati tutti dal kernel del sistema operativo su opportuna richiesta, e l'esecuzione di ogni altro processo. Questo risolverebbe il problema alla radice in quanto non permetterebbe di fatto l'acquisizione dei privilegi di esecuzione a nessun codice malizioso residente in memoria. Eventuali problemi legati all'applicazione di politiche di questo tipo, o la necessità di dover permettere a più processi di avere privilegi di esecuzione, potrebbero venire fuori solo in una fase di implementazione del protocollo (ad esempio troppo poca flessibilità o

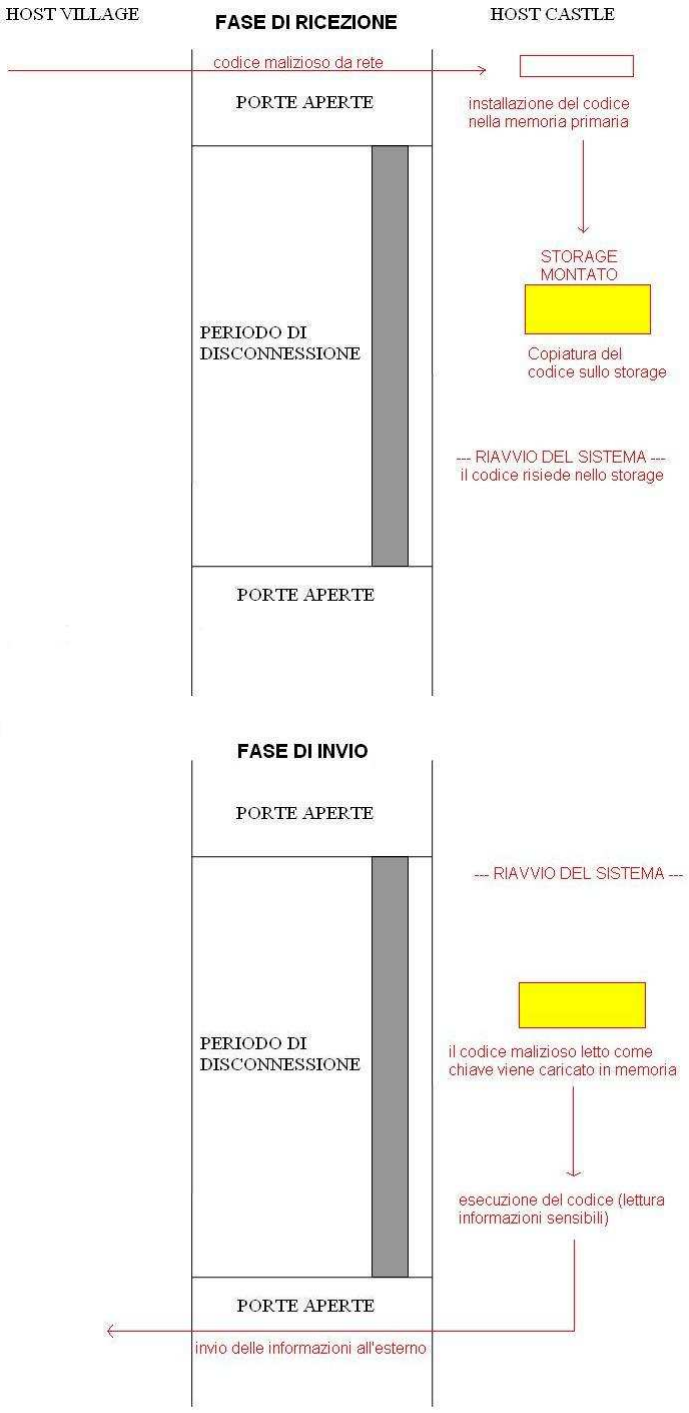


Figura 2.4: Possibile attacco

complessità del codice sotto queste ipotesi): eventuali correzioni o politiche alternative, che però debbono garantire la stessa sicurezza e rendere inoffensivi attacchi del tipo analizzato, andranno prese in considerazione in quella fase dello sviluppo.

2.3.7 Considerazioni finali su ATHOS

Alla luce di quanto mostrato finora, occorre fare qualche osservazione e dettagliare maggiormente alcuni punti di forza e di debolezza del protocollo, oltre che a definire eventuali altri vincoli che in fase di implementazione del sistema devono essere rispettati.

In primis, si noti che il messaggio cifrato in fase di cifratura ed il messaggio decifrato in fase di decifratura risiedono in memoria principale: questa pertanto deve essere abbastanza grande da garantire che un messaggio possa essere mantenuto in memoria per intero. La determinazione della dimensione ottimale della memoria primaria può essere ricavata da studi statistici sulla grandezza media del messaggio che gli utenti del sistema possono avere da gestire. Occorre sempre tenere ben presente, però, che il sistema è usato per cifrare messaggi, quindi tratta dati di grandezza limitata, e che il costo della memoria primaria che si può rendere necessaria è comunque molto contenuto.

Inoltre, l'host VILLAGE permette ad utenti generici di eseguire le loro operazioni normalmente e a tutti gli effetti senza i problemi di riavvio evidenziati nell'approccio precedente (disconnessione dell'host dalla rete).

Il dispositivo di I/O su cui si scrive il messaggio in chiaro deve essere sempre in modalità NON ESEGUIBILE poiché potrebbe verificarsi l'eventualità che del codice malizioso presente in memoria primaria al momento della chiusura delle porte in fase di ricezione possa trascriversi nel dispositivo stesso.

Il protocollo, se correttamente implementato e se le ipotesi ed assunzioni su cui si basa sono rispettate, garantisce che un attacco diretto all'area dati contentente la chiave non sia possibile. Si noti inoltre che l'utilizzo di un supporto di input da cui leggere il messaggio

in chiaro e di un supporto di output su cui salvare il messaggio (che banalmente potrebbe essere un identico supporto R/W generico) non sono strettamente necessari; nel caso in cui le informazioni scambiate siano solo messaggi temporanei (lettura e scarto) che non richiedono un salvataggio per un uso successivo, questi dispositivi possono essere rimossi dall'architettura del sistema e le fasi del protocollo corrispondenti al loro mount e dismount possono essere eliminate senza problemi. L'immissione del messaggio avverrà in questo caso tramite tastiera mentre l'output del messaggio cifrato sarà dato in output a video sulla macchina ricevente. Un utilizzo del sistema di questo tipo è molto più sicuro perché il salvataggio delle informazioni scambiate implica la possibilità che le informazioni -in chiaro- sul dispositivo di salvataggio possano essere carpite in qualche modo ed apre il problema di garantire pari sicurezza (fisica) incondizionata a questi altri dispositivi di storage.

Il problema principale di un simile protocollo è nell'alto overhead dovuto ai riavvii, che seppur il più possibile minimali, implicano tempi di attesa notevoli e di totale inattività, in quanto nessuna operazione di cifratura o decifratura è possibile durante il reboot.

Un altro problema è legato al fatto che sia possibile eseguire solo una operazione per volta o di ricezione o di invio, pagando tra due operazioni successive comunque almeno una volta il tempo del reboot.

2.4 PORTOS (Pair-Organized Recovering Time On Stall) Defensive System

Il limite più evidente del protocollo ATHOS è il costo in termini di tempo che si paga per avere una sicurezza incondizionata dello storage, causato maggiormente dai "tempi morti" di attesa per il riavvio del sistema, che, seppur sotto l'ipotesi che il sistema operativo debba essere minimale ed il suo riavvio il più rapido possibile, rimane comunque molto elevato, poiché il sistema non può in alcun modo venire utilizzato in questi periodi.

La situazione migliorerebbe di molto se nel momento in cui si esegue un riavvio, ci fosse

comunque la possibilità di eseguire un'altra operazione in parallelo, di fatto non pagando la perdita di tempo dovuta al reboot. Una estensione del protocollo ATHOS, che chiameremo PORTOS (Pair-Organized Recovering Time On Stall) Defensive System, realizza un simile miglioramento a costo di complicare leggermente l'architettura del sistema.

PORTOS prevede di utilizzare due host CASTLE anziché uno solo, di modo che al riavvio di uno dei due, l'altro resti comunque sempre disponibile per operazioni di invio e ricezione. La struttura di PORTOS è schematizzata in fig. a pag. 73.

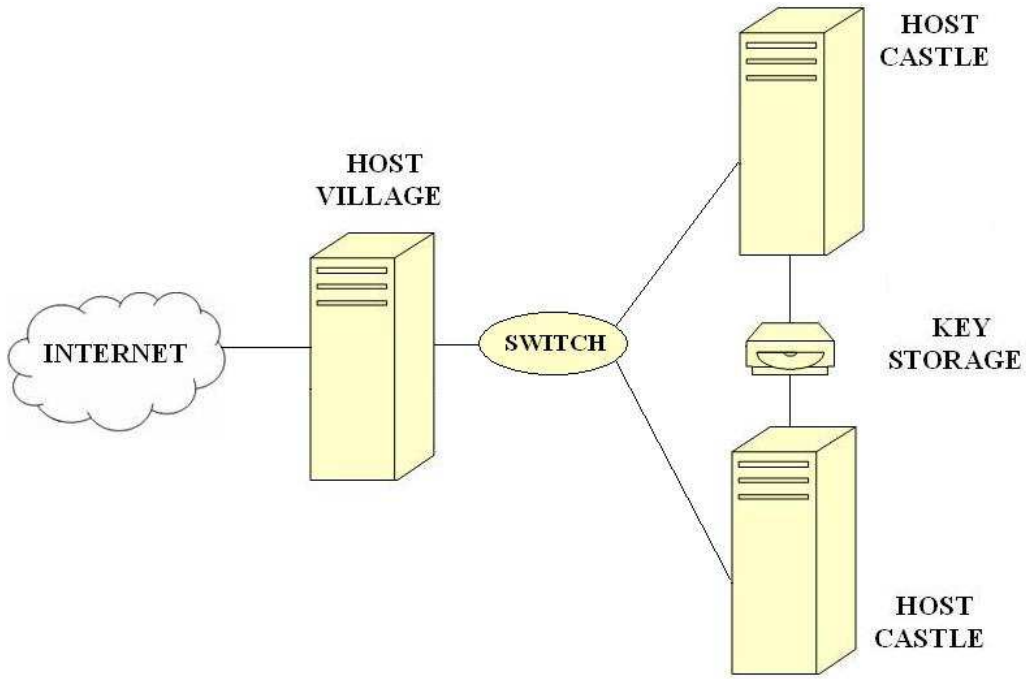


Figura 2.5: Architettura di PORTOS

Nell'architettura schematizzata, gli host CASTLE sono dello stesso tipo di quelli descritti per ATHOS, mentre il dispositivo di storage viene condiviso dai due host CASTLE e montato alternativamente sull'uno e sull'altro, ma mai contemporaneamente su entrambi.

2.4.1 Note sull'utilizzo degli host CASTLE

I due host CASTLE sono pensati per essere utilizzati alternativamente in modo da garantire maggiore disponibilità, ma in un certo istante solo uno dei due può essere usato per inviare/ricevere. L'utilizzo in parallelo delle due macchine presupporrebbe di avere lo storage condiviso contemporaneamente da entrambi gli host. Nonostante possa essere condiviso tra le due parti, l'accesso allo stesso deve però essere esclusivo, quindi non più di un host per volta può comunque accedere al disco, in quanto più operazioni di lettura e eliminazione della chiave non possono essere eseguite contemporaneamente.

Se così fosse, infatti, occorrerebbe suddividere la chiave in parti destinate al primo host e parti destinate al secondo host: la situazione sarebbe molto problematica, verrebbe aumentata la complessità del sistema operativo poiché la gestione di operazioni mutuamente escludenti su un supporto condiviso richiede l'applicazione di algoritmi la cui complessità in certi casi è notevole, portando comunque alla creazione di ritardi che in alcuni casi possono essere anche significativi e che rallenterebbero l'attività a tal punto da non essere giustificati.

Inoltre verrebbe richiesto ad Alice e Bob di interfacciarsi per decidere quali parti utilizzare ed in quali messaggi, anche questo attraverso algoritmi non banali e complessi. Una parallelizzazione delle operazioni non sarebbe neppure realizzabile se si montassero due copie del dispositivo di storage sui due host CASTLE in quanto, nel momento in cui una parte di chiave su un supporto viene usata, l'altro supporto dovrebbe essere aggiornato prontamente, eliminando la chiave usata con tutta una serie di problemi di commit da gestire. In pratica, quindi, la scelta del meccanismo sequenziale in cui si accede allo storage, si legge, si usa la chiave e la si elimina, senza prevedere accessi in alcuna forma concorrenti, resta l'approccio migliore per semplicità degli algoritmi di gestione e per la facilità nel mantenimento della sincronizzazione tra Alice e Bob (Alice e Bob hanno la stessa copia di chiavi, per ogni operazione usano ed eliminano gli stessi bit e quindi la sincronizzazione tra le due chiavi usate è automaticamente mantenuta).

2.4.2 Note sullo SWITCH

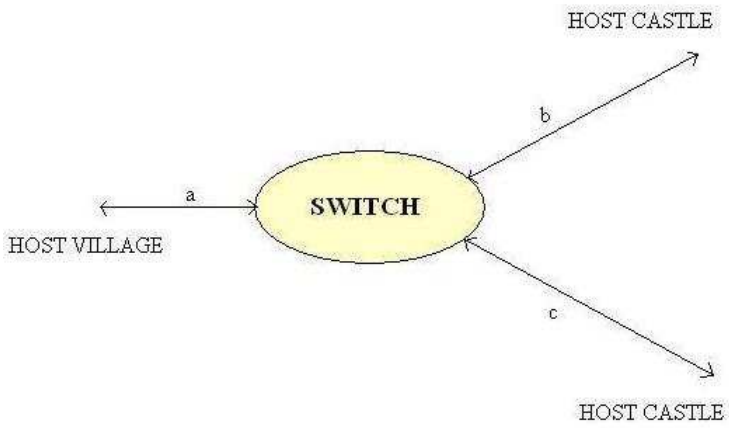


Figura 2.6: Lo switch

L'utilizzo di due host CASTLE da collegare allo stesso host VILLAGE presuppone l'utilizzo di uno SWITCH (fig. a pag. 75). Tale switch non richiede un'architettura particolare anche perché ha solo una funzione di smistamento, secondaria quindi rispetto alle funzioni dell'host VILLAGE e degli host CASTLE; tuttavia è necessario evidenziare quali, tra gli attacchi possibili, siano particolarmente importanti e lesivi della sicurezza del sistema, al fine di determinare politiche di gestione del traffico e accorgimenti da adottare onde evitarli.

In primo luogo occorre notare che il ramo (a) dello switch è il canale che permette di collegare i due host CASTLE all'host VILLAGE, pertanto tale ramo deve essere sempre collegato all'host VILLAGE. Questo sottolinea il fatto che lo switch è collegato indirettamente ad Internet e pertanto passibile di eventuali attacchi a livello Data Link. Ciò nonostante è chiaro che gli attacchi possibili sullo switch, qualunque essi siano (ARP Poisoning, ecc) non permettono all'attaccante di venire a conoscenza di nessun dato sensibile, ne di messaggio ne di chiave poiché di fatto attraverso lo switch ed in tutte le direzioni viaggiano solo dati cifrati. La compromissione di uno switch può invece costare caro in termini di efficienza e disponibilità: un attacco volto a ledere le attività di smistamento dello switch porterebbe

di fatto al mancato funzionamento del sistema, venendo meno la possibilità di comunicare tra host CASTLE e host VILLAGE. Di conseguenza, ogni attacco rivolto allo switch non diminuirebbe in alcun modo il livello di sicurezza del sistema, ne minaccerebbe la segretezza delle informazioni. Ciò nonostante lo switch ha un'importanza strategica perchè la caduta dello switch porta necessariamente al mancato funzionamento del sistema. Per questo motivo, l'utilizzo dello switch tra gli host CASTLE e l'host VILLAGE diminuisce la robustezza del nuovo sistema rispetto ad ATHOS, perché costituisce un nuovo punto di attacco che non era presente nel caso precedente, dove il collegamento tra CASTLE e VILLAGE era diretto. Poiché le funzioni dello switch sono di semplice smistamento si potrebbe pensare di utilizzare un HUB anziché uno switch ma l'utilizzo dello switch permette di implementare politiche di filtraggio opportune che possono consentire una migliore protezione degli host CASTLE da tentativi di introduzione di codice malizioso negli stessi. In pratica l'utilizzo dello switch (che resta comunque necessario visto il bisogno di collegare due host CASTLE allo stesso host VILLAGE) se da un lato rende meno robusto il sistema nella sua totalità poiché aggiunge una nuova componente costantemente connessa (anche se indirettamente) alla rete esterna, dall'altro può invece essere sfruttato per difendere meglio dai tentativi di intrusione, aumentando di fatto il livello di sicurezza specifico degli host CASTLE.

2.4.3 Considerazioni finali su PORTOS

Il primo vantaggio di PORTOS risiede nell'utilizzare un'architettura molto economica, solo leggermente più costosa di ATHOS a causa dell'utilizzo di uno switch e di un host CASTLE in più, il cui costo di realizzazione, come gi à detto per ATHOS, dovrebbe essere comunque abbastanza contenuto.

Inoltre, vengono usati gli stessi protocolli per gestire l'invio e la ricezione utilizzati in ATHOS ed utilizzati alternativamente sui due host CASTLE. Per questo motivo i vantaggi sono gli stessi mentre i difetti sono ridotti poiché l'utilizzo di due host CASTLE, invece

che uno, migliora la disponibilità del sistema, permettendo l'utilizzo del tempo di reboot o almeno di parte di esso, per eseguire altre operazioni di invio e ricezione.

Il livello di sicurezza è ancora aumentato perché l'uso di politiche di filtering sullo switch migliora la capacità di monitorare il traffico da e per gli host CASTLE e rileva eventuali tentativi di attacco.

Nonostante l'uso di due host CASTLE, però, entrambi non possono essere utilizzati in parallelo per i problemi di sincronizzazione sullo storage evidenziati. Di per sé, costituisce uno spreco perché costringe (reboot a parte) a non utilizzare in nessun modo le risorse dell'host non utilizzato.

PORTOS permette all'utilizzatore di avere un'alta probabilità di avere un host CASTLE libero su cui eseguire un'operazione di invio o ricezione. Questo è vero sotto l'ipotesi che i messaggi scambiati siano abbastanza grandi da permettere che una singola operazione abbia una durata media che sia confrontabile con il periodo di tempo richiesto per il reboot.

Detto T_{boot} il periodo di tempo necessario per un riavvio di un host CASTLE, e detto T_{op} il tempo medio di un'operazione di invio/ricezione,, il caso ottimo si ha quando $T_{boot} \leq T_{op}$. In questo caso riusciamo a garantire che ci sia SEMPRE un host libero su cui lavorare, ovvero la piena disponibilità del sistema.

Al contrario, nel caso in cui $T_{boot} > T_{op}$, la minima probabilità di avere un host CASTLE libero in un dato istante è data dal rapporto:

$\frac{T_{op}}{T_{boot}}$

Tale valore percentuale costituisce la probabilità *minima*, cioè sotto l'ipotesi peggiore, in cui l'utilizzo del sistema sia continuato ed *a pieno regime* (cioè in ogni momento c' è richiesta di eseguire operazioni di invio e ricezione). È chiaro che nei momenti di inattività o scarsa attività del sistema, la probabilità di avere host liberi aumenta considerevolmente.

PORTOS, tuttavia, risulta essere meno efficiente nel momento in cui le operazioni di invio/ricezione siano molto brevi e molto frequenti.

In questo caso infatti si ha che $T_{op} << T_{boot}$ e quindi la probabilità di trovare un host CASTLE libero è molto bassa.

In questo caso la disponibilità potrebbe diminuire talmente tanto che il guadagno derivante dall'utilizzo di due host CASTLE anziché uno, come in ATHOS, non sarebbe apprezzabile.

Anche se la disponibilità di PORTOS in ogni momento è certamente maggiore di quella di ATHOS (vi è comunque un host in più), tale valore potrebbe essere così basso e assimilabile a quello di ATHOS, sotto le ipotesi fatte, che un utilizzo di tale sistema rispetto al precedente, con i conseguenti aumenti di costo e complessità, non sarebbe di fatto giustificato. Occorre pertanto determinare un'architettura di sistema che permetta di garantire alti valori di disponibilità in condizioni di operazioni alternate, brevi e frequenti.

2.5 ARAMIS (Advanced Redundant Asymmetric Multipoint Installation) Defensive System

Sebbene il sistema PORTOS si ponga come una eccellente risposta al problema, da una parte, di garantire una sicurezza incondizionata da attacchi da rete, mantenendo, dall'altra parte, un'efficienza elevata per lo scambio dei messaggi tra Alice e Bob, è pur vero che si evidenzia un calo vertiginoso dell'efficienza del sistema a fronte di un utilizzo per lo scambio di messaggi brevi e frequenti. Pertanto, benché nella maggior parte dei casi PORTOS sia adeguato, risulta essere necessario, comunque, studiare un sistema da utilizzare in questi casi in cui PORTOS non funziona.

Il sistema che si vuole andare a definire, quindi, cerca di rispondere efficientemente a quei casi in cui il numero di *switching* tra un invio ed una ricezione sia molto alto nell'unità di tempo, così alto da fare in modo che il tempo di un'operazione sia molto inferiore a quello del riavvio degli host CASTLE.

Lo scopo di ARAMIS è quello di fare in modo che, in qualunque momento e per qualunque durata del tempo medio di invio di un messaggio, vi sia sempre disponibile

un host CASTLE a cui connettersi per eseguire un'operazione di invio o ricezione.

La definizione di ARAMIS parte dalla consapevolezza che ATHOS e PORTOS sono due sistemi efficienti, seppur sotto opportune ipotesi che ne definiscono i limiti e il range di scenari reali in cui possono essere utilizzati: questa considerazione fa capire come ARAMIS utilizzi, per quanto possibile, l'idea di PORTOS, così come, in precedenza, PORTOS aveva utilizzato il protocollo e l'idea architetturale di ATHOS.

Per questo, ARAMIS utilizza gli stessi processi di PORTOS, vale a dire la stessa architettura a due livelli, sicura contro gli attacchi, e gli stessi protocolli di invio e ricezione che, sull'architettura a due livelli, erano funzionali e poco attaccabili.

L'intuizione di ARAMIS nasce dall'osservazione che nel momento in cui un sistema PORTOS è inutilizzabile, a causa di una fase di boot, l'utente potrebbe comunicare solo nell'ipotesi che un altro sistema sia libero. Da questa considerazione, ARAMIS viene pensato come un sistema formato da tanti sottosistemi PORTOS opportunamente collegati tra loro, in numero tale da permettere, in ogni momento, che ci sia almeno un sottosistema PORTOS libero per garantire all'utente di eseguire le operazioni volute.

2.5.1 L'architettura del sistema

Il fulcro dell'architettura di ARAMIS (fig. a pag. 80) è l'insieme dei sottosistemi PORTOS che vengono collegati tramite uno switch alla rete esterna. Questi sottosistemi sono versioni leggermente modificate dell'architettura di PORTOS vista in precedenza (fig. a pag. 81) poiché non comprendono il collegamento diretto con l'esterno, né lo storage delle chiavi che invece viene condiviso tra tutti i sottosistemi.

L'utente che deve accedere al sistema, lo fa attraverso un'interfaccia, ovvero un insieme di dispositivi di I/O, che permettono l'interazione dell'utente col sistema. Tale interfaccia viene dinamicamente collegata al primo sistema PORTOS libero, in un certo istante, tramite uno switch che ha una doppia funzione: la prima è di connettere lo storage al sottosistema

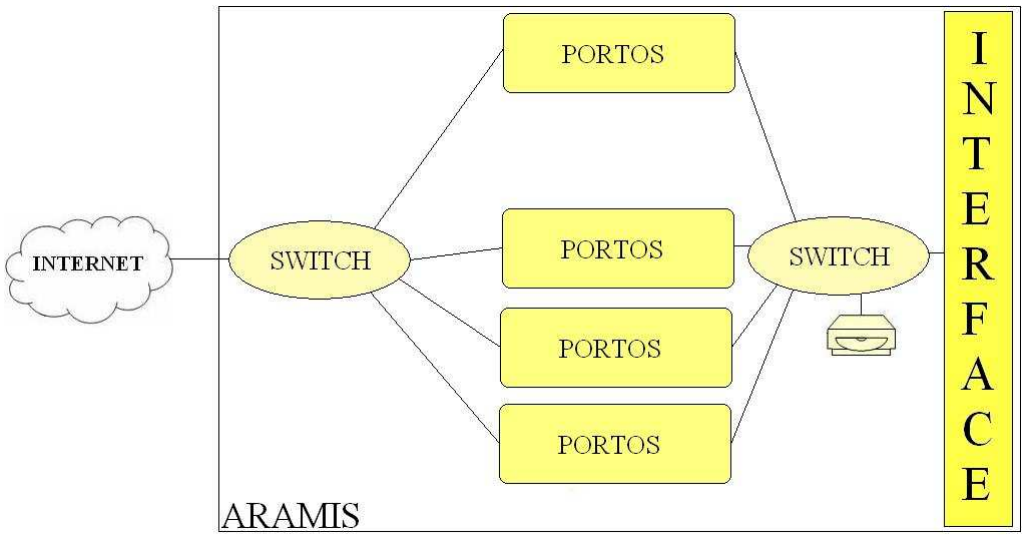


Figura 2.7: L'architettura di ARAMIS

PORTOS che richiede lo storage (sarà poi appannaggio del sottosistema PORTOS dirottare lo storage uno dei due host CASTLE), la seconda è proprio di connettere l'interfaccia, e quindi l'utente, ad un sottosistema PORTOS libero, nel momento in cui l'utente richieda di accedere al sistema.

Il sistema fa uso di due diversi SWITCH. Lo switch esterno, che collega i sottosistemi PORTOS alla rete esterna, è esattamente uguale a quello trattato in PORTOS con la differenza che ad esso si possono collegare un numero di sottosistemi maggiore di due. Inoltre, possono essere attivi più di un collegamento ai sottosistemi PORTOS, poiché tutti gli host VILLAGE dei sottosistemi sono normalmente e di default collegati ad internet.

Lo switch interno è leggermente più complesso: esso, come accennato in precedenza, collega sia l'interfaccia comune ad un sottosistema PORTOS che lo storage al medesimo sottosistema. Pertanto avrà un doppio collegamento con i sottosistemi, uno per il flusso di informazioni tra l'interfaccia e il sottosistema e l'altro per il flusso tra lo storage ed il sottosistema stesso.

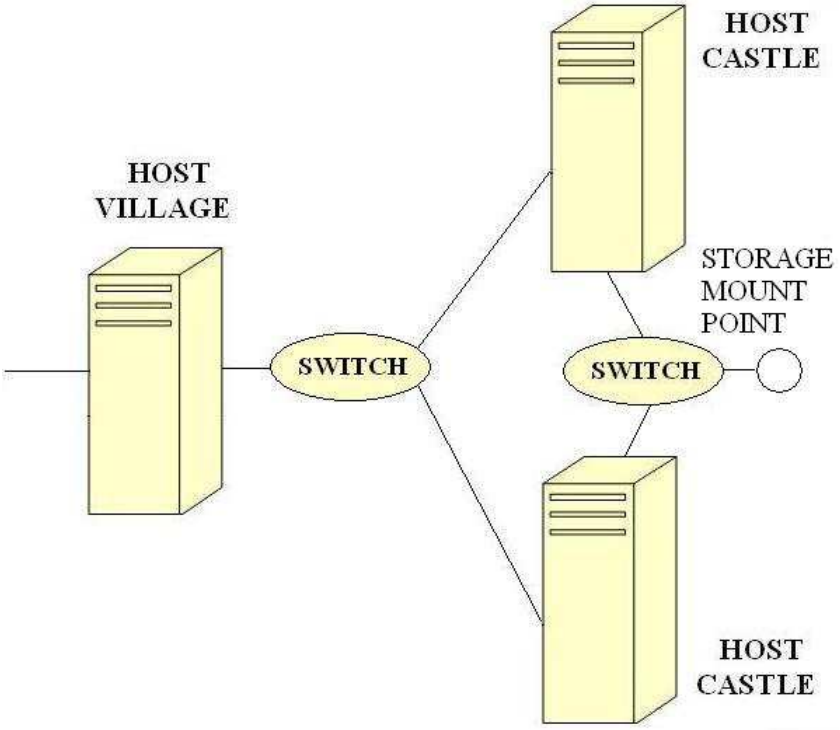


Figura 2.8: L'architettura dei sottosistemi PORTOS in ARAMIS

2.5.2 I flussi operazionali

Vista la complessità dell'architettura di ARAMIS, è bene a questo punto mettere in evidenza quale sia il flusso delle operazioni che si susseguono per realizzare le operazioni di cifratura e decifratura, a partire dal login dell'utente all'interfaccia fino al termine delle operazioni. Di seguito, riportiamo quindi una sequenza di macrofasi che devono susseguirsi:

- **Login dell'utente all'interfaccia comune**. L'utente legale, correttamente riconosciuto come tale dal sistema, accede all'interfaccia comune.
- **Collegamento dell'interfaccia ad un sottosistema PORTOS libero**. Il sistema collega l'interfaccia, tramite lo switch interno ad un sottosistema PORTOS libero. Inizialmente, poiché il sistema, come ATHOS e PORTOS, è monoutente (cioè un unico utente che usa il sistema in ogni istante), tutti i sottosistemi sono liberi.
- **Collegamento dell'interfaccia ad un host CASTLE**. Il sottosistema PORTOS collega l'interfaccia ad uno degli host CASTLE attraverso uno switch interno opportuno, dello stesso tipo di quello interno di ARAMIS.
- **Collegamento dello storage al sistema PORTOS utilizzato**. Lo storage viene collegato attraverso lo switch interno di ARAMIS al sottosistema PORTOS a cui l'interfaccia è collegata.
- **Collegamento dello storage ad uno degli host CASTLE**. Lo switch interno di PORTOS collega lo storage allo stesso host CASTLE a cui ha precedentemente collegato l'interfaccia comune.
- **Esecuzioni dell'operazione**. L'utente esegue l'operazione voluta.
- **Disconnessione di interfaccia e storage**. Una volta eseguita l'operazione di invio o ricezione, nel momento in cui l' host CASTLE è riavviato, lo storage e l'interfaccia vengono disconnessi.

- **Riconnessione per operazione successiva**. A questo punto, se l'utente deve eseguire altre operazioni di invio o ricezione, occorre trovare un altro host CASTLE libero, poiché quello utilizato in precedenza è occupato nel riavvio. Di conseguenza, i passi che ARAMIS segue sono:

  - **Collegamento all'altro host CASTLE**. Se l'altro host CASTLE dello stesso sottosistema PORTOS è libero, l'interfaccia prima e lo storage dopo (come previsto dai protocolli di invio e ricezione che prevedono alcune frasi tra il momento in cui si collega l'utente, in questo caso tramite l'interfaccia comune, e quello in cui si collega lo storage) vengono collegati all'altro host CASTLE.
  - **Collegamento ad un altro sottosistema PORTOS**. Se entrambi gli host CASTLE sono impegnati in operazioni di riavvio, il sistema cerca il primo sottosistema PORTOS libero e vi connette interfaccia e storage secondo i passi indicati in precedenza, nel caso del primo collegamento.

- **Disconnessione dell'utente**. Quando non ha più operazioni da eseguire, l'utente di disconnette dall'interfaccia comune. Interfaccia comune e storage sono disconnessi da qualunque sottosistema PORTOS.

I passi visti utilizzano, anche se non esplicitamente mostrato, lo switch esterno per collegarsi alla rete. Tale switch ha un comportamento totalmente passivo, limitandosi a fornire l'accesso alla rete esterna per l'invio di un messaggio cifrato e limitandosi ad indirizzare sulla porta su cui è in attesa la ricezione il messaggio cifrato proveniente da rete.

2.5.3 Il numero dei sottosistemi PORTOS

Lo scopo di ARAMIS è, come detto, di garantire una disponibilità pressoché totale del sistema, ovvero tale da poter permettere in qualunque momento di eseguire un'operazione. ARAMIS, inoltre, non è un sistema con un'architettura fissa ma ha una variabile, il numero

dei sottosistemi PORTOS, che deve essere scelta in modo tale che la disponibilità totale sia garantita anche nel caso peggiore.

Per determinare il numero minimo di sottosistemi PORTOS necessari, sia N, occorre tenere conto di diverse grandezze, tra le quali, il tempo *massimo* che un host CASTLE impiega per riavviarsi risulta essere quella predominante.

Un'altra grandezza da considerare è il tempo *minimo* di un'operazione di invio e ricezione. Inoltre, si noti che il numero di host CASTLE per ogni sottosistema PORTOS è pari a 2, pertanto il numero totale di host disponibili sarà $2N$.

Infine, un'ultima grandezza da valutare sono i tempi di *overhead* dovuti al montaggio e smontaggio di storage ed interfaccia: tuttavia possiamo supporre, verosimilmente, che queste grandezze siano abbastanza piccole, rispetto ai tempi di boot ed ai tempi di un'operazione, da poter essere considerati infinitesimi e quindi tralasciati. Uno studio che tenga conto anche di questi tempi di overhead va considerato in una fase implementativa del sistema, mentre a questo livello, lo scopo non è di fornire misurazioni precise ma semplicemente l'algoritmo e la logica che sono alla base della scelta del numero ottimale di sottosistemi.

Si vedrà ora in dettaglio il procedimento da seguire per la determinazione del numero minimo di sottosistemi PORTOS necessari.

- Sia t_{op} il tempo minimo di un'operazione (e.g. invio o ricezione del messaggio vuoto).
- Sia T_{boot} il tempo massimo di un riavvio.
- L'uso di ARAMIS è giustificato nel caso in cui $t_{op} << T_{boot}$. Pertanto, è chiaro che in un caso come questo, la disponibilità totale si avrà soltanto nel momento in cui vi siano abbastanza host CASTLE da fare in modo che almeno uno sia sempre libero: questo avviene quando il numero di host CASTLE sia tale da permettere di eseguire operazioni fino a che il primo host CASTLE occupato non si sia riavviato correttamente e sia nuovamente pronto ad essere utilizzato.

- Quanto detto, accade se $2N \times t_{op} \geq T_{boot}$. Sotto questa ipotesi, si garantisce che, nel caso peggiore, sia possibile eseguire comunque tutte le operazioni di lunghezza minima richieste prima che il primo host CASTLE torni ad essere disponibile.

- Da ciò deriviamo che $2N \geq \frac{T_{boot}}{t_{op}}$ e pertanto che il numero minimo di sottosistemi PORTOS richiesti è dato da :

$$N = \left\lfloor \frac{T_{boot}}{2 \times t_{op}} \right\rfloor$$

La stima fatta è una stima *pessimistica*. Nel caso in cui si contino anche gli overhead dovuti allo switching, il numero minimo di sottosistemi diminuirebbe, perché il tempo minimo di un'operazione dovrebbe tenere conto anche dei ritardi di switching, facendo di conseguenza aumentare il tempo dell'operazione a minima durata. Di conseguenza, essendo il numero di sottosistemi inversamente proporzionale al tempo di operazione minimo, tale valore tenderebbe ad essere più basso.

2.5.4 Politiche di switching

Un sistema complesso come ARAMIS potrebbe far pensare alla necessità di un sistema operativo abbastanza complesso da gestire tutte le diverse fasi, e che il sistema operativo MOSES residente sugli host CASTLE non sia sufficiente per sovrintedere le attività di switching.

Nonostante l'apparenza ciò non è vero e per capirlo basta osservare che l'attività di switching è particolarmente meccanica e non richiede processi decisionali particolari che tengano conto di un certo numero di variabili.

Per meglio capire questo concetto, osserviamo le attività dei due switch principali e dello switch interno di PORTOS che permette di montare l'interfaccia e lo storage sugli host CASTLE.

Lo switch esterno esegue operazioni su richiesta senza potere decisionale di alcun tipo: esso, infatti, da un lato riceve traffico dall'interno e lo invia all'esterno appena possibile,

mentre dall'altro riceve traffico da inviare all'unica porta dove è pendente una richiesta di ricezione. La possibilità che ci sia solo un'unica porta su cui possa essere pendente una richiesta di ricezione, è data dal fatto che tutto il sistema è utilizzato da un unico utente (monoutenza) che può essere impegnato solo per un'operazione e solo su un unico host CASTLE; pertanto da un solo sottosistema PORTOS può essere pendente una richiesta di ricezione sullo switch esterno in qualunque momento.

Lo switch interno a PORTOS deve semplicemente collegare l'interfaccia e lo storage ai due host CASTLE. Pertanto, si può supporre che ogni switch di PORTOS si cerchi di collegare al primo host CASTLE oppure, se questo non fosse libero, si cerchi di collegare all'altro[1]. Se in un certo istante nessuno dei due host CASTLE risulta libero, lo switch si disconnette dallo switch interno di ARAMIS, per poi ricollegarsi nel momento in cui uno dei due host CASTLE torna libero.

Lo switch interno di ARAMIS, invece, deve assegnare l'interfaccia comune al primo PORTOS con almeno un host CASTLE libero, e successivamente assegnare allo stesso sottosistema lo storage. Supponiamo che i sottosistemi siano ordinati in qualche modo, ad esempio numerati: lo switch altro non dovrà fare che cercare di collegarsi al primo, se questo fosse disconnesso, poiché inutilizzabile per eseguire operazioni, dovrà cercare di collegarsi al successsivo, in maniera ciclica, fino a quando non ne trova uno disponibile.

Il comportamento degli switch è predeterminato e molto semplice, pertanto non si rende necessario un sistema operativo che sovrintenda tutto il funzionamento di ARAMIS, ma basta applicare le semplici politiche viste e qualche accorgimento a MOSES (i.e. disconnessione automatica dell'host CASTLE dallo switch interno di PORTOS nel momento del riavvio), che non ne aumenta comunque la complessità, per risolvere il problema.

[1]Si deve supporre che MOSES, in qualche modo, segnali allo switch che la macchina è in fase di boot e, pertanto, inutilizzabile (e.g. disconnettendo il collegamento dallo switch, rispistinandolo solo a reboot avvenuto).

2.5.5 Considerazioni finali su ARAMIS

L'unica parte di ARAMIS che non è stata trattata esplicitamente è l'interfaccia comune. Il motivo per cui una descrizione dettagliata non è stata fatta è semplicemente perché tale interfaccia altro non è che un insieme di dispositivi di input/output passivi, il cui utilizzo è sovrinteso da MOSES nel momento in cui viene collegata tramite lo switch ad un host CASTLE, ed utilizzata nello stesso identico modo dei dispositivi di input e di output utilizzati in ATHOS e PORTOS. L'unico dettaglio degno di nota è nel fatto che l'interfaccia deve essere in grado di collegarsi logicamente allo switch interno di ARAMIS nel momento in cui un utente legale vi accede: tutto questo può essere risolto molto semplicemente ed in automatico.

L'utilizzo di ARAMIS è raccomandato soltanto nei casi in cui PORTOS non sia utilizzabile. Questo perché ARAMIS è una soluzione poco economica, soprattutto molto meno economica rispetto a PORTOS: questo infatti ha costi contenuti e, come detto in precedenza, è ottimale nella maggior parte dei casi. ARAMIS potrebbe essere invece usato in casi estremi, qualora i ritardi creatisi per il riavvio degli host CASTLE in PORTOS generassero tempi morti molto elevati a causa di operazioni molto brevi e frequenti.

ARAMIS, infatti, ha costi molto più elevati: se N sono i sottosistemi PORTOS necessari, allora il costo di ARAMIS rispetto a PORTOS è almeno N volte tanto il costo di PORTOS (inoltre vi sono anche i costi degli switch aggiuntivi).

Il problema dei costi, tuttavia, non è il solo: le risorse fisiche allocate per ARAMIS sono scarsamente utilizzate, poiché in realtà si può pensare che solo raramente si possa presentare il caso peggiore, in cui tutti i sottosistemi PORTOS possono venire utilizzati. Di conseguenza, la maggior parte dei sottosistemi PORTOS saranno inutilizzati o utilizzati pochissimo.

La struttura di ARAMIS è molto più complicata di quella di PORTOS, tanto più quando il numero di sottosistemi necessari diventa grande: questo porta una difficoltà di debugging

del sistema a fronte di errori o malfunzionamenti.

Il grande vantaggio di ARAMIS, tuttavia, è quello di essere l'unico sistema tra quelli visti a garantire, se correttamente configurato e dimensionato, la disponibilità *totale*, sotto qualunque condizione di utilizzo. Di conseguenza, diviene estremamente utile in quei casi in cui la tempestività sia un fattore critico, cioè quando l'invio o la ricezione di un messaggio debba essere fatta in tempo reale, ovvero nel momento in cui si presenti la necessità di farlo.

L'architettura di ARAMIS proposta vuole essere solo una guida per la realizzazione di un sistema a disponibilità totale, ma non ha pretese di essere la migliore in assoluto tra quelle realizzabili utilizzando gli stessi protocolli di ATHOS e PORTOS.

Ad esempio, in fase di implementazione del sistema, si potrebbe valutare l'ipotesi di non utilizzare uno switch interno ai sottosistemi PORTOS ma collegare direttamente lo switch interno di ARAMIS agli host CASTLE dei sottosistemi. Inoltre si può pensare persino di non utilizzare sottosistemi PORTOS ma N host CASTLE collegati tramite switch ad un unico host VILLAGE a sua volta collegato all'esterno. Lo scopo dell'architettura di ARAMIS presentata, quindi, è quello di voler essere il più generale possibile, nell'impossibilità, a questo livello, di valutare quale possano essere la scelte migliori, prima della realizzazione pratica del sistema stesso.

Capitolo 3

Il problema dell'identifcazione degli utenti

3.1 Il problema

I protocolli visti in precedenza costituiscono una protezione efficace per il dispositivo di storage solo sotto l'ipotesi che gli utenti che accedono all'host CASTLE contenente lo storage, e quindi la chiave, siano effettivamente chi dicono di essere, cioé coloro che legalmente hanno pieno diritto all'accesso dello storage.

Così come la violazione del dispositivo di storage compromette la sicurezza dell'intero sistema QCRYPT, allo stesso modo l'erronea identificazione di un intruso (sia Eve) come utente legale comporta la totale compromissione della sicurezza dello storage realizzata dai protocolli visti e, di conseguenza e transitivamente, quello dell'intero sistema quantistico.

Violare l'identificazione quindi comporta violare direttamente l'intero sistema: nel momento in cui un Eve viene identificato come "legale" (i.e. come Alice o Bob), di fatto acquisice i diritti di un utente sull'intero sistema, dalla possibilità di accedere alla chiave fino all'uso delle operazioni di scambio sicuro di messaggi per carpire informazioni all'altra parte (di fatto convinta di essere in comunicazione con l'utente legale).

L'introduzione dello storage delle chiavi fa assumere al problema dell'identificazione

un'importanza ancor più strategica. Nel protocollo di scambio di chiavi QCRYPT è infatti previsto l'uso di una chiave (condivisa) per l'identificazione solo all'inizio della fase di distribuzione della chiave, i successivi scambi di messaggi (e.g. liste di basi) su canale pubblico sono autenticati utilizzando parte della chiave fino a quel punto generata. Inoltre nell'ipotesi iniziale le due parti restano collegate al sistema portando avanti una comunicazione bilaterale, fino a che la loro conversazione è finita, e generando una mole di chiavi sufficienti per quella sessione.

Tutto questo era valido sotto le ipotesi in cui le sessioni di scambio fossero poche e lunghe e che ci fosse la reale possibilità di avviare il sottosistema quantistico con facilità e rapidità ogniqualvolta si fosse presentata la necessità di una comunicazione tra Alice e Bob.

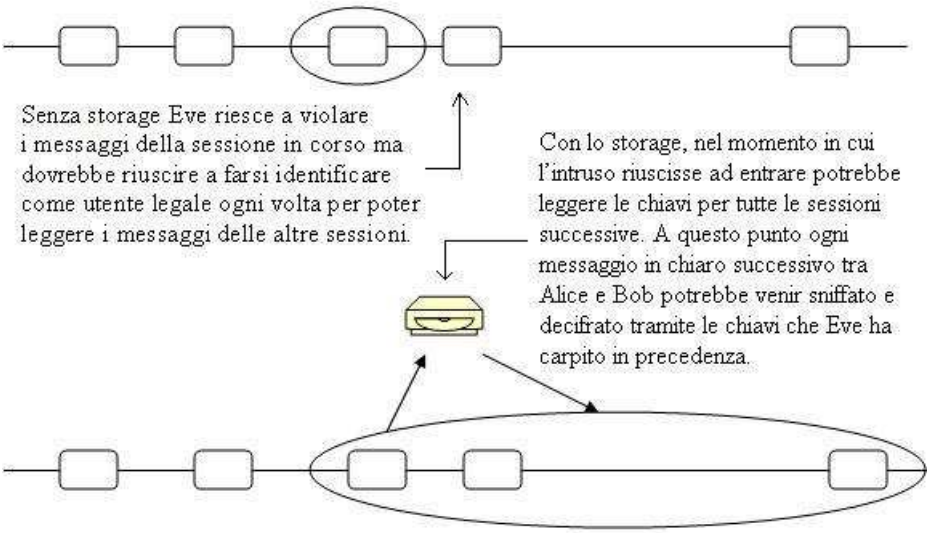
Poiché lo studio dell'implementazione reale del sistema, in un contesto il più possibile generale, ha sottolineato come l'overhead ed i tempi di avvio siano troppo grandi per giustificare l'avvio del sottosistema quantistico ad ogni sessione, si è reso necessario utilizzare la macchina per lunghi periodi generando chiavi a sufficienza per "coprire" un certo numero di sessioni di comunicazione tra due parti, salvandole su un apposito dispositivo di storage.

A questo punto occorre fare un'attenta riflessione sul perché questo approccio faccia acquisire all'identificazione una maggiore importanza. Nell'approccio originale, se Eve si fosse sostituita ad Alice o Bob per una sessione di comunicazione, sarebbe venuta a conoscenza, al più, delle informazioni di quella sessione.

Nel secondo approccio, invece, l'identificazione di un intruso come utente legale comporterebbe la possibilità dello stesso di carpire tutte le chiavi dello storage, venendo dunque a conoscenza di un quantitativo di chiavi tale da poter utilizzare per decifrare le successive comunicazioni tra Alice e Bob.

É chiaro quindi che l'identificazione in questo scenario deve essere tale che una situazione di questo tipo non si presenti, ovvero, in caso di errata autentificazione, sia possibile limitare il più possibile i dati che possa carpire dallo storage, in modo tale da rendere sicure il maggior

numero di sessioni successive possibili.



Queste osservazioni sottolineano quanto, prima di preoccuparsi dell'identificazione vera e propria degli utenti, sia necessario determinare un modo per limitare il più possibile i danni delle sostituzioni di intrusi, senza limitare per questo i privilegi ed i diritti degli utenti legali.

3.1.1 Limitazione della possibilità di interazione con lo storage

La prima considerazione che va fatta è che l'accesso illimitato allo storage permette di essere in possesso contemporaneamente di tutte le chiavi.

Questo in primis è pericoloso per i motivi spiegati in precedenza. Inoltre l'avere accesso alla totalità delle chiavi, per un utente legale è comunque inutile in quanto, in un certo istante e per cifrare un certo messaggio, all'utente non interessa altro che avere un numero di bit di chiave sufficienti a cifrare lo stesso, senza nemmeno dover essere a conoscenza della natura di questi bit di chiave.

Un primo vincolo che si può imporre per garantire che sostituzioni impreviste rechino danni enormi, compromettendo la sicurezza di tutte le chiavi salvate sul dispositivo, è

quello di non permettere interazioni dirette tra l'utente e lo storage, utilizzando opportunamente un filtro che si occupi di recuperare la quantità di chiave necessaria dallo storage ed utilizzarla per cifrare il messaggio in chiaro dell'utente senza metterne il medesimo a conoscenza.

Il filtro migliore a tale scopo potrebbe essere MOSES, il sistema operativo su ROM residente sull'host CASTLE. Tale sistema operativo deve fare in modo di frapporsi tra ogni interazione tra l'utente e l'host CASTLE in modo da evitare un'interazione diretta. Ogni richiesta di accesso allo storage, da parte dell'utente, deve dunque essere vagliata da MOSES e realizzata dopo l'approvazione della stessa. Nella prima stesura dei requisiti di MOSES, infatti, si era sottolineata la necessità di implementare una primitiva che potesse essere usata dall'utente per leggere dallo storage; in pratica l'utente attraverso una primitiva del sistema operativo accederebbe direttamente al dispositivo ed avrebbe la possibilità, di eseguire direttamente operazioni sullo storage, dal prelevamento alla cancellazione della chiave.

Nell'idea originale, infatti, MOSES risulta essere un fornitore di servizi su richiesta, ovvero un sistema che offre la possibilità all'utente di eseguire determinate operazioni, predefinite in numero e quantità, e fortemente standardizzate. Il flusso operativo viene comunque gestito dall'utente che realizza le varie fasi di ATHOS o PORTOS chiamando, quando necessario, le diverse primitive del sistema operativo. Lo schema generale di funzionamento è dato in fig. a pag. 93.

Ogni fase del protocollo è quindi delegata all'utente che ne gestisce il flusso. Questo approccio dà maggiore libertà all'utente di gestire la cifratura/decifratura con una maggiore flessibilità.

A questo punto però, dopo le considerazioni fatte inizialmente, occorre realizzare l'accesso allo storage non in maniera diretta ma in modo da passare attraverso il sistema operativo stesso e ad esso delegare l'intero flusso esecutivo. Per questo motivo, la richiesta di accesso

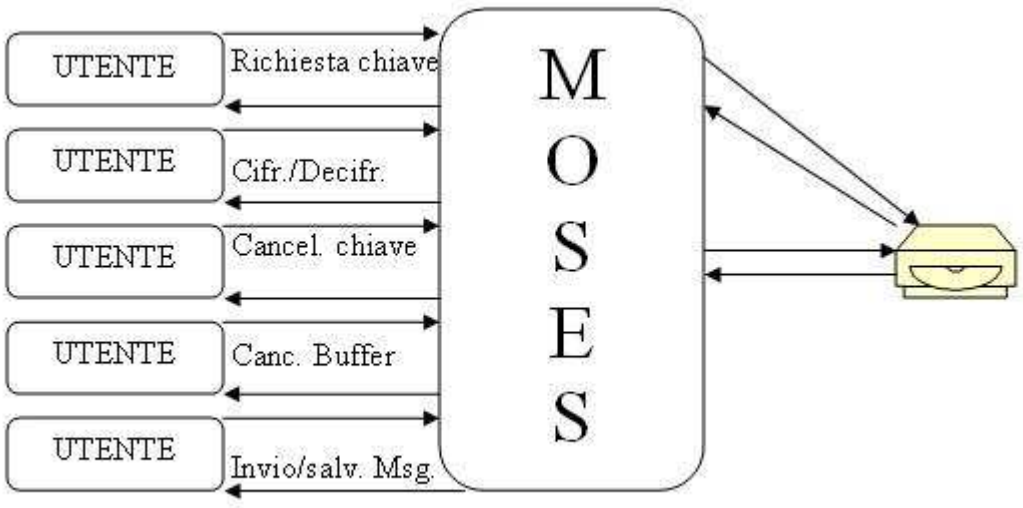


Figura 3.1: Interattività tra MOSES e l'utente nell'idea originale

allo storage deve essere suddivisa in due parti ed in altrettante primitive.

Questo comporta un nuovo modo di accedere allo storage ed un nuovo modo di realizzare operazioni di cifratura/decifratura, delegando l'intera responsabilità delle operazioni al sistema operativo, per il quale si può assumere che non sia malizioso (a differenza dell'utente che potrebbe esserlo). All'atto di accedere allo storage, l'utente deve utilizzare un'opportuna primitiva per informare MOSES della necessità di cifrare/decifrare un messaggio e passare ad esso il messaggio da cifrare. Da questo momento in poi l'intera attività di realizzare la cifratura/decifratura, con annesso accesso allo storage è interamente delegata e realizzata dal sistema operativo MOSES.

Un approccio di questo tipo, che delega a MOSES l'incombenza di gestire l'intero processo di cifratura/decifratura, porta necessariamente ad una maggiore complessità del sistema operativo stesso ma tale aumento di complessità non dovrebbe essere tale, in linea teorica, da non verificare il vincolo di minimalità di MOSES: ciò significa che la struttura interna del sistema operativo, benché complicata maggiormente da questo nuovo vincolo, dovrebbe

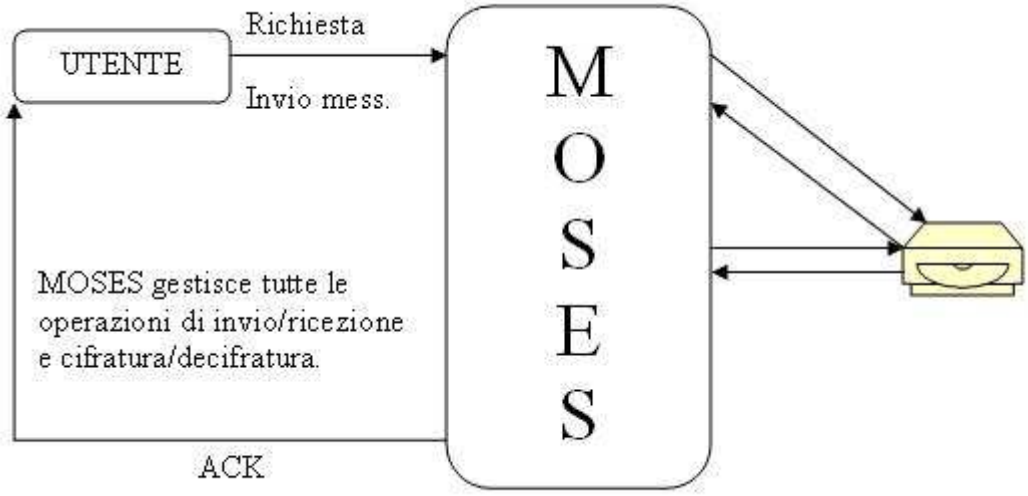


Figura 3.2: Interattività modificata tra MOSES e l'utente

comunque mantenersi abbastanza semplice e lineare.

La delega a MOSES dell'intero processo si rende comunque necessaria, anche a fronte di un aumento di complessità, poiché garantisce che nessuna persona, nemmeno se autorizzata, possa disporre liberamente dei dati sullo storage, prevenendo quindi ogni forma di accesso alle chiavi, maliziosa o meno, che non sia strettamente necessario.

Si noti che l'ipotesi di semplicità di MOSES deve essere mantenuta valida in ogni fase di sviluppo del sistema ma sarà in fase di implementazione che sarà fondamentale determinare strategie tali da garantire questa ipotesi e tutti i vincoli necessari alla sicurezza mostrati finora.

Quanto evidenziato permette dunque di limitare i danni ed i privilegi che un utente malizioso, connesso al sistema per scambio di identità con uno legale, potrebbe ottenere; il problema però che ancora non è stato affrontato, né risolto, è come impedire, o perlomeno rendere il più arduo possibile, l'identificazione di un intruso come utente legale.

3.2 SCANNER - Smart Card Authentication with New Number at Every Reading

Risolto il problema di perdere la segretezza dei dati a fronte di un'identificazione errata di un intruso, occorre analizzare delle metodologie che siano abbastanza affidabili e sicure nell'identificazione dell'utente, cioé tali che la probabilità di identificare correttamente l'utente tenda al 100%.

Inoltre, nel caso in cui si verifichi (probabilità che deve essere infinitesima) un errore di identificazione, occorre ridurre al minimo, o rendere praticamente nulla, la quantità di informazioni che possono venir accedute dall'attaccante, a partire dal momento dell'errata identificazione in poi.

Il primo punto su cui riflettere sta nel fatto che un utente malizioso, scoperto un modo per essere identificato come legale, potrebbe usare sempre questo metodo per sostituzioni successive.

Di conseguenza è necessario innanzitutto impedire che un errata identificazione si possa ripetere una seconda volta con le stesse modalità, pena la compromissione di tutte le sessioni di comunicazione successive a quella dell'intrusione.

Fatta questa precisazione, bisogna fare in modo che l'identificazione dell'utente avvenga attraverso un metodo che sia estremamente sicuro: pertanto non deve essere sensibile, od esserlo il meno possibile, ad errori dell'utente stesso.

Questa ipotesi mostra chiaramente che un'identificazione attraverso un segreto od una password non è attuabile in quanto l'utente è *troppo* responsabile della conservazione e della non *divulgazione* del segreto.

Inoltre tale tipo di identificazione è inutilizzabile poiché, in qualunque caso, non dà adeguate garanzie di sicurezza. Questo infatti ridurrebbe i rischi di un'errata gestione del segreto da parte dell'utente solo se la password da ricordare fosse breve ed unica.

Se così fosse però la password potrebbe venir individuata nel tempo da un attaccante ed

usata per identificarsi al posto dell'utente legale. Affinché un'identificazione di questo tipo sia abbastanza sicura, occorre cambiare la password con una frequenza tale da garantire il livello di sicurezza voluto: in pratica più piccolo è il periodo di utilizzo di una password, minori le probabilità che la password possa venire carpita ed utilizzata da un attacante.

Nel nostro scenario, tuttavia, la percentuale di sicurezza richiesta é molto elevata e questo comporterebbe un cambio di password molto frequente: il numero di password diverse che l'utente si troverebbe costretto a ricordare sarebbe talmente spropositato che si troverebbe in qualche modo costretto a trascriverle o tenerle salvate da qualche parte con il risultato che il rischio di divulgarle erroneamente, o che vengano carpite in qualche modo, aumenti in maniera considerevole.

Ciò che emerge dalle precedenti riflessioni è che non è consigliabile quindi adottare un metodo basato su un segreto condiviso, preferendo invece basarsi su un approccio che permetta di identificare l'utente in funzione di "qualcosa che possiede" (e lo possiede solo lui!) rispetto a "qualcosa che sa".

L'utente quindi si identifica con il sistema mostrando di essere in possesso dell'oggetto che lo contraddistingue unicamente e lo identifica.

Questa idea potrebbe sembrare non molto diversa da quella precedente in quanto le possibilità di perdere l'oggetto sarebbero comunque alte. In questo caso, tuttavia, l'utente si accorgerebbe di aver perso o di non essere più in possesso dell'oggetto e potrebbe bloccarne l'utilizzo, revocandone la validità, ed impedendo, di conseguenza, l'utilizzo del medesimo da parte di un attaccante per identificarsi al suo posto.

Una possibile situazione che potrebbe verificarsi però, è che l'oggetto possa venir copiato ad insaputa dell'utente, che non si accorgerebbe di nulla, ed utilizzato successivamente per identificare l'attaccante. In questo caso la sicurezza dell'identificazione tenderebbe a zero e quindi l'affidabilità dell'intero sistema verrebbe meno.

Di conseguenza, una caratteristica imprescindibile che viene richiesta all'oggetto è di

non essere suscettibile alla copia, o almeno che l'avvenuta copia possa dare un feedback all'utente legale in modo tale da invalidare l'uso dell'oggetto copiato.

Tutte queste osservazioni fanno propendere per una scelta basata su smart card che sia scrivibile e che venga creata e distribuita nella fase di inizializzazione del sistema, ed una unica per ogni utente. Occorre ora determinare quale tipologia di smart card si adatti il più possibile al sistema in questione.

3.2.1 Valutazione delle tipologie di identificazione attraverso supporto fisico

Questa sezione vuole occuparsi di visitare velocemente ed in maniera critica le odierne tipologie dei sistemi di identificazione mediante supporto fisico al fine di determinare la tipologia che maggiormente si confaccia al caso in questione.

Il primo sistema che prendiamo in considerazione sono le cosiddette sono le carte a banda magnetica (**magnetic stripe card**). Le carte di questo tipo sono molto usate come carte di credito e nascono con l'intento di custodire dati in un formato che sia leggibile da una macchina generica, con l'esigenza di minimizzare l'uso di carta nelle transazioni finanziarie e di essere utilizzate per permettere l'automazione di determinati processi (e.g. il pagamento o il prelievo di contante). Sono formate da una banda magnetica di colore scuro dello spessore di 1,2 cm a sua volta formata da tre bande magnetiche. Su tali bande, l'informazione è codificata attraverso la polarizzazione delle particelle contenute nelle tre bande.

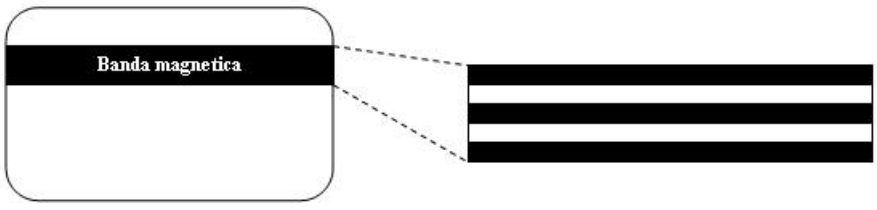


Figura 3.3: Carte a banda magnetica

Ogni banda è divisa in domini ognuno dei quali contiene un bit di informazione in funzione della polarizzazione contenuta in quel dominio. Il bit di informazione 0 è codificato attraverso una polarizzazione uniforme nel dominio mentre il bit di informazione 1 é polarizzato su un dominio in cui sussistono polarizzazioni uguali e contrarie.

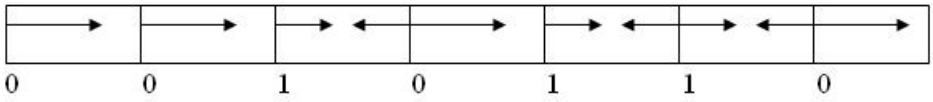


Figura 3.4: Magnetizzazione di una banda

La lunghezza di ogni banda è di circa 9,6 cm ed in ognuna di esse possono esserci più di 300 domini per un totale di bit di dati massimo compreso tra i 900 ed i 1000 bit.

Le carte a banda magnetica sono molto economiche ed hanno modalità e tempi di realizzazione molto semplici e brevi. Il problema principale di queste carte è che sono facilmente violabili: esse possono infatti essere lette e modificate da chiunque in possesso dei dispositivi corretti.

Il processo di lettura di una carta a banda magnetica e la conseguente copiatura su una seconda carta (che prende il nome di Card Skimming) avviene attraverso due appositi dispositivi: un lettore, che ha mediamente un costo attorno ai $100 ed un encoder che ha invece un costo di circa $1000 dollari se abbastanza economico.

É chiaro quindi che l'equipaggiamento per violare la segretezza delle informazioni su una carta di questo tipo è appannaggio di tutti. Inoltre, il processo di lettura e copia è di per se stesso molto veloce e poco complesso.

Tutte queste osservazioni sanciscono la non utilizzabilità di questa carta per proteggere e conservare dati confidenziali, pertanto tale tecnologia non è adatta per una funzione di identificazione dell'utente, poiché non fornisce abbastanza garanzie sull'identità di chi la possiede, in quanto facilmente copiabile. Successive alle carte a banda magnetica e con una

tecnologia decisamente più complessa, sono invece le **smart card**.

Le smart card sono anch'esse carte di plastica, di dimensioni simili se non uguali alle carte a banda magnetica, ma contenenti al loro interno un chip od una memoria su cui sono salvate informazioni ed eventualmente elaborate. Ad esempio i dati non sono necessariamente salvati in chiaro come sulle carte a banda magnetica, ma vi si può applicare un processo crittografico in modo tale che le informazioni non siano in chiaro sul chip.

La famiglia delle smart card è molto vasta. Pertanto vi sono tre diversi modi di classificare le smart card, in funzione del componente presente all'interno, in funzione del tipo di interfaccia con i reader/writer usati per trasferire da e ad essa le informazioni, ed in funzione del sistema operativo presente su di essa.

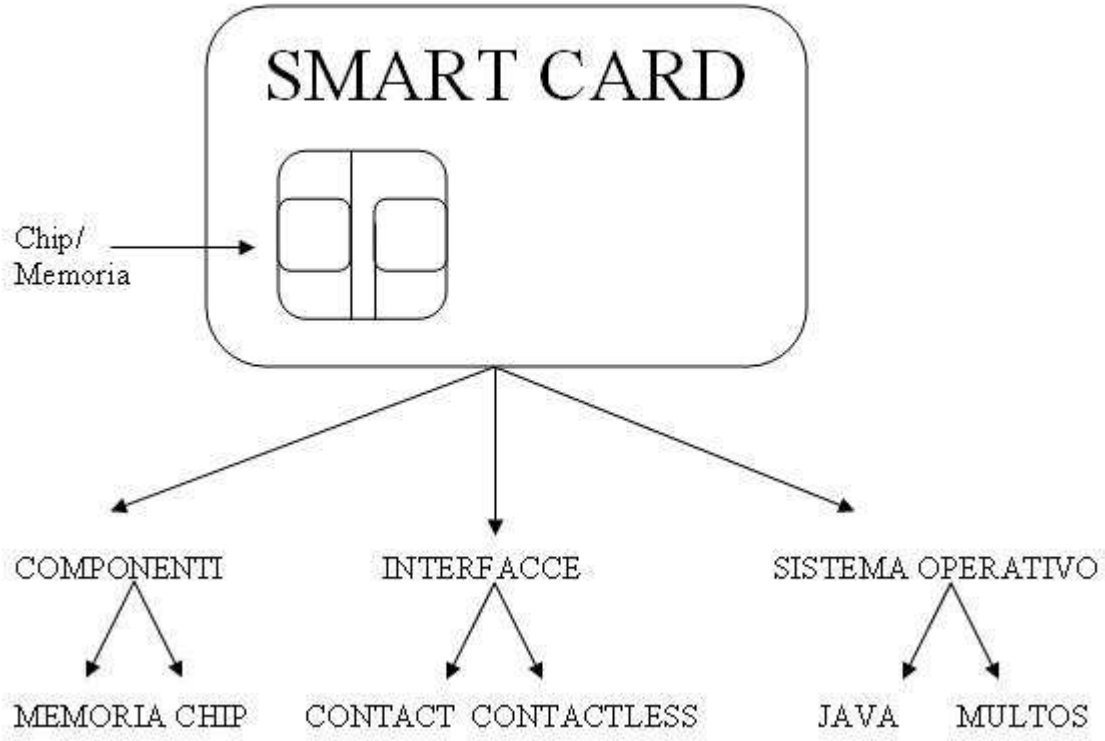


Figura 3.5: Smart Card e tipologie di classificazione

La prima classificazione delle smart card dipende dalla presenza o meno di un microprocessore in grado di eseguire operazioni sulle informazioni salvate (CHIP CARD) o meno (MEMORY CARD).

Le **memory card** sono le più diffuse e le meno costose. Di solito contengono una EEPROM (Electronically Erasable Programmable Read Only Memory) che costituisce la parte di storage della smart card su cui le applicazioni possono scrivere i dati, con il vantaggio però che i dati possono essere bloccati attraverso un PIN. Oltre alla EEPROM, sulle memory card si trova una ROM che contiene informazioni che non cambiano durante il ciclo di vita della carta.

La lettura e la scrittura sono regolate da una logica di controllo, mentre l'accesso alle regioni di memoria delle EEPROM sono possibili previa l'immissione di una password che può essere chiesta all'utilizzatore oppure insita nel reader e nel writer.

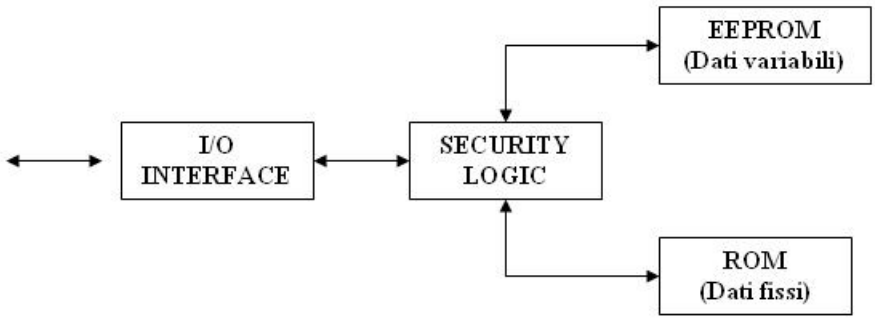


Figura 3.6: Memory Card

Questo tipo di smart card può contenere da un minimo di 100 bytes fino ad un massimo di 8 KB. La tecnologia è molto economica ed il costo di una smart card di questo tipo si aggira intorno al dollaro. Grazie alla sua economicità e semplicità, tale tipo di carta viene molto usata nelle carte telefoniche prepagate. Le **Chip Card** invece, contengono sempre una ROM ed una EEPROM con le medesime funzioni di quelle presenti nelle Memory Card ma in più contengono una CPU che esegue operazioni sui dati in EEPROM ed una RAM

che funge da buffer per le operazioni eseguite dalla CPU. L'architettura interna di una Chip Card in genere è organizzata come in fig. a pag. 101:

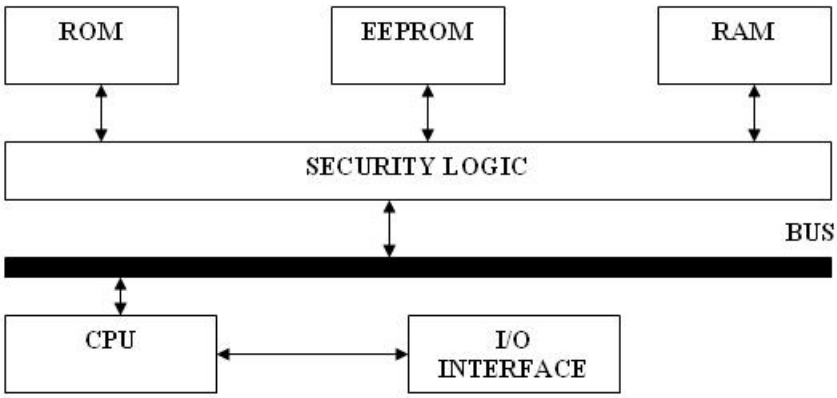


Figura 3.7: Chip Card

Queste carte sono leggermente più costose (da $2 a $20) delle Memory Card ma permettono di aumentare la sicurezza proteggendo (e.g. attraverso la crittografia) maggiormente i dati nella EEPROM. Il secondo tipo di classificazione delle smart card riguarda le interfacce cioé il modo con cui la smart card si connette agli appositi dispositivi per inviare e ricevere le informazioni.

In questa categoria distinguiamo tra Contact Card e Contactless Card. Le **Contact Cards** devono essere inserite all'interno di un apposito lettore affinché vi si possa accedere. Ogni carta contiene tra i 6 e gli 8 contatti dorati che entrano in diretto contatto con il dispositivo di lettura e scrittura, e la stessa carta riceve energia per eseguire le operazioni direttamente dal dispositivo stesso. A titolo di esempio, la struttura della superficie di contatto nello standard ISO-7816 è mostrata in fig. a pag. 102

A differenza delle Contact Card, le **Contactless Card** utilizzano segnali radio e non richiedono di essere inserite in un dispositivo.

L'area di transazione, ovvero la distanza massima dal dispositivo per poter eseguire una

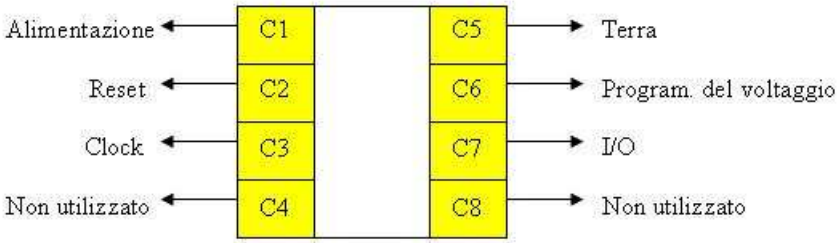


Figura 3.8: Standard ISO-7816

transazione varia tra pochi centimetri fino ad un massimo di cinquanta. Le Contactless Card hanno un periodo di vita più lungo e sono più affidabili delle Contact card ma hanno anche costi maggiori.

Le smart card, infine, sono classificate anche attraverso il sistema operativo residente sulla ROM della carta.

Vi sono diversi sistemi operativi utilizzabili, tra i quali i più diffusi sono MultOS e JAVA.

Si noti che le smart card con sistema operativo presuppongono necessariamente di dover eseguire qualche tipo di operazione, pertanto questo genere di carte sono necessariamente chip card. **MultOS** è un sistema operativo che permette di salvare diversi programmi applicativi nella EEPROM della smart card, mentre in precedenza ciò non era possibile.

Inoltre, gli applicativi installati tramite MultOS sono tutti indipendenti dalla piattaforma poiché MultOS implementa una macchina virtuale. MultOS è importante, e di per sé è rivoluzionario, in quanto in precedenza gli sviluppatori dovevano riscrivere una nuova versione dell'applicativo per ogni tipo di smart card. Una nota particolare meritano anche le smart card basate su Java, dette **Java Cards**.

Una Java Card è una smart card basata sulla tecnologia java ed in grado di eseguire applicativi java appositamente realizzati (applet). L'architettura di una java card è data in fig. a pag. 103.

L'architettura a livelli è formata da un livello fisico che corrisponde al sistema di circuiti

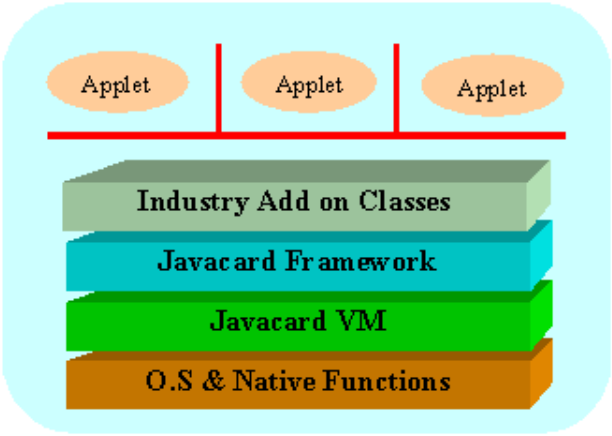


Figura 3.9: Architettura di una Java Card

integrati della carta su cui gira un sistema operativo minimo (nativo) che esegue operazioni base a livello macchina.

La java card richiede come supporto fisico una chip card che contenga almeno una ROM di 16 KB, una EEPROM di 8 KB ed una RAM di 256 B che funga da buffer. Sopra il livello fisico, si trova una apposita JVM (Java Virtual Machine) con il compito di eseguire ed interpretare il bytecode (cioé il codice compilato) degli applet. La JVM inoltre funge da filtro per rendere trasparente i dettagli proprietari della tecnologia fisica sottostante ai livelli superiori.

Sopra la JVM si trova un apposito framework che fornisce librerie di supporto (API) agli sviluppatori di applet e che dagli stessi applet possono venire utilizzati per ottenere servizi; inoltre è previsto un ulteriore livello in cui eventuali add-on di terze parti, dopo opportuna certificazione, possono essere aggiunti. Infine, il livello più alto è dedicato agli applet che possono riguardare molteplici tipologie di programma (e.g. applet per transazioni sicure, per identificazione e riconoscimento, etc.).

Le java card hanno il vantaggio dal punto di vista della sicurezza di sfruttare le caratteristiche di un linguaggio di per se stesso robusto e sicuro, oltre che ad avere grandi

possibilità di sviluppo, vista l'ampia diffusione della tecnologia Java.

A questo punto, per poter determinare quale sia la tipologia migliore di smart card per gli scopi di sicurezza prefissati, occorre formalizzare con precisione quale sia la strategia che si vuole adottare per ridurre il più possibile la probabilità di intrusione e per mantenere al minimo le informazioni accedute a fronte di un'identificazione errata.

3.2.2 L'idea di SCANNER

L'utilizzo di una smart card di per se garantisce che i dati su di essa contenuti siano difficilmente accessibili da fonti non autorizzate, a meno di un accesso a forza bruta ed una violazione fisica della smart card stessa.

Tuttavia, se un attaccante venisse in possesso della smart card, questi potrebbe utilizzarla per identificarsi come l'utente legittimamente proprietario della stessa.

É chiaro che la sparizione della smart card metterebbe in sospetto il legittimo proprietario, che, non trovandola, potrebbe denunciarne la scomparsa al gestore del sistema (nel nostro caso al sistemista responsabile degli host CASTLE) in modo tale che l'accesso al sistema attraverso quella smart card venga inibito. Alla luce di questa considerazione, è però opportuno fare delle osservazioni che si rendono necessarie quando lo scopo è di mantenere un livello di sicurezza tendente il più possibile al 100%.

In primo luogo, si noti che anche qui il fulcro di tutta la sicurezza ruota attorno all'utente: è proprio l'utente, infatti, che in una circostanza come questa deve accorgersi della sparizione; pertanto va considerato che i tempi impiegati per rilevare la perdita possono essere molto grandi.

Nel frattempo, l'attaccante può aver utilizzato la smart card per identificarsi un numero anche grande di volte ed aver carpito di conseguenza eventuali informazioni provenienti dalla controparte (non può più carpire direttamente la chiave visto le limitazioni imposte a MOSES, ma può comunque venire in possesso di informazioni decifrando messaggi in arrivo

all'utente legale).

É chiaro che in una conversazione con la controparte, può essere la stessa controparte ad accorgersi di non stare "conversando" con l'utente legittimo, ma poiché occorre porsi sempre nel caso peggiore, supponiamo che l'intruso sia abbastanza "bravo" da non farsi riconoscere.

In secondo luogo, occorre notare che, se l'utente utilizza sempre la medesima smart card per identificarsi e con le stesse informazioni in essa contenute, si può presentare la peggiore situazione in un ambito di identificazione come questo: l'attaccante infatti potrebbe entrare in possesso della smart card per un periodo limitato di tempo entro il quale ne realizza una copia e fa ritornare l'originale all'utente, in modo tale che questo non si accorga dell'avvenuta sottrazione.

Da questo ne consegue la necessità di dover utilizzare dati di identificazione sempre diversi per ogni sessione ed in modo tale che questi garantiscano che l'utente che si sta identificando sia lo stesso identificato nella sessione precedente e non possa essere che lui. Questo implica che i dati di identificazione sulla smart card devono essere tali da garantire all'utente, al termine di una sessione, la sua corretta identificazione anche nella sessione successiva, ma in modo tale che nessun altro possa identificarsi al suo posto.

L'idea da cui si parte, dunque, è di basare l'identificazione dell'utente, per la sua n-esima sessione di comunicazione, su un segreto che sia determinato durante la sessione (n-1)- esima e quindi tale che nessun intruso ne possa essere a conoscenza in alcun modo, dal momento che non ha partecipato alla sessione.

Questo approccio richiederebbe di calcolare un segreto ad ogni sessione in maniera che tale segreto sia sufficientemente sicuro.

In pratica, l'unico vincolo è che la generazione del segreto attraverso funzioni pseudo-random o hash richiede che queste siano abbastanza buone e che garantiscano che il segreto generato sia sicuro (cioé il più possibile casuale con una probabilità uniforme sul condominio

della funzione).

Nel caso in questione, però, questo problema può essere risolto molto facilmente, senza dover determinare alcuna funzione di generazione né tantomeno preoccuparsi del fatto che tale funzione possa essere buona o meno.

Ogni identificazione, infatti, permette all'utente di accedere al sistema per cifrare/decifrare messaggi attraverso l'uso di chiavi che sono totalmente random per definizione, da usare per l'One Time Pad. Di conseguenza, ad ogni identificazione si può pensare di usare come chiave per la sessione successiva una parte della chiave contenuta nello storage.

In pratica, quindi, ad ogni sessione, dopo che è avvenuta correttamente l'identificazione dell'utente tramite la chiave presente sulla chiave, si elimina tale chiave inserendo al suo posto altrettanti bit di chiave nuovi, prendendoli dai primi disponibili sullo storage. Per il primo utilizzo, cioè per identificare l'utente alla prima sessione, si utilizza una chiave inserita al momento dell'inizializzazione della smart card; tale chiave può essere a sua volta una parte della chiave presente nello storage, che ovviamente deve essere eliminata dallo stesso, poiché non utilizzabile per cifrare.

L'algoritmo per l'identificazione e l'aggiornamento, è indicato più formalmente di seguito, dopo aver specificato meglio le componenti hardware necessarie per il processo di identificazione.

3.2.3 Architettura hardware per l'identificazione

Il processo di identificazione mediante smart card richiede innanzitutto un apposito lettore in grado anche di modificare il contenuto nella EEPROM della smart card. Il lettore deve essere opportunamente collegato all'host CASTLE relativo: per ATHOS vi deve essere un lettore per ogni host CASTLE, mentre per PORTOS può essere collegato fisicamente solo ad uno degli host CASTLE (analogamente per ARAMIS). Il lettore deve essere fisicamente connesso ma disconnesso di default. Deve avere la capacità di connettersi e richiedere

funzioni all'host CASTLE attraverso MOSES. Il lettore deve avere qualche capacità di calcolo. Nello specifico deve contenere:

- Tre messaggi predefiniti. ASK per la richiesta di identificazione, ACK per l'accettazione dell'identificazione, NACK per il rifiuto. Questi messaggi sono standard, fissi e predeterminati, uguali su tutti i lettori del sistema.

- Primitive per il calcolo di una funzione hash su un messaggio di richiesta fissato, data una chiave (ASK_CALC) Il lettore deve avere la capacità di calcolare una funzione hash, prestabilita ed uguale per tutti i lettori del sistema, su un messaggio predeterminato detto ASK e in funzione del valore di chiave letta dalla smart card.

- Primitive per il calcolo di una funzione hash su un messaggio di accettazione fissato, data una chiave (ACK_CALC). Primitive analoghe alle precedenti, calcolano la stessa funzione hash ma su un messaggio predefinito di risposta, detto ACK.

- Primitive per il calcolo di una funzione hash su un messaggio di rifiuto fissato, data una chiave (NACK_CALC).

- Primitive per la composizione di un messaggio di richiesta (REQ_CALC). Un messaggio di richiesta (REQUEST) è un messaggio che contiene l'identificativo di un utente legale ed una firma digitale costituita dalla funzione hash calcolata dal lettore su ASK, in funzione della chiave letta dalla smart card.

- Primitive per la composizione di un messaggio di risposta (ACCEPT_CALC). Il lettore deve poter rispondere ad una request con una ACCEPT nel caso in cui il la richiesta sia accettabile in quanto presente la controparte richiesta in attesa sulla macchina.

- Primitive per l'invio di messaggi all'host CASTLE (SEND_MESS). Il lettore deve poter mandare i messaggi di REQUEST e ACK all'host CASTLE per essere spediti.

Queste primitive devono tenere traccia anche del tipo di ogni messaggio (ask, ack, nack).

- Primitive per l'accettazione di un digest dall'host CASTLE. (DIG_ACCEPT) Il lettore deve poter accettare richieste di invio di un digest da parte dell'host CASTLE.

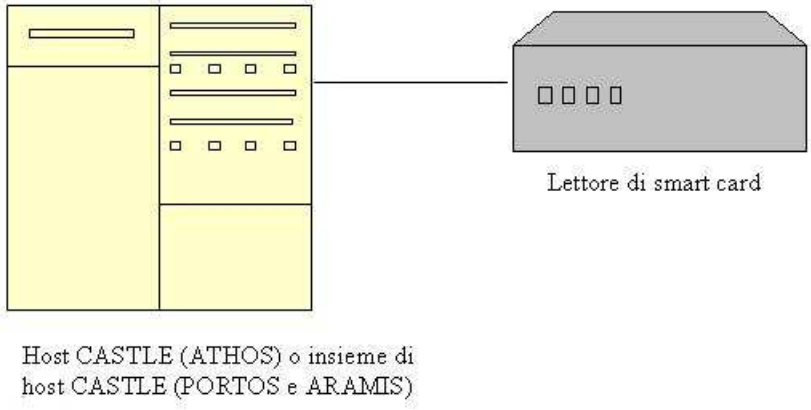


Figura 3.10: Architettura di SCANNER

Lo stesso MOSES va modificato opportunamente aggiungendo altre primitive necessarie per la realizzazione del sistema di identificazione. Più nello specifico, deve contenere:

- Primitive per la ricezione di una REQUEST dal lettore (REQ_REC). MOSES deve poter accettare l'invio di una REQUEST dal lettore, riceverla, elaborarla, estrarre l'id della controparte per identificare l'host CASTLE su cui si trova.

- Primitive per l'invio di un digest ad un host CASTLE remoto (SEND_DIG). A tale scopo le primitive utilizzate per l'invio del messaggio cifrato potrebbero essere adatte e potrebbe non esserci bisogno di scriverne altre. Il digest viene inviato all'host CASTLE determinato in funzione dell'id della controparte. Tiene traccia del tipo di digest.

- Primitive per la ricezione di digest e l'invio al lettore (RECV_DIG). Alla ricezione di

una digest, MOSES deve elaborare la richiesta ed inviarla al lettore. Tiene traccia del tipo di digest.

- Primitive per segnalare la presenza di un digest al lettore (DIG_ALERT). L'host CASTLE alla ricezione di un digest da remoto deve poter segnalare la presenza di tale digest al lettore per chiedere se è pronto a riceverlo. Indica il tipo di digest.

- Primitive per l'invio di un digest al lettore (SEND_DIG_READ). Permettono all'host CASTLE di inviare un digest al lettore. Tengono traccia del tipo di digest (e quindi di messaggio) inviato.

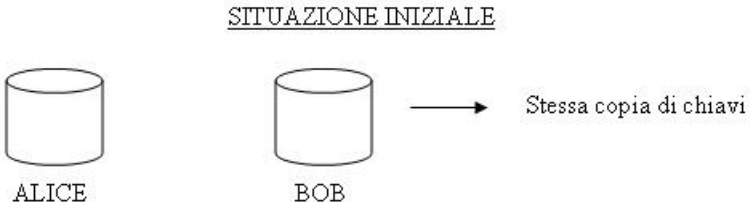
- Primitive per inviare una chiave al lettore di smart card (SEND_KEY). MOSES deve poter inviare al lettore una chiave da salvare nella smart card.

  Un'assunzione da fare in questo caso è che il lettore di smart card non sia malizioso ed esegua il calcolo della funzione di hash in maniera corretta. Definita la struttura hardware ed i vincoli che tale struttura deve rispettare, occorre ora definire il protocollo di identificazione in ogni sua fase, al fine di chiarire lo scopo di tutti i vincoli introdotti finora.

N.B.: Le operazioni di creazione e consegna non devono essere contemporanee ma devono entrambe essere eseguite prima di poter iniziare sessioni di comunicazione tra le due parti.

3.2.4 Protocollo per l'identificazione dell'utente

Fase iniziale



Il protocollo presuppone una fase iniziale realizzata *una tantum* ma necessaria per la corretta realizzazione del protocollo di identificazione.

Più nello specifico, tale fase iniziale richiede le seguenti fasi:

- Realizzazione di una smart card (stesso tipo e caratteristiche) con k bit di chiave di identificazione utilizzando i primi k bit di chiave contenuto nel supporto.
- Eliminazione dei bit di chiave dal supporto (e.g. tramite le primitive in MOSES).
- Consegna della smart card inizializzata ad Alice e Bob (richiede riconoscimento fisico).

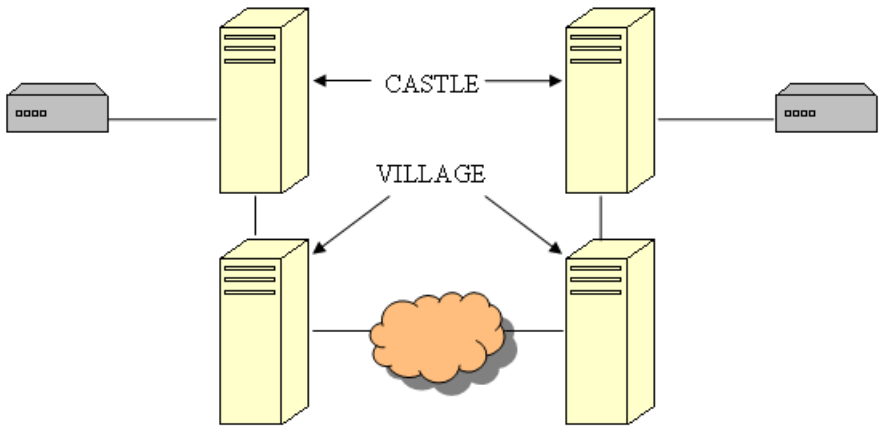


Figura 3.11: Architettura dettagliata del sistema

Fase di identificazione

La fase di identificazione vera e propria si basa su una fase centrale di *attesa* in cui si verifica la presenza e l'identità di entrambi gli utenti che devono comunicare. La definizione di tale fase prende spunto dal protocollo **3-way handshake** poiché permette un facile controllo ed identificazione di eventuali messaggi o pacchetti di comunicazione persi. In questo scenario infatti la perdita di determinati messaggi può comportare la corretta identificazione di

un intruso come legale o la mancata identificazione, viceversa, di un utente legittimo. Si vedranno ora nello specifico, tutte le fasi del processo di identificazione su una struttura hardware distribuita come quella mostrata in precedenza:

1. **Inserimento della smart card nel lettore**. La smart card dell'utente viene inserita nell'apposito lettore.

2. **Indicazione della controparte**. Sull'host CASTLE, l'utente (sia Alice) indica con quale controparte vuole iniziare una sessione (sia Bob).

3. **Attesa**. Prima di identificare correttamente Alice e permetterle di accedere all'host CASTLE, occorre verificare che la controparte indicata da Alice, cioé Bob sia sul corrispondente sistema remoto. Si vuole garantire e realizzare sessioni di conversazione a due, pertanto non si deve permettere l'inizio di una sessione di comunicazione se la controparte non è presente. Per garantire tutto questo, il processo identificativo avviene attraverso le seguenti fasi:

   (a) Il lettore di smart card (disconnesso) legge il valore di chiave di Alice, sia K dalla smart card e l'id della controparte (Bob), sia IDb, entrambe salvate sulla smart card.

   (b) Il lettore di smart card calcola il digest su ASK: DIGEST=DIG_CALC(ASK,K).

   (c) Il lettore di smart card crea una REQUEST con l'identificatore di Bob: REQUEST=REQ_CALC(IDb, DIGEST)

   (d) Il lettore disconnette la smart card.

   (e) Il lettore svuota il buffer usato per calcolare la funzione hash di tutto tranne che il digest.

   (f) Il lettore si connette all'host CASTLE.

(g) Il lettore invia la REQUEST contenente anche il digest, all'host CASTLE: SEND_MESS(REQUEST,ask).

Il digest rimane conservato sul buffer del lettore.

Il lettore si mette in attesa di una risposta. Dopo un periodo di tempo Δt senza ricevere risposte inoltra nuovamente la REQUEST.

Dopo un periodo ΔT (con $\Delta T = n * \Delta t$) senza risposta respinge la richiesta di identificazione di Alice.

(h) L'host CASTLE, tramite MOSES, elabora la REQUEST identificando l'host su cui inviare il digest e lo invia. SEND_DIG(DIGEST,host(IDb),ask).

(i) L'host CASTLE di Bob riceve la richiesta col digest ed ottiene il digest: RECV_DIG(DIGEST,ask).

(j) L'host CASTLE si connette al lettore segnalando la presenza di un digest. Il lettore accetta solo nel momento in cui ha in buffer un digest, cioè un utente in attesa, altrimenti (cioé se sta calcolando un digest o non vi sono digest e quindi utenti in attesa) non risponde e la richiesta resta pendente fino a che il lettore non la accetta. DIG_ALERT(DIGEST,ask).

(k) Quando il lettore ha un digest in buffer (richiesta pendente) accetta il digest presente nell'host CASTLE: DIG_ACCEPT().

(l) All'acettazione, l'host CASTLE invia il digest al lettore: SEND_DIG_READ(DIGEST,ask).

(m) Alla ricezione del digest, il lettore confronta il digest presente nel suo buffer con quello ricevuto e se sono identici esegue i seguenti passi:

   i. Chiude la connessione con l'host CASTLE.

   ii. Svuota il buffer. Perde entrambi i digest che comunque a questo punto non servono più.

iii. Collega nuovamente la smart card.

iv. Legge nuovamente la chiave.

v. Disconnette la smart card.

vi. Crea il digest di un messaggio predefinito ACK: RESP=ACK_CALC(ACK).

vii. Svuota il buffer di eventuali informazioni temporanee, tenendo solo il nuovo digest.

viii. Crea un messaggio di approvazione con l'indirizzo di Alice e vi aggiunge il digest ACC=ACCEPT_CALC(IDa, RESP).

ix. Si connette all'host CASTLE ed vi invia l'accettazione: SEND_MESS(ACC,ack).

x. Ricevuta l'accettazione, l'host CASTLE, tramite MOSES, elabora la ACC identificando l'host su cui inviare il digest di accettazione e lo invia: SEND_DIG(RESP,ho

xi. L'host CASTLE di Alice riceve la richiesta col digest ed ottiene il digest: RECV_DIG(RESP,ack).

xii. L'host CASTLE di Alice segnala al lettore l'arrivo di un digest: DIG_ALERT(RESP,ack).

xiii. Alla ricezione dell'alert sul messaggio di acknowledge, il lettore si disconnette, svuota il buffer, connette la smart card di Alice, legge la chiave, disconnette la smart card e calcola il digest sul messaggio ACK, dopodiché svuota il buffer di tutto tranne che del digest.

xiv. il lettore accetta il digest: DIG_ACCEPT().

xv. L'host CASTLE invia il digest: SEND_DIG_READ(RESP,ack)

xvi. Il lettore confronta i digest, se sono uguali identifica correttamente Alice e aggiorna la chiave sulla smart card (si veda dopo) e la fa accedere all'host CASTLE, altrimenti inoltra un'altra REQUEST.

xvii. Dopo aver inviato l'accettazione, il lettore di Bob resta in attesa per un periodo di tempo pari a T, dopodiché, se non riceve altre REQUEST, assume

che l'accettazione sia stata correttamente ricevuta da Alice (e quindi questa sia stata opportunamente identificata) e **identifica Bob**, *aggiorna la chiave* sulla smart card e lo fa accedere al suo host CASTLE.

(n) Se il confronto dà invece esito negativo, il lettore di Bob invia allo stesso modo un messaggio NACK in modo molto simile al caso precedente per ACK. Per completezza qui di seguito sono indicati i passi specifici, ed evidenziate le differenze rispetto al caso di ACK.

i. Chiude la connessione con l'host CASTLE.

ii. Svuota il buffer.

iii. Collega nuovamente la smart card.

iv. Legge nuovamente la chiave.

v. Disconnette la smart card.

vi. Crea il digest di un messaggio predefinito NACK: RESP=NACK_CALC(NACK).

vii. Svuota il buffer di eventuali informazioni temporanee, tenendo solo il nuovo digest.

viii. Crea un messaggio di approvazione con l'indirizzo di Alice e vi aggiunge il digest: NACC=ACCEPT_CALC(IDa,RESP).

ix. Si connette all'host CASTLE ed vi invia il rifiuto: SEND_MESS(NACC,nack).

x. Ricevuta l'accettazione, l'host CASTLE, tramite MOSES, elabora la ACC identificando l'host su cui inviare il digest di accettazione e lo invia: SEND_DIG(RESP,host(IDa),nack).

xi. L'host CASTLE di Alice riceve la richiesta col digest ed ottiene il digest: RECV_DIG(RESP,nack).

xii. L'host CASTLE di Alice segnala al lettore l'arrivo di un digest: DIG_ALERT(RESP).

xiii. Alla ricezione dell'alert sul messaggio di rifiuto, il lettore si disconnette, svuota il buffer, connette la smart card di Alice, legge la chiave, diconnette la smart card e calcola il digest sul messaggio NACK, dopodiché scuota il buffer di tutto tranne che del digest.

xiv. il lettore accetta il digest: DIG_ACCEPT().

xv. L'host CASTLE invia il digest SEND_DIG_READ(RESP,nack).

xvi. Il lettore confronta i digest, se sono uguali respinge la richiesta di identificazione di Alice, altrimenti inoltra un'altra REQUEST.

xvii. Dopo aver inviato il rifiuto, il lettore di Bob resta in attesa per un periodo di tempo pari a T, dopodiché, se non riceve altre REQUEST, assume che il rifiuto sia stato correttamente ricevuto da Alice (e quindi questa sia stata opportunamente identificata) e respinge la richiesta di identificazione di Bob, impedendogli di accedere al sistema.

4. **Aggiornamento della chiave sulla smart card**. Prima di permettere l'accesso all'utente (sia per Alice che per Bob), il sistema accede al supporto con le chiavi per le sessioni di comunicazione tra Alice e Bob, prende i primi k bit, li salva sulla smart card e li elimina dal supporto. Tale chiave verrà utilizzata alla prossima sessione per l'identificazione. La chiave usata invece per l'attuale autenticazione viene persa, sovrascritta dalla nuova. La presenza di una nuova chiave per ogni sessione fa in modo che un attaccante che riesce a carpire la smart card, copiarla e restituirla all'utente legittimo in modo che non si accorga di nulla, sia comunque passibile di identificazione da parte del sistema e dell'utente. Se si usasse sempre la stessa chiave per tutte le sessioni, infatti, una volta che l'attaccante entra in possesso della smart card può identificarsi come l'utente ogni volta che vuole mentre in questo caso, al momento del collegamento sulla sua smart card e su quella della controparte legale viene inserita

una nuova chiave, mentre su quella dell'utente legale no. Nel momento in cui si cerca di instaurare una nuova sessione tra l'utente legale e la controparte, le due chiavi risultano essere diverse, quindi l'identificazione non va a buon fine ed entrambi hanno la possibilità di rilevare la precedente situazione di intrusione.

5. **Rimozione della smart card e disconnessione dell'utente**. la smart cart card aggiornata viene espulsa, il lettore disconnesso e resettato.

3.2.5 Considerazioni su SCANNER

SCANNER è un sistema di autenticazione costruito in modo tale da garantire l'identificazione di coppie di utenti spazialmente dislocati. In pratica, l'identificazione di Alice avviene correttamente solo previa l'identificazione ed il riconoscimento di Bob, ovvero della controparte con cui Alice, all'atto della connessione al sistema, dichiara di voler comunicare.

L'accesso al sistema ed allo storage viene garantito solo a coppie di utenti che poi comunicano utilizzando ognuno chiavi da due copie dello stesso storage.

Un processo di identificazione di questo tipo non è banale ne tantomeno immediato da realizzare, e richiede di tenere presente molti casi e sottocasi, come si è potuto vedere dalla complessità dell'algoritmo di identificazione.

Ciò nonostante, il vantaggio che comporta in termini di sicurezza è notevole.

SCANNER, infatti, non utilizza un database di utenti legali a cui accedere ogni volta che viene perpetrata una richiesta di login al sistema. Attraverso un database, infatti, l'accettazione di un utente sarebbe molto facile e veloce, in quanto basterebbe verificare la presenza dei dati contenuti sulla smart card con quelli contenuti nel database per un certo utente, e rapidamente si potrebbe determinare la legalità o meno dello stesso.

L'utilizzo di un database tuttavia introduce un nuovo potenziale punto di debolezza nel sistema, nonché un punto di attacco da parte di terze parti maliziose. Garantire una sicurezza totalmente incondizionata ad un database contenente dati così sensibili non è

un problema di facile risoluzione, ed i costi per imbastire un sistema di difesa adeguato potrebbero essere considerevoli.

SCANNER è stato studiato partendo dal presupposto di creare un sistema senza un database di utenti, al fine di prevenire queste situazioni. Di conseguenza, benché l'algoritmo di identificazione di SCANNER sia abbastanza complesso, il costo in termini di complessità è giustificato dai guadagni in termini di sicurezza dell'intero sistema.

L'identificazione di utenti a coppie nasce anche dall'osservazione che, in un sistema dedicato esclusivamente allo scambio di messaggi cifrati tra utenti (senza alcuna altra finalità), non avrebbe alcun senso permettere un accesso al sistema ad un utente, senza che sia presente la controparte per inviare o ricevere messaggi.

Un ruolo fondamentale in SCANNER è esercitato dal lettore di smart card. Come si può notare al lettore sono devolute diversi compiti di calcolo e viene disconnesso e riconnesso continuamente all'host CASTLE.

La scelta di devolvere al lettore la responsabilità di eseguire i conti sul digest è dovuta fondamentalmente a ragioni di sicurezza simili a quelle che hanno spinto a determinare l'architettura del sistema di protezione dello storage. Così come in quel caso si era scelto di utilizzare un'architettura a due livelli (VILLAGE-CASTLE), la stessa scelta viene fatta in questo caso (CASTLE-LETTORE) con il lettore che si disconnette e si resetta opportunamente al fine di evitare attacchi residenti sull'host CASTLE, che potrebbero essere volti a carpire le informazioni sulla smart card.

In teoria, le informazioni presenti sulla smart card sono transitorie, cioè una volta letta la chiave per l'identificazione quella chiave, anche se carpita, non sarebbe utilizzabile, poiché la chiave sulla smart card verrebbe aggiornata nel momento in cui l'utente legale accede al sistema; di conseguenza l'eventuale chiave carpita non sarebbe di alcuna utilità.

Tuttavia, per poter capire il rischio di un attacco di questo tipo, che a prima vista sembra innocuo, occorre ragionare non nell'ottica in cui un utente, Alice, possa comunicare con un

unico interlocutore, Bob, ma bensì un insieme di interlocutori: Bob_1, Bob_2, ..., Bob_n. In uno scenario di questo tipo, verosimilmente molto più somigiante ad uno scenario reale che non il precedente con solo un interlocutore per ogni utente, occorre mantenere più chiavi di identificazione, una per ogni coppia Alice-Bob_i. Per ciò che riguarda la conservazione delle diverse chiavi sulla smart card, qualunque Chip Card con un sistema operativo come MultOS o Java permette la conservazione di dati diversi sulla stessa smart card ed in locazioni separate ed isolate della EEPROM.

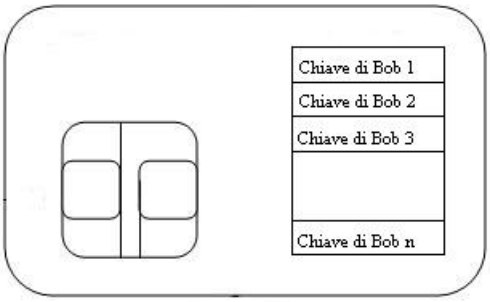


Figura 3.12: Conservazione di più chiavi di identificazione sulla stessa smart card

Il problema di un attacco da parte di codice malizioso sull'host CASTLE a questo punto diviene però reale: in questo caso infatti, gli attacchi da rete potrebbero essere volti a carpire non la chiave attualmente utilizzata da Alice, bensì le altre chiavi presenti sulla smart card e non utilizzate in quel momento.

Supponiamo ad esempio di avere la smart card di Alice che in un certo istante contiene n chiavi. Ad un certo punto Alice decide di logarsi al sistema per comunicare con Bob_1. Se all'atto dell'inserimento nel lettore ci fossero programmi maliziosi, questi potrebbero carpire le chiavi di tutti i Bob_i con $1 < i \leq n$. Tali chiavi, infatti, sono buone per essere copiate su una smart card ed utilizzate per autenticarsi come Alice nelle comunicazioni con i Bob_i con $1 < i \leq n$. L'unica chiave che viene consumata e aggiornata, infatti, è quella di Bob_1,

le altre sono totalmente utilizzabili, ed un utente malizioso potrebbe logarsi ed iniziare una sessione di comunicazione con gli altri Bob senza che il sistema sia in grado di riconoscerlo come intruso, ma riconoscendolo, erroneamente, come Alice.

In un'ottica di questo tipo, quindi, è necessario adottare una politica di connessioni, disconnessioni e reset, anche con il lettore.

Il lettore, a questo punto, acquisisce un'importanza strategica ed esegue operazioni ad hoc, specifiche di SCANNER ed in un certo ordine prestabilito: pertanto, il lettore non può essere un lettore standard di quelli in commercio. Pertanto, potrebbe esser valutata l'ipotesi, in questo caso, di un lettore progettato e costruito appostiamente per SCANNER.

Il tipo di smart card che meglio si adatta al protocollo di SCANNER, invece, è una smart card di tipo CHIP con una EEPROM in grado di memorizzare chiavi crittografiche e di gestirle opportunamente. Pertanto, è chiaro che una normale MEMORY card non è adatta allo scopo perché non permette l'utilizzo di un sistema operativo come MultOS o Java, necessari per poter gestire le chiavi per i diversi utenti che sono presenti sulla smart card.

All'inizio di tutto questo studio, si era preso in considerazione l'ipotesi di poter utilizzare hardware ad hoc, nel momento in cui questo si fosse reso particolarmente necessario ai fini della sicurezza, sebbene sotto l'ipotesi di farne un uso il più possibile ridotto. Nonostante sia stato possibile evitare l'utilizzo di hardware dedicato fino ad ora, in questo caso specifico e per le motivazioni evidenziate, una scelta di questo tipo è fortemente raccomandabile per il lettore di smart card. Uno studio successivo sulla reale realizzabilità di un simile lettore, tenendo conto dei costi e dei tempi di realizzazione, della possibilità di realizzazione e vendita di un lettore di questo tipo, potrebbe evidenziare un'impossibilità od una poca convenienza ad realizzare un hardware dedicato: se così fosse, occorrerebbe rivedere e modificare il protocollo di SCANNER in modo che sia possibile implementarlo utilizzando lettori di smart card standard.

Conclusione

Il lavoro svolto ha permesso di mettere in luce alcuni aspetti e soluzioni interessanti per il problema della protezione di uno storage di dati da attacchi da rete, nel momento in cui questo debba venir collegato ad un host connesso ad Internet. La soluzione proposta, che inizialmente doveva risolvere il problema specifico del sistema QCRYPT di proteggere le chiavi generate ed utilizzate offline, si è rivelata essere valida anche in contesti diversi e più generali.

Più specificamente, i tre sistemi proposti, ATHOS, PORTOS e ARAMIS, possono essere utilizzati in qualunque contesto in cui sia necessario garantire una sicurezza incondizionata sulla rete: la scelta di adottare l'uno o gli altri protocolli dipende da ragioni di costo e di efficienza richieste. A grandi linee quindi, sono state proposte tre architetture che hanno un costo di realizzazione cresente, direttamente proporzionale alle prestazioni, in termini di disponibilità, che possono garantire.

La scelta di concentrarsi sulla sicurezza dagli attacchi da rete dello storage, piuttosto che sulla sicurezza fisica dello stesso, dipende da una considerazione generale sui sistemi destinati ad avere necessità di utilizzare sistemi di crittografia quantistica: l'uso di un sistema quantistico come quello analizzato, con la conseguente necessità di una sicurezza totale, sono appannaggio di realtà militari e governative dove, ovviamente, non si può prescindere dalla sicurezza fisica dei sistemi di elaborazione usati.

Oltre a questa considerazione, la scelta è dipesa dal fatto che, in questi ultimi anni, le protezioni fisiche di storage o dispositivi di backup hanno avuto uno sviluppo notevole,

specie nelle forme di difesa “estreme” dove ogni tentativo di accedere fisicamente allo storage porta all’eliminazione fisica dei dati ed alla loro totale inutilizzabilità. Pertanto è parso lecito supporre che, in qualche modo, la sicurezza fisica dello storage sia garantita in toto dalle istituzioni o dai gruppi che usano gli stessi sistemi di crittografia quantistica.

Lo sviluppo dei sistemi ATHOS, PORTOS ed ARAMIS, hanno portato ad evidenziare la necessità di definire un sistema operativo, MOSES, che eseguisse determinate procedure per sovrintendere le operazioni dei protocolli dei tre sistemi: benchè tale insieme di operazioni sia implementabile in qualunque sistema operativo attuale, il consiglio è di procedere a sviluppare il sistema operativo a sé stante, onde garantire il più possibile la minimalità del sistema stesso. Le soluzioni così proposte si sono rivelate ottimali e garantiscono, se opportunamente implementate, la sicurezza incondizionata del dispositivo di storage, sulla cui natura non sono state fatte assunzioni, tranne alcune piccole ipotesi che sono essenziali per poterne garantire la sicurezza.

Il problema dell’identificazione, invece, è stato studiato nell’ottica di ridurre il più possibile i danni causati dall’errato riconoscimento di un utente malizioso come legale sui sistemi di protezione dello storage elaborati in precedenza. L’idea è stata quella di partire con il determinare, in primis, il metodo più consono per l’identificazione dell’utente (scegliendo quindi le smart card) per poi determinare modifiche ai sistemi ATHOS, PORTOS e ARAMIS, ed al sistema operativo MOSES che potessero consentire di ridurre al minimo l’accesso alle chiavi contenute nello storage. In risposta a questo problema, si è modellato un sistema di riconoscimento, la cui applicabilità esula, anche in questo caso, dal contesto in cui è stato studiato. Tale sistema ha una bassissima percentuale di errore ed una percentuale di perdita di dati sicuri molto bassa, sebbene non si sia giunti a poter garantire una sicurezza incondizionata.

Lo studio realizzato, oltre che al raggiungimento di risultati tecnici, ha permesso di mettere in risalto alcuni spunti di riflessione interessanti in ambito di sicurezza e crittografia.

Molti sistemi, che sono totalmente sicuri in teoria, possono avere problemi in fase di implementazione legati alla fallacità dei sistemi su cui le idee vengono implementate: il problema principale del sistema QCRYPT, ad esempio, sono i polarizzatori che hanno il compito di misurare la polarizzazione dei fotoni entangled. Questi polarizzatori, infatti, richiedono una precisione di misurazione molto alta e pertanto devono essere sempre tenuti sotto costante monitoraggio, pena l'inefficienza del sistema di distribuzione.

Un altro fatto degno di nota è che in fase di implementazione non sempre il sistema riesce ad essere eseguito esattamente come modellato nella teoria, ma si rendono necessarie delle modifiche al flusso esecutivo o ad alcune fasi dei protocolli previsti; a fronte di situazioni di questo tipo occorre modellare quindi soluzioni che consentano di raggiungere, ove possibile, il livello di sicurezza e performance stimato nella fase di teorizzazione: il problema dello storage trattato, ad esempio, si è palesato nella fase di implementazione del QCRYPT, quando si è reso necessario salvare le chiavi generate, vista l'impossibilità di avere sempre online il sistema di generazione delle chiavi.

L'ultima osservazione riguarda la fase di identificazione: tale fase è cruciale in ogni sistema di crittografia nel momento in cui identificare un utente vuol dire permettergli l'accesso a contenuti protetti, se non addirittura al segreto stesso. Pertanto, qualunque sistema sicuro si possa costruire, il suo livello di sicurezza non può prescindere dal livello di affidabilità del sistema di identificazione: affinché la sicurezza del sistema QCRYPT, con gli opportuni sistemi di difesa dello storage, possa dirsi incondizionata a tutti gli effetti, occorre riuscire a determinare un metodo di autenticazione che sia incondizionatamente sicuro, pena la perdita di sicurezza dell'intero sistema.

Glossario di crittografia

ALGORITMO CRITTOGRAFICO Indica un qualunque insieme di regole atte a realizzare un messaggio cifrato a partire da un messaggio in chiaro. Ha bisogno di un parametro detto chiave da utilizzare in combinazione col messaggio in chiaro per ottenere il messaggio cifrato.

CANALE Qualunque mezzo su cui possono viaggiare le informazioni scambiate tra due o più parti. Non deve necessariamente essere un mezzo guidato (filo, fibra ottica, ecc..) ma può essere anche un mezzo non guidato (aria, atmosfera, vuoto e.g. comunicazioni Wireless, onde elettromagnetiche).

CANALE PUBBLICO Canale sul quale le informazioni che vi viaggiano sono accessibili a chiunque si ponga in ascolto su di esso, compresi eventuali attaccanti con intenzioni maliziose.

CANALE QUANTSTICO Canale nel quale valgono le leggi della meccanica quantistica. Un tipico canale quantistico è costituito da fibre ottiche.

CHIAVE Informazione segreta da utilizzare assieme all'algoritmo crittografico per cifrare. Costituisce l'informazione che rende sicuro il testo cifrato se rimane solo appannaggio delle persone destinate a leggerlo.

CIFRARIO Algoritmo crittografico che opera su caratteri o bits, in maniera totalmente indifferente al significato degli stessi, e li sostituisce con altri secondo una legge definita ed in funzione di una informazione segreta.

CIFRARIO ASIMMETRICO Contrariamente al cifrario simmetrico, usa due chiavi distinte

per la fase di cifratura e di decifratura. Si parla pertanto di coppie di chiavi di cui una può venire usata solo per cifrare e l'altra solo per decifrare.

CIFRARIO SIMMETRICO Tipo di cifratura in cui si usa una stessa chiave sia per cifrare che per decifrare. La chiave quindi costituisce il segreto sia per cifrare il messaggio in chiaro che per poter accedere al messaggio cifrato.

CIFRATURA Il processo atto a trasformare un messaggio in chiaro in un messaggio cifrato attraverso un algoritmo crittografico ed una chiave.

CODICI A CORREZIONE DI ERRORE (E.C.C.) Codici in grado di rilevare e correggere errori su una certa stringa di bit. Si basano sulle proprietà algebriche e sulle caratteristiche dei campi finiti. In genere vengono usati per correggere errori dovuti alla trasmissione su un canale. Il processo di correzione ha però richieste di calcolo e tempo notevoli, in maniera proprozionale al numero di errori che si vuole poter rilevare.

CRITTOANALISI scienza che studia i metodi per ottenere un testo in chiaro a partire da un insieme di testi cifrati, senza conoscere la chiave.

CRITTOGRAFIA Scienza che ha per oggetto l'occultamento del significato di un messaggio, in modo tale che questo possa essere recuperato soltanto da chi sia a conoscenza di un opportuno "segreto".

CRITTOGRAFIA QUANTISTICA Branca della crittografia che sfruttando determinate proprietà della meccanica quantistica, permette di realizzare sistemi crittografici incondizionatamente sicuri.

CRITTOGRAMMA (Ciphertext) Il risultato dell'applicazione di un algoritmo crittografico (con relativa chiave) ad un messaggio in chiaro. Il crittogramma è un messaggio senza senso per chi non conosce il mezzo (segreto) per riottenere il messaggio in chiaro.

DECIFRATURA Procedimento opposto alla cifratura, è il processo che permette di estrarre il testo in chiaro da un testo cifrato attraverso un opportuno algoritmo inverso a quello di cifratura ed una chiave.

DISTRIBUZIONE DELLE CHIAVI Processo di condivisione di una chiave segreta tra due parti spazialmente distanti. Lo scopo di questo processo è di fare in modo che la chiave arrivi correttamente alle due parti senza che questa venga carpita da terze parti non autorizzate.

ONE TIME PAD Unico sistema di cifratura incondizionatamente sicuro. Richiede di disporre di una chiave che sia totalmente random lunga quanto il messaggio da cifrare. Se il messaggio in chiaro e la chiave sono due stringhe binarie, il cifrario di Vernam permette di ottenere il messaggio cifrato semplicemente applicando la funzione binaria XOR tra ogni bit del messaggio ed il bit della chiave nella posizione corrispondente.

SICUREZZA INCONDIZIONATA É il livello di sicurezza più elevato. Garantisce che le informazioni segrete restino tali senza vincoli di tempo e di luogo. In pratica garantisce che se un'informazione è protetta in un certo momento, lo sarà necessariamente ed allo stesso modo in tutti i periodi temporali successivi.

STEGANOGRAFIA scienza che si occupa di occultare i messaggi che devono essere mantenuti segreti. A differenza della crittografia, non prevede di proteggere i messaggi rendendoli incomprensibili ma li nasconde in mezzo ad altre informazioni in modo che la presenza di informazioni riservate passi inosservata.

TESTO IN CHIARO - (Plaintext) Il messaggio non occultato che deve essere protetto attraverso un procedimento crittografico.

XOR (Or esclusivo) Funzione logica binaria che restituisce 0 per ogni coppia di bit identici in input, 1 per ogni coppia di bit differenti.

Bibliografia

[1] *Qcrypt, processo di stima del qber*, Tech. report, Elsag s.p.a., 2003.

[2] *Parametri di sicurezza in qcrypt*, Tech. report, Elsag s.p.a., 2004.

[3] Paolo De Nicolo Anna Martinoli, *Qcrypt detailed technical design document*, Tech. report, Elsag s.p.a., 2002.

[4] Sandra Bruzzo Anna Martinoli, Paolo De Nicolo, *System technical design document*, Tech. report, Elsag s.p.a., 2002.

[5] Dan Brown, *Il codice da vinci*, 2003.

[6] Artur Ekert Charles H. Bennet, Gilles Brassard, *Quantum cryptography*, Scientific American (1992), 26–33.

[7] Gilles Brassard Charles H. Bennet, *Quantum cryptography: Public key distributing and coin tossing*, Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, Bangalore, India (10-12 Dicembre 1984), 175–179.

[8] Shari Lawrence Plfeeger Charles P. Plfeeger, *Security in computing*, 1997.

[9] Scott H. Clearwater Colin P.Williams, *Explorations in quantum computing*, 1997.

[10] Caxton C. Foster, *Cryptanalysis for microcomputers*, 1982.

[11] Simson Garfinkel, *Pgp: Pretty good privacy*, 1995.

[12] John E. Hershey, *Cryptography demystified*, 2003.

[13] Dominic Mayers Hitoshi Inamori, Norbet Lutkenhaus, *Unconditional security of practical quantum key distribution*, (17 Giugno 2003).

[14] Doris Baker H.X. Mel, *Cryptography decrypted*, 2001.

[15] Dave Jarvis, *Quantum entanglement illustrated*, (2005).

[16] Samuel J. Lomonaco Jr., *A quick glance at quantum cryptography*, (23 Novembre 1998).

[17] David Kahn, *The codebreakers*, Scribner, 1996.

[18] Fred Piper Kenneth G. Paterson and Rudiger Schack, *Why quantum cryptography?*, (22 Agosto 2004).

[19] Tolga Kilicli, *Smart card howto*, (19 Settembre 2001).

[20] Robert Edward Lewand, *Cryptological mathematics*, 2000.

[21] J. Lawrence Carter Mark N. Wegman, *New hash function and their use in authentication and set equality*, Journal of Computer and System Science (5 Novembre 1980).

[22] Charles H. Bennet Gilles Brassard Claude Crepau Ueli M. Mauer, *Generalized privacy amplification*, IEEE Transactions on Information Theory (6 Novembre 1995), 1915–1924.

[23] Sushil G. Jajodia Neil F. Johnson, Zoran Duric, *Information hiding: Steganography and watermarking - attacks and countermeasures (advances in information security, volume 1*, 2001.

[24] Wolfang Tittel Nicolas Gisin, Gregoire Ribordy and Hugo Zbinden, *Quantum cryptography*, Reviews of Modern Physics (18 Settembre 2001).

[25] Mark Vanderwauver Paul Ashley, *Practical intranet security: Overview of the state of the art and available technologies*, 1999.

[26] Steve Schneider Peter Ryan, *Modelling and analysis of security protocols*, 2000.

[27] Clifford Pickover, *Cryptorunes*, 2000.

[28] Stefano Pirandola, *Introduction to classical and quantum error correction*, (2004).

[29] Edgar Allan Poe, *The gold-bug and other tales*, 1843.

[30] Bruce Schneier, *Applied cryptography second edition protocols, algorthms, and source code in c - second edition*, John Wiley and Sons, Chickester, England, 1996.

[31] Richard J. Hughes D.M. Alde P. Dyer G.G. Luther G.L. Morgan M. Shauer, *Quantum cryptography*, Contemporary Physics (1995), 36–149.

[32] Simon Singh, *Codici & segreti*, Rizzoli, Milano, Ottobre 1999.

[33] Charles H. Bennet Francois Bessette Gilles Brassard Louis Salvail John Smolin, *Experimental quantum cryptography*, Journal of Cryptology (1992).

[34] Paul Stanton, *Securing data in storage: A review of current research.*

[35] Fabien A. Petitcolas Stefan Katzenbeisser, *Information hiding: Techniques for steganography and digital watermarking*, Stefan Katzenbeisser and Fabien A. Petitcolas Editors, 2000.

[36] Neal Stephenson, *Cryptonomicon*, 1999.

[37] Peter Wayner, *Disappearing cryptography: Being and nothingness on the net*, AP Professional, 1996.

[38] Chen Zhiqun, *Java card (tm) technology for smart cards: Architecture and programmer's (the java series)*, Sun Microsystems.

[39] Phillip M. Zimmermann, *Pgp source code and internals*, 1995.