Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea Triennale in Informatica

Studio e realizzazione di strumenti di Crittografia per sistemi Cloud-Storage

Tesi di Laurea in Progetto di Sistemi Virtuali

Relatore:
Chiar.mo Prof.
Renzo Davoli

Presentata da:
Miro Mannino

Sessione
autunnale 2010-11

Indice

Capitolo 1

Introduzione

1.1 Cloud computing e privacy

Come ogni tecnologia, il Cloud Computing nasce da un'**esigenza**. Ogni tecnologia, dopo essere stata introdotta, altera quelli che sono gli usi e costumi della società, sia in modo negativo che in modo positivo e ciò introduce nuovi problemi e nuove esigenze, che lasceranno spazio a nuove tecnologie.

L'evoluzione dei mercati ha portato oggi l'esigenza di far concentrare le aziende al loro core business lasciando poco spazio alla gestione dell'information technology. Inoltre, con l'evoluzione rapida dei mercati, le aziende non riescono più a realizzare grossi investimenti per poterli sfruttare a lungo termine, ma hanno bisogno di una gestione delle risorse più flessibile che possa garantire un abbattimento dei costi. Il cloud computing è in grado di fornire a queste aziende la possibilità di scalare secondo le proprie esigenze la quantità di risorse, pagando quindi il loro utilizzo effettivo. Permette la possibilità di aumentare in modo considerevole il numero di risorse in brevissimo tempo per riuscire a far fronte a tutte quelle situazioni in cui fenomeni virali su internet riescono a fare aumentare in modo esponenziale il numero di utenti.

Tuttavia il cloud computing introduce nuovi problemi, poiché l'affidamento dei propri dati a terze parti potrebbe non garantire la sicurezza necessaria. Per i privati è possibile infatti che chi mantiene i dati possa utilizzarli per condurre delle **ricerche di mercato** e **profilare gli utenti**. Nel caso di aziende questi tipi di problemi di privacy possono anche degenerare in casi di **spionaggio industriale**. Le società di cloud computing molto spesso poi risiedono in stati diversi da quello in cui si trova l'utilizzatore e in questi casi bisogna prestare attenzione alle leggi a cui sono sottoposti. A seguito di riflessioni importanti condotte da importanti figure come Richard Stallman[1] e Neelie Kroes[2], si focalizza sempre più l'attenzione sulla sicurezza del cloud computing e si cerca di fornire strumenti sempre più avanzati e trasparenti che la garantiscano.

1.2 Scopo del progetto

Utilizzando un sistema operativo dove "everything is a file", è molto utile conoscere dei meccanismi che permettano di virtualizzare singoli sottoalberi del file system in modo da utilizzare i file come risultato di svariate operazioni, queste poi possono anche non coinvolgere i dispositivi di archiviazione reali. FUSE[3] è un progetto molto famoso che mette a disposizione una libreria, con la quale è possibile implementare un proprio file system senza dover lavorare a livello kernel. In questo modo è possibile implementare un file system specifico, in grado di compiere operazioni particolari che vengono eseguite in user mode. La realizzazione di un file system con FUSE è poi relativamente semplice rispetto a doverlo implementare lato kernel.

Nel mese di Aprile del 2011, Christopher Soghoian[4] scrive un articolo che prende di mira uno dei servizi più famosi nel campo del cloud storage, Dropbox. Soghoian spiega la necessità di tutelarsi da questo servizio, poichè questo non protegge i dati mantenuti nei loro server con sistemi crittografici. I modi con il quale è possibile proteggersi sono molti ed alcuni vengono anche suggeriti dagli stessi fornitori come nel caso di Dropbox, che consiglia l'uso di EncFS[5].

EncFS, utilizzando FUSE, realizza una crittografia dei dati in modo trasparente. Il metodo di funzionamento è molto semplice: la cartella che deve essere protetta, che chiameremo RootFolder, viene montata su un altra cartella, che chiameremo MountFolder. Tutte le operazioni che vengono svolte sulla cartella MountFolder vengono eseguite sulla RootFolder, in questo modo il contenuto della MountFolder è quello della RootFolder, ma con alcune differenze. In EncFS le operazioni di write e read compiono anche rispettivamente una cifratura e de-cifratura del contenuto, quindi nella RootFolder i file sono memorizzati in maniera cifrata, mentre nella MountFolder è possibile leggere il contenuto di questi file in chiaro.

Visto l'interesse verso FUSE e verso la crittografia si è pensato di realizzare un progetto che proteggesse in modo trasparente l'utente da questi servizi di cloud storage utilizzando FUSE. Per andare incontro alle esigenze

dei servizi offerti da un servizio di cloud storage come Dropbox bisognava che il progetto diventasse più complesso rispetto ad altri progetti come EncFS. Dropbox, infatti, mette a disposizione la possibilità di condividere delle cartelle con più utenti, per montare la RootFolder sulla MountFolder è necessario quindi un meccanismo più complesso che utilizzasse chiavi pubbliche e private. Da questo il nome: pubcFS. Con tale meccanismo si riescono ad eliminare tutti quei svantaggi legati alla comunicazione della password.

PubcFS utilizza un algoritmo crittografico a chiave simmetrica per cifrare il contenuto e i nomi dei file. L'algoritmo utilizzato per cifrare è AES-256 in CFB mode. In questo modo la cifratura dei contenuti, che deve essere abbastanza performante, non subisce delle limitazioni. La chiave simmetrica, che chiameremo Key, viene memorizzata in maniera cifrata in una particolare cartella di configurazione della RootFolder, a questo scopo viene utilizzata una chiave pubblica di uno specifico utente. Il risultato è quindi che la chiave simmetrica viene memorizzata molte volte ed ogni volta in modo diverso, poiché ogni copia è decifrabile solamente dall'utente che possiede la chiave privata corrispondente. L'utente, che poi vuole montare la RootFolder nella MountFolder, deve quindi esibire la propria chiave privata in modo che pubcFS possa decifrare la chiave Key. Ogni utente può chiedere, utilizzando la propria chiave privata, di garantire l'accesso ad un altro utente specificando la chiave pubblica di quest'ultimo. Come algoritmo crittografico per la gestione delle chiavi è stato utilizzato RSA, gli utenti possono utilizzare le proprie chiavi RSA in formato PEM per poter essere utilizzate con pubcFS. PubcFS riesce anche a gestire tutte quelle chiavi private che vengono protette da password e riesce a gestire anche chiavi di dimensioni diverse.

PubcFS come EncFS riesce a cifrare i nomi dei file. Questi vengono cifrati sempre utilizzando AES-256 e utilizzando la chiave Key, esattamente come per il contenuto. Per fare in modo che poi sia possibile nominare i file con il testo cifrato, questo viene trasformato in base64url[6]. Tale base è molto simile a quella base64 con l'unica differenza che viene resa compatibile per gli URL. A differenza di EncFS, per default i nomi dei file non vengono cifrati,

questo perché servizi come Dropbox notificano il cambiamento dei file con dei messaggi ed è scomodo, nel caso che questi abbiano un nome cifrato, non riuscire a capire quelli che cambiano. Per fare in modo che il nome del file venga cifrato basta rinominare il file aggiungendo il prefisso "enc_".

Capitolo 2

Contesto scientifico e tecnologico

2.1 Cloud Computing

2.1.1 Nascita e motivazioni

Come ogni tecnologia, il Cloud Computing nasce da un'**esigenza**. Ogni tecnologia dopo essere stata introdotta altera quelli che sono gli usi e costumi della società, sia in modo negativo che in modo positivo. In questo modo nasceranno nuovi problemi, nuove esigenze e quindi nuove tecnologie.

Il Cloud Computing nasce come soluzione ad un problema che oggi ogni azienda deve affrontare: l'**Information Tecnology.** Essa consiste nel trattare l'informazione mediante l'uso di apparecchiature digitali e sistemi software. Con la **globalizzazione** i mercati hanno avuto un'evoluzione notevole; un'azienda sempre più ha smesso di prendersi cura delle poche persone che vivono nei dintorni. Deve infatti prendersi cura di un numero notevole di clienti e naturalmente questo prevede una mole di lavoro che difficilmente è affrontabile senza l'aiuto di automazioni. Per questo motivo ci si affida sempre più ai sistemi software e hardware che riescono a farsene carico.

Simone Brunozzi (Technology Evangelist per Amazon Web Services) spiega [7] proprio come oggi l'IT sia una necessità ed è ormai diventato un costo, le aziende quindi sono spinte a trovare un modo per limitare questo costo con l'uso di nuove tecnologie come il **Cloud Computing**. Quello che interessa ad un'azienda è riuscire ad ottenere da questi sistemi un **servizio** che possa aiutarla nello svolgimento del lavoro. Non gli interessa come questo servizio venga svolto, ma che venga svolto in tempi ragionevoli. Uno dei grandi problemi di un'azienda è proprio quello di doversi interessare, in base alle previsioni di mercato, alle **necessità del sistema**. Dovendo quindi ragionare su quanta potenza di calcolo serve, sulla quantità di archiviazione e su una serie di altri fattori che fanno perdere tempo distogliendo l'attenzione dai problemi reali.

In una situazione tradizionale si fa un **grosso investimento**, che coprirà quello che prevede essere la mole di lavoro richiesta. In un primo momento, il numero di clienti non è sufficiente per sfruttare a pieno le potenzialità

del sistema, avendo quindi una perdita. In un secondo momento, invece, un cambio improvviso nei mercati potrebbe far schizzare il numero oltre la capacità consentita dallo stesso sistema, avendo una **perdita di clienti**. Può poi succedere anche che il numero dei clienti, dopo un calo degli stessi, ancora una volta non sfrutta a pieno le capacità del sistema. Oltre al numero di clienti esistono situazioni in cui si necessità di moltissima **potenza di calcolo per brevi periodi** e rimane ancora una volta una perdita fare un investimento enorme per essere sfruttato solo per brevi periodi. E' difficile quindi che la capacità del sistema rimanga sempre di pari passo con la mole di lavoro richiesta, l'ideale sarebbe che essa si modelli in base a quelle che sono le esigenze attuali.

Il Cloud Computing nasce proprio per questo motivo: l'hardware da comprare e montare fisicamente negli edifici della propria azienda diventa "**virtuale**" ed è subito a disposizione di chi vuole utilizzarlo, solamente pagando quello che vuole utilizzare. Altre aziende, che forniscono servizi di cloud computing, hanno a disposizione una quantità molto elevata di macchine e queste le affittano a chi ne richiede il loro utilizzo. In questo modo la capacità di calcolo diventa **on demand**, capace di adattarsi all'attuale mole di lavoro. In questo modo i costi dell'infrastruttura, che servono per far funzionare il sistema software, diventano **flessibili** e si adattano a quello che è il reale utilizzo e quindi l'attuale guadagno.

In questo modo le aziende per aprire non devono fare grossi investimenti e sono avvantaggiate poiché, per avviare l'azienda, potrebbero non avere subito tutto il denaro necessario. Le aziende, che hanno improvvise crescite della propria quota di mercato, possono sfruttare al meglio tutti i clienti chiedendo al fornitore di questi servizi un incremento delle macchine da utilizzare, le aziende che si ritrovano invece una grossa perdita della propria quota di mercato non mandano in fumo grossi investimenti, poiché possono ancora una volta chiedere al fornitore di decrementare il numero di macchine utilizzate.

Oltre a sfruttare a pieno le capacità del sistema bisogna anche tenere conto di tutti quegli aspetti tecnologici molto costosi che sono necessari in

certi ambienti. Uno dei temi che infatti l'azienda deve affrontare è la **gestione del rischio**: hanno bisogno di proteggere dei dati importanti da perdite accidentali, hanno bisogno di poter distribuire dei dati in giro per il mondo o hanno bisogno di tante altre cose particolari che naturalmente necessitano di molto denaro per essere implementate. Se pensiamo solamente al proteggere i dati da perdite accidentali, bisogna tenere conto dei disastri naturali, bisogna quindi distribuire i propri dati in diversi posti anche distanti tra loro. Questi posti, dei veri e propri data center, dovranno comunicare fra di loro per permettere la loro sincronizzazione. Ovviamente mettere in piedi un'infrastruttura di questo tipo richiede moltissimo denaro. Amazon fornisce questo tipo di servizi e naturalmente riesce a fornirli a costi ragionevoli, poiché si è precedentemente attrezzata con queste infrastrutture molto costose facendolo in larga scala.

Un altro aspetto da tener conto è l'interruzione del servizio che causa delle perdite economiche notevoli, riuscire ad ottenere un servizio affidabile richiede delle buone competenze che spesso non sono presenti in aziende che si occupano di ben altre cose.

L'azienda, per tutti questi motivi, dovrebbe solamente concentrarsi investendo sul **core business** senza preoccuparsi di trovare anche buoni tecnici che possano gestire queste situazioni particolari. Il concetto, che sta alla base del cloud computing, in sostanza si spiega anche con l'esempio della corrente elettrica: "produrre la corrente elettrica in ogni casa costa di più di far produrre la corrente elettrica da una grande centrale, poiché quest'ultima lo fa in larga scala". [7]

2.1.2 Concetti principali

Il cloud computing consiste in un insieme di tecnologie, che permettono l'accesso a risorse per memorizzare ed elaborare dati direttamente usando la rete, la memorizzazione e l'elaborazione di tali dati quindi avviene in luoghi spesso distanti dall'utilizzatore e la rete diventa un elemento essenziale per l'utilizzo di questi servizi e deve essere affidabile e veloce. Le aziende

utilizzano il cloud computing affidando a terze parti alcune attività, che in precedenza venivano svolte dalla stessa, questo affidamento viene chiamato **outsourcing**. L'azienda, che affida a terze parti le proprie attività, ha naturalmente dei cambi di strategia nella loro gestione passando da un controllo diretto ad un controllo indiretto, che viene concordato con il fornitore di questi servizi tramite dei contratti. Come già detto in precedenza i motivi che portano un'azienda a fare outsourcing sono essenzialmente: **riduzione dei costi**, investimento sul **core business**. Esistono quindi altre aziende che forniscono questi servizi di cloud computing, che vengono comunemente chiamate **Cloud Provider**.

Una prima distinzione che si ha è la differenza fra **Public Cloud** e **Private Cloud**:

- Il private cloud è un cloud che è interno all'azienda.
- Il public cloud è fatto in modo da essere disponibile ad altre aziende che comprano questi servizi di cloud computing.

Da un punto di vista hardware il cloud computing crea numerosi vantaggi:

- L'idea di avere **infinite macchine, che è possibile utilizzare** semplicemente richiedendole al provider che fornisce i servizi di cloud computing. In questo modo si eliminano tutte le previsioni di utilizzo del sistema che vengono fatte quando si vuole fare un grosso investimento per comprare le macchine, che servono per far funzionare un determinato servizio.
- L'azienda può **iniziare con un numero ridotto di macchine** e incrementarlo secondo le esigenze, semplicemente richiedendole al provider che fornisce i servizi di cloud computing.
- La possibilità di avere un servizio dove si **paga per utilizzo**, utile per le aziende a cui servono tante macchine per brevi periodi.

Con l'uso del cloud computing gli utenti finali possono accedere ai propri dati o lavorare con i propri servizi ovunque e quando si vuole. I provider

di questi servizi semplificano agli estremi il modo con cui si può accedere in remoto a questi servizi.

Esistono varie tipologie di Cloud Computing:

- **(SaaS) Software as Service**: consiste nell'utilizzo di programmi in remoto, spesso attraverso un server web.
- **(PaaS) Platform as a Service**: simile al precedente, ma, invece che uno o più programmi singoli, viene eseguita in remoto una piattaforma software, che può essere costituita da diversi servizi e programmi.
- **(IaaS) Infrastructure as a Service**: utilizzo di risorse hardware in remoto. Questo tipo di Cloud è quasi un sinonimo di **Grid Computing**[1], ma con la caratteristica che le risorse vengono utilizzate **on demand**, cioè al momento in cui un cliente ne ha bisogno.
- **(DaaS) Data Storage as a Service:** consiste nella conservazione remota di dati, è come il precedente, dove la risorsa che viene virtualizzata e remotizzata è uno spazio di archiviazione per la conservazione di dati.

L'architettura si basa su una serie di server reali che forniscono il servizio. Tali server vengono configurati e gestiti dall'azienda che vuole fornire questo tipo di servizi, che sicuramente sarà più qualificata di un'azienda che invece si occupa anche di altro. Si espongono delle interfacce che elencano e gestiscono i servizi offerti. Alcuni clienti (**Clienti Finali**) utilizzano il servizio precedentemente configurato da altri clienti (**Clienti Amministratori**). Il cliente amministratore utilizza le interfacce messe a disposizione per selezionare il servizio richiesto e per amministrarlo configurandolo, attivandolo o disattivandolo. Le caratteristiche fisiche dell'implementazione del server reale sono irrilevanti ai fini del servizio stesso.

[1] I sistemi Grid sono un'infrastruttura di calcolo distribuito che nasce dall'esigenza di elaborare grandi quantitativi di dati. Si utilizza una vasta quantità di risorse che vengono utilizzate mediante una rete che le interconnette.

Il cloud computing fa nascere anche nuove opportunità di lavoro, poiché necessita di Cloud Provider. Oggi sono in molti che hanno già iniziato a guadagnare facendo i cloud provider, compagnie importanti come Amazon, Google, eBay, Microsoft, ed altre ancora. Il cloud provider fa decisamente molti soldi, poiché dopo un considerevole investimento che serve per costruire tutta la struttura, ha un ritorno economico notevole, poiché ci sono molti utenti che richiedono l'uso di questi servizi e quindi si riesce, facendo economia di scala, a recuperare tutti i soldi e guadagnarne molti altri ancora.

Con il Web 1.0 per accettare carte di credito sconosciute bisognava sottoscrivere un contratto con servizi come VeriSign o Authorize.net che perduravano per molto tempo. Con l'avvento di PayPal, invece, si paga in ogni transizione e questo cambia il modello di business in uno chiamato **pay-as-you-go.** Amazon Web Services ha infatti introdotto questo tipo di modello pay-as-you-go senza contratti; tutto quello che serve al cliente è una carta di credito. [8][9]

2.1.3 Cloud Storage

Il cloud storage è un tipo di servizio offerto da chi si occupa di cloud computing. Questo servizio prevede la **conservazione di alcuni dati** su uno spazio riservato su un server. Questo server può essere dedicato e quindi **riservato alla sola rete** della sede alla quale appartiene, oppure può essere un **server virtuale ospitato** presso altre strutture di aziende che si occupano proprio di cloud computing.

Chi si occupa di hosting, avendo a che fare con data center molto grandi, possono benissimo fornire questi tipi di servizi vendendo spazio di archiviazione. Tale spazio di archiviazione è spesso **virtuale** poiché viene distribuito su più server, questo per ottimizzare lo spazio che si può mettere a disposizione. In questo modo l'utente può anche richiedere ulteriore spazio o cederlo secondo le reali esigenze, in questo modo si pagherà solamente quello che realmente viene utilizzato. Inoltre, virtualizzando e quindi astraendo lo spazio a disposizione si possono avere notevoli vantaggi nella varietà di servizi

che possono essere offerti al cliente: lo spazio diventa flessibile e quindi la dimensione riservata al singolo cliente può variare senza problemi nel tempo, l'astrazione dello spazio di archiviazione può essere sfruttato per creare file system virtuali, che memorizzano file, o per creare altri tipi di storage virtuali per la memorizzazione di database o altro ancora.

Per conservare i dati vengono utilizzati protocolli standard come **iSCSI** per fare in modo che lo spazio venga virtualizzato a livello molto basso, vengono anche utilizzati protocolli più ad alto livello come CIFS/NFS o WebDAV per una condivisione a livello di file. Per manipolare i dati vengono utilizzate delle API che ne definiscono l'interfaccia; le operazioni di manipolazione dei dati possono ovviamente essere fatte in modo remoto.

I dati che vengono conservati presso queste strutture, chiamate data center, sono molto protetti da eventuali perdite accidentali. Spesso queste strutture sono distribuite in varie zone anche molto lontane fra loro, tenute sincronizzate, creando dei veri e propri duplicati per impedire la perdita degli stessi.

Ci sono oggi tante aziende che si occupano di cloud storage, i servizi più famosi sono:

- **Dropbox**: `http://www.dropbox.com`
- **Amazon S3**: `http://aws.amazon.com/s3/`
- **Ubuntu One**: `http://one.ubuntu.com`
- **4shared**: `http://www.4shared.com`
- **DivShare**: `http://www.divshare.com`
- **ADrive**: `http://www.adrive.com`

Uno dei grandi problemi del cloud storage è la privacy. I dati sono mantenuti presso strutture estranee al cliente che li possiede, che sempre il rischio che il fornitore di questi servizi di storage può utilizzare questi dati contro la volontà del cliente finale. [10]

2.1.4 Amazon

Simone Brunozzi durante la conferenza[7] spiega come Amazon fornisce questi servizi di cloud computing, ne spiega i vantaggi dell'uso, la sicurezza che Amazon mette a disposizione e le aziende, che usando queste tecnologie, hanno avuto successo.

Brunozzi spiega come questi servizi devono essere facili da usare, non bisogna leggere due manuali prima di poter utilizzarli. Amazon ha un focus enorme sulla sicurezza, questo perché possiede dati molto confidenziali come oltre 90 milioni di carte di credito. Grazie ad una struttura molto controllata, tecnologicamente sofisticata, Amazon ha garantito, per tutti questi anni di operato, una sicurezza assoluta ai propri clienti e questo gli ha fatto guadagnare talmente tanta fiducia da essere un punto di riferimento per le altre aziende che operano nello stesso settore.

Sottolinea anche come un'azienda, per mettere in piedi un'infrastruttura molto potente, deve spendere moltissimi soldi e successivamente di come questi investimenti non sono sufficienti o sono stati troppi per le reali esigenze. Con Amazon chiunque, munito di una carta di credito, può andare su Amazon e cominciare ad utilizzare centinaia di server senza l'obbligo di utilizzo e il pagamento per tempi prolungati, successivamente si può smettere di utilizzarli smettendo di pagare.

Amazon ha molti anni di esperienza nel design, costruzione e manutenzione di grossi data center. I data center sono in strutture anonime, le strutture critiche poi sono ben controllate e il perimetro controllato a livello militare. L'accesso fisico è ben controllato da uno staff di sicurezza utilizzando video sorveglianza, sistemi di controllo di intrusioni ed altri strumenti elettronici. Gli impiegati autorizzati devono autenticarsi almeno tre volte prima di raggiungere il piano dedicato al data center e i visitatori invece vengono continuamente scortati per non permettere che visitatori possano introdursi furtivamente. Gli impiegati non vengono sempre autorizzati ad entrare nella zona dei data center, infatti essi vengono autorizzati solamente nel caso che ci sia un interesse specifico, una volta che questo interesse cessa di esistere

il permesso viene revocato, lo stesso succede per i dipendenti che cessano i rapporti con Amazon. [7] [11]

Amazon Simple Storage Service S3

Amazon offre un servizio di Cloud Storage chiamato Amazon S3, i dati vengono salvati in data center molto protetti, i dati vengono duplicati e distribuiti su più data center collocati in zone distanti per prevenire la perdita dei dati in seguito a guasti o in seguito ad una catastrofe naturale. Amazon S3 può memorizzare oggetti che possono avere una dimensione da 1B fino a 5TB. Il numero di **oggetti** che è possibile memorizzare è illimitato. Ogni oggetto viene memorizzato in un contenitore chiamato **bucket**, accessibile mediante un'unica chiave assegnata allo sviluppatore. I dati possono essere memorizzati in zone anche molto distanti dall'utilizzatore, più ci si allontana da queste zone e più la velocità con la quale è possibile accedere a questi dati diminuisce. Ogni bucket può infatti essere memorizzato in una **Region** particolare, in modo da ottimizzare la latenza, minimizzare i costi ed altro ancora. Le regioni disponibili oggi sono: US Standard, EU (Ireland), US West (Northern California), Asia Pacific (Singapore) e Asia Pacific (Tokyo).

Amazon fornisce poi anche uno strumento molto potente chiamato **Amazon CloudFront**, con il quale in pochi minuti è possibile creare una distribuzione che farà in modo di portare alcuni dati che interessano in determinate zone. Amazon possiede infatti numerosi data center che fungono da cache, che possono essere utilizzati per questo scopo. In questo modo i dati possono essere reperibili con una velocità eccellente perché più vicini a quelli dell'utilizzatore.

Per prevenire accessi non autorizzati ai dati memorizzati, Amazon definisce sempre chi, come e quando un impiegato può avere l'accesso ad essi. Per assicurarsi della sicurezza del sistema le API di Amazon S3 forniscono una protezione che permette l'accesso solamente ai creatori del bucket o dell'oggetto. I permessi di scrittura e di cancellazione sono controllati mediante un **Access Control List** (ACL) associata al bucket. Il permesso di modifica-

re l'ACL associata al bucket è a sua volta protetto mediante un'altra ACL accessibile generalmente solo dal creatore.

I dati vengono protetti anche nei casi in cui possono essere intercettati durante un trasferimentofra i vari nodi di Amazon. Per una sicurezza massima viene utilizzato **SSL**. Per una sicurezza massima Amazon S3 è accessibile da postazioni cifrate via SSL, queste postazioni sono accessibili via Internet con Amazon EC2, in questo modo si ha la sicurezza che i dati siano al sicuro garantendo la sicurezza sia all'interno della rete di Amazon che fuori.

Amazon specifica anche come i dati memorizzati in S3 vengono cifrati e rimangono tali in tutta la rete interna, consiglia poi che per un ulteriore sicurezza si può sempre cifrare i dati prima di essere inviati in Amazon S3 da parte dell'utente stesso. Per questo motivo Amazon mette a disposizione **AWS SDK for Java** che fornisce un semplice meccanismo per la cifratura e decifratura dei dati lato client. In questo modo i dati arrivano ai server di Amazon S3 già cifrati e nessuno può decifrarli senza la chiave privata utilizzata dal client. Naturalmente è necessario non perdere tale chiave, poiché non si potrebbe più accedere al contenuto. Per utilizzare tali funzionalità esiste una classe chiamata AmazonS3EncryptionClient che implementa la stessa interfaccia della classe che viene utilizzata in modo standard che si chiama AmazonS3Client, in questo modo l'applicazione precedentemente implementata potrà utilizzare S3 cifrando e decifrando i dati lato client senza grossi cambiamenti.

Quando un oggetto viene eliminato da Amazon S3 la mappatura di questo con l'oggetto fisico viene eliminato in tutta la rete in cui questo viene distribuito in pochi secondi e diviene quindi inaccessibile, fisicamente però i dati vengono rimossi solamente a seguito di nuove scritture come avviene in generale nei sistemi di uso comune. [7][11][12]

Amazon EC2

Esiste anche Amazon EC2, che fornisce dei server virtuali on demand, si può scegliere fra diverse dimensioni del server. Una volta attiva la macchina

è possibile caricare qualsiasi sistema operativo e avendo i privilegi di root si avrà un accesso totale alla macchina. Uno storage permanente, chiamato Elastic Block Store, permette la memorizzazione di dati come ad esempio database. Si possono creare delle zone, queste isolano i server di quella particolare zona dagli altri, in questo modo è possibile costruire un servizio ad alta affidabilità, dove se qualche macchina ha un problema, il servizio continua comunque a funzionare. Viene fornito anche un IP statico chiamato Elastic IP che permette alle macchine del cliente, tramite i loro DNS, di puntare alle macchine utilizzate. [7]

Amazon Elastic MapReduce

Amazon Elastic MapReduce permette di eseguire un calcolo distribuito senza particolari difficoltà, si può scegliere su quante macchine si vuole eseguire il calcolo, con quali dati e con quale algoritmo e in pochi minuti quindi è possibile far cominciare il calcolo. Di solito vengono utilizzati strumenti differenti per il calcolo distribuito che richiedono un elaborata configurazione e particolari conoscenze. In questo modo si rende semplice l'utilizzo di questi strumenti. [7]

Casi di successo

Brunozzi conclude parlando di casi di successo di aziende che con Amazon sono riusciti a fare successo. Animoto è un azienda che ha fatto un applicazione web che prende tante immagini e ne costruisce un filmato con vari effetti. Quest'applicazione, che prima era poco utilizzata, necessitava di meno di 80 server su Amazon; una volta messa su Facebook ha subito un incremento esponenziale delle visite e in tre giorni ha incrementato i server a più di 3500. L'azienda adesso ha continuato ad incrementare le proprie visite avendo anche dei picchi di 8000 server. Senza l'uso di servizi come Amazon, per riuscire ad avere questo tipo di incremento, bisognava investire milioni di euro comprando questi server e dopo qualche mese riuscire finalmente ad utilizzarli, in questo modo si sarebbero persi molti clienti, poiché avrebbero

trovato un servizio mal funzionante. Dopo un investimento di milioni di euro questi server, una volta finito l'effetto virale di questa applicazione, avrebbero smesso di essere utilizzati, rendendo inutile l'investimento fatto. [7]

2.1.5 Dropbox

Dropbox è uno dei servizi di cloud storage fra i più conosciuti attualmente. Le sue prestazioni e la sua facilità nell'utilizzo sicuramente hanno contribuito oggi a rendere questo prodotto così famoso. [13]

Caratteristiche principali

Mette a disposizione gratis uno spazio di 2GB oppure un quantitativo maggiore pagando. Il costo dell'utilizzo però non viene calcolato per lo spazio effettivamente utilizzato ma per lo spazio che viene riservato. Esistono infatti poche fasce ed ognuna con un prezzo diverso, per quantitativi maggiori di 100GB viene specificato di contattare il settore delle vendite di Dropbox e vengono fatte offerte personalizzate in base alle proprie esigenze.

Questo spazio messo a disposizione può essere gestito mediante un'interfaccia web dalla quale è possibile aggiungere e rimuovere file. Dropbox mette anche a disposizione diversi client per tanti sistemi operativi che permettono l'accesso a questo spazio: Windows, Mac OS, Linux, iOS, Android e BlackBerry. I client più famosi sono sicuramente quelli che è possibile installare su Windows, Linux e Mac OS. Questi client creano una cartella sulla propria home directory e tale cartella rimane sincronizzata con tutto il contenuto salvato sul server. Un demone installato si occuperà di inviare tutti i cambiamenti locali al server e di controllare e rendere disponibile i cambiamenti remoti. I dati rimangono comunque salvati sul computer dell'utilizzatore e quindi disponibili anche quando si è offline. Quando il computer è online il client sincronizza i dati. Questa sincronizzazione, che può durare anche molto tempo, può essere regolata in modo da non occupare troppa banda, e quindi non essere troppo invasiva.

Dropbox è anche molto famoso per la sua velocità, infatti quando si hanno dei file molto grossi, che vengono cambiati solamente in piccole parti, **durante la sincronizzazione vengono mandate solamente le parti che differiscono**. Per questo motivo Dropbox è in grado anche di salvare le variazioni che il file ha subito nel tempo, quindi è possibile recuperare il file come si presentava molto tempo prima, realizzando anche un semplicissimo servizio di versioning. Per mantenere la coerenza Dropbox non elimina irrimediabilmente i file che vengono eliminati, ma lascia sempre la possibilità di poter essere ripristinati utilizzando però l'interfaccia web, dove è anche possibile eliminare del tutto i file cancellati.[14]

Sharing

Nello spazio messo a disposizione c'è una cartella chiamata Public, i file che sono contenuti dentro tale cartella sono pubblici e quindi accessibili da chiunque. Per accedere a tali file comunque bisogna possedere il link associato al file, questo può essere ottenuto mediante l'interfaccia web. I client inoltre mettono a disposizione la possibilità di ottenere tale link semplicemente cliccando sul tasto destro sul file[2]. Dropbox inoltre mette a disposizione la possibilità di **condividere delle cartelle con altre persone**, mediante l'interfaccia web è possibile invitare altri utenti semplicemente inserendo il loro nome utente e quindi ovviamente anche accettare gli inviti degli altri utenti. Le cartelle condivise con gli altri utenti sono accessibili solamente dalle persone che sono state abilitate, i cambiamenti a tali cartelle sono visibili in modo istantaneo da tutti gli altri utenti. Lo spazio occupato dai file, che sono contenuti dentro le cartelle condivise, viene diviso fra gli utenti che posseggono tale cartella. Al creatore di un file viene attribuito tutto lo spazio per quel file, mentre agli altri utenti viene associato lo spazio delle porzioni

[2]per fare questo Dropbox ha bisogno di interagire con il menù contestuale del sistema operativo, e questa interazione varia radicalmente in base al sistema operativo, per i sistemi operativi open source infatti questa interazione viene fornita con un modulo separato e il codice di quest'ultimo è anch'esso open source.

che sono state modificate. Una cartella chiamata "Photos" invece può essere utilizzata per mettere a disposizione pubblicamente una galleria di immagini visibile a tutti.[14]

Sicurezza

Dropbox evidenzia come tiene molto alla sicurezza dei dati mediante l'utilizzo dei migliori metodi e tools disponibili attualmente, un gruppo di persone si occupano della sicurezza di questa parte del software cercando di mantenerli sicuri. I dati sono tenuti in maniera sicura, mediante l'utilizzo di sistemi crittografici, inoltre vengono svolti dei backup per garantire che i dati non vengano persi a seguito di guasti. Viene utilizzato AES a 256 bit per la crittografia degli spazi di archiviazione del server, mentre viene utilizzato SSL (Secure Sockets Layer) per la crittografia che garantisce la sicurezza dei dati scambiati via internet durante la sincronizzazione.

Dropbox specifica anche che i propri impiegati non hanno il permesso di vedere il contenuto dei file salvati dagli utenti, ma che hanno solamente il permesso di poter vedere i nomi di tali file e i metadata associati. Con rare eccezioni un gruppo ristretto di impiegati è abilitato all'accesso dei dati per ragioni di privacy policy, queste eccezioni sono solitamente giustificate dagli obblighi verso la legislazione vigente e Dropbox, come altre compagnie che svolgono servizi di cloud computing, non è sicuramente un'eccezione in questo senso. Inoltre vengono anche impiegate una serie di misure aggiuntive elettroniche e fisiche che impediscono l'accesso non autorizzato ai dati.

Dropbox usa Amazon Simple Storage Service (S3) come spazio di archiviazione. Quest'ultimo, come già descritto prima, ha delle robuste politiche di sicurezza.[14]

Privacy Policy

Quando si registra un account Dropbox memorizza una serie di informazioni personali come il nome, numero telefonico ed altre informazioni, opzionalmente, se si richiedono dei servizi avanzati anche il numero di carta

di credito e indirizzo di fatturazione ed altro ancora. Altre informazioni personali possono essere quelle derivate dall'importazione dei contatti da altri servizi come quello di posta elettronica. Queste informazioni personali possono essere usate per il corretto funzionamento del servizio o comunque per un futuro miglioramento del servizio.

Quando si utilizza il servizio di Dropbox vengono memorizzate tutte le informazioni riguardanti il proprio dispositivo: indirizzo IP (Internet Protocol), il tipo di browser, le pagine web visitate prima di visitare quella dell'interfaccia web (solo nel caso in cui si stia utilizzando l'interfaccia web), operatore telefonico associato al dispositivo. Inoltre viene memorizzato un log con tutte le attività che sono state svolte, tutti i metadata relativi ai file che sono stati memorizzati nel servizio. Nei dispositivi provvisti di strumenti di geo-localizzazione (come il GPS) vengono memorizzate anche le informazioni sul luogo dove si trova il dispositivo, Dropbox specifica come queste informazioni non vengono attualmente utilizzate, ma che in futuro potranno esserlo per migliorare il servizio. In assenza di strumenti di geo-localizzazione viene approssimato il luogo utilizzando l'indirizzo IP.

Il contenuto dei file o le informazioni registrate possono essere utilizzate per ragioni di necessità a seguito di richieste legali, per proteggere la sicurezza di qualunque persona e per prevenire casi di uso fraudolento del servizio offerto da Dropbox. I file, che vengono messi a disposizione delle autorità che ne hanno richiesto l'utilizzo, vengono decifrati. Dropbox specifica però che non riesce a decifrare i file che sono stati precedentemente cifrati da parte dell'utente e questi vengono messi a disposizione delle autorità così come sono. Per questo motivo i file comunque non vengono cancellati subito dai server, vengono mantenuti dei back-up per un periodo più o meno lungo, lo stesso vale nel caso l'account su Dropbox venga rimosso. [15]

2.1.6 Problemi legati al Cloud Computing

Il cloud computing, visto che consiste essenzialmente anche nell'affidare i propri dati a terze parti, è rischioso. L'utilizzo dei servizi di cloud compu-

ting per la memorizzazione di dati personali da parte di privati è infatti un problema per la privacy. E' possibile che chi mantiene i dati possa utilizzarli per condurre delle **ricerche di mercato** e **profilare gli utenti** utilizzando proprio il contenuto dei dati che vengono memorizzati in remoto. Nel caso di aziende questi tipi di problemi di privacy possono anche degenerare in casi di **spionaggio industriale**, infatti le aziende lavorano per lo più con dati che gestiscono il loro core business e quindi tutte quelle informazioni sensibili alla stessa come progetti, prototipi ed altro ancora.

Le società, che offrono servizi di cloud computing, molto spesso hanno server localizzati in diversi stati e le **server farm** che mantengono i dati spesso risiedono in stati diversi da quello dell'utente e sono quindi sottoposti a leggi diverse, di conseguenza sfuggono dai controlli al quale ingenuamente pensiamo che siano sottoposti. Bisogna sempre essere informati su come questi dati vengono distribuiti. Si fa riferimento a Amazon S3 che mantiene sempre i dati all'interno della Zona indicata, in questo modo sappiamo con certezza dove risiedono i dati.

Richard Stallman afferma[1] che il concetto di Cloud Computing è solamente una stupida moda, promossa da chi ha interesse a danneggiare gli utenti. Il fatto di conservare online le proprie fotografie, email, contatti ed anche i propri lavori significherebbe affidare queste informazioni, a volte sensibili, a grosse società come Google, Microsoft, Amazon ed altre ancora. Questo significherebbe rinunciare completamente al controllo dei propri dati, che dovrebbero essere salvati sui propri dispositivi di memorizzazione e poter essere utilizzati con applicazioni libere e gratuite. Infatti si ribadisce il fatto che l'uso di applicazioni non libere ti lascia senza difese e quindi alla mercè di chi ha realizzato quel software, il cloud computing è quindi esattamente come utilizzare software proprietario. Con il cloud computing si ridefiniscono delle cose che già si facevano precedentemente, impacchettandole in una più allegra versione in stile Web 2.0. Continua dicendo che ci sono molti sostenitori del cloud computing che credono che l'evoluzione verso questa direzione era inevitabile, ma che queste persone solitamente sono quelle che

poi promuovono soluzioni cloud.

Neelie Kroes, vice presidente della Commissione Europea ed anche Commissario per l'Agenda Digitale, sottolinea[2] come il mondo di Internet si sia evoluto in modo da diventare uno strumento utile e meraviglioso, ma che il successo di questa evoluzione dipende dal fatto che non c'è stato un eccessivo accanimento legislativo che ne ha regolato l'uso. In questo senso però Internet potrebbe anche degenerare allontanando gli utenti e gli investitori, che oggi sono così tanto contenti per mancanza di fiducia. Spiega che un intervento legislativo non dovrebbe puntare sul controllo della rete, ma dovrebbe assicurare l'uguaglianza delle parti. Nel caso, ad esempio, ci sia una violazione dei dati presenti in un determinato servizio di cloud computing, non ci sono regole precise che tutelano l'utente. L'autoregolamentazione che è così diffusa nel mondo di Internet difficilmente tutelerà l'utente nel caso di un torto da parte di una società che fornisce tali servizi. Il problema della privacy quindi nel mondo del cloud è essenzialmente un **problema di tipo giuridico**, oltre che un problema di tipo tecnologico e gestionale (**privacy by design**). Spiega anche che l'evoluzione in questo campo è molto sostenuta e per questo motivo bisognerebbe, in tempi molto rapidi, colmare questa mancanza giuridica. Ancora oggi però non c'è una definizione universale e precisa di cosa si intenda per "cloud" e questo rende ciò ancora più difficile. Sottolinea anche come ci sia il rischio che l'applicazione di norme volte a proteggere la privacy del cittadino possano essere aggirate grazie all'architettura di questi sistemi, volutamente progettata in questo modo, per essere controllabile il meno possibile. Naturalmente queste norme dovrebbero in qualche modo essere internazionali, e non dipendenti dai singoli paesi, solo in questo modo può essere possibile controllare tutte quelle compagnie che portano i dati in giro per il mondo per sfuggire ai controlli.

Negli Stati Uniti, dopo le leggi anti-terrorismo del 2011, che prendono il nome di **Patriot Act**[16], le società, che forniscono un servizio di cloud computing, sono tenute a svelare i dati degli utenti in caso si verifichino dei sospetti. I Servizi segreti come CIA e FBI hanno accesso illimitato ai da-

ti degli utenti, non è richiesto nessun consenso da parte di qualche giudice. Inoltre le società non sono autorizzate ad informare il cliente che i suoi contenuti vengono controllati dalle autorità, tutto quindi avviene segretamente e senza nessuna limitazione. Con il Patriot Act le società, che hanno sede negli Stati Uniti, sono tenute lo stesso a fornire questi dati anche se sono memorizzati in altri paesi. Durante il procedimento contro WikiLeaks infatti le autorità hanno controllato i dati, che erano stati memorizzati su server di Amazon. Tuttavia comunque è difficile che le autorità si mettano a spiare i dati di utenti privati, che ovviamente non hanno molto di interessante.

Anche a seguito di queste riflessioni, alcune forse anche un po' esagerate, si capisce come bisogna comunque **rimanere attenti** a questo nuovo fenomeno. Alcuni servizi come Gmail oggi non sono più solo una moda, ma servizi molto evoluti che offrono grandi potenzialità e che in questo modo compensano in parte il fatto di non essere open source. Inoltre è sicuramente impensabile sfruttare queste nuove tecnologie solo dopo un attento intervento legislativo e tecnologico. Aspettare un intervento di tipo legislativo imporrebbe infatti di dover attendere la definizione di standard e la definizione di leggi che dovrebbero essere condivise dalla maggior parte dei paesi. Aspettare un intervento di questo tipo provocherebbe un rallentamento, se non addirittura un blocco, all'evoluzione di queste tecnologie. Per questo bisognerebbe che si conducesse una campagna di informazione mediante internet e i media tradizionali, per mettere a conoscenza gli utenti dei reali rischi sulla privacy. In questo modo l'utente consapevole può in qualche modo prevenire in modo opportuno. [2]

Prevenzione

Bisogna quindi che i consumatori prendano coscienza che ci sono dei rischi nell'uso di queste tecnologie e che quindi è necessario comportarsi di conseguenza per tutelarsi. I fornitori, solitamente, propongono dei contratti "as is", che **non garantiscono determinati livelli di performance**. Bi-

sognerebbe, invece, tutelarsi facendo in modo che i contratti garantissero un livello minimo del servizio secondo determinati parametri tecnici misurabili.

Per quanto riguarda la privacy i fornitori di servizi cloud computing dovrebbero garantire l'uso di determinate tecnologie, che garantiscono la **riservatezza dei dati**. Inoltre dei controlli da parte dello stato dovrebbero poter verificare che queste tecnologie vengano realmente applicate. Bisogna quindi stare attenti quando i dati vengono conservati in luoghi stranieri dove potrebbero non esserci più controlli di questo tipo. Per questo motivo bisogna che il fornitore del servizio indichi in **quale stato vengono conservati i dati** e se in quel determinato stato eventuali controversie, che possano nascere, siano risolvibili con un giudice dello stesso stato, piuttosto che uno straniero, questo per fare in modo che i costi della controversia non diventino inaccessibili. Inoltre è bene assicurarsi che in tali stati sia possibile ottenere l'esecuzione dei provvedimenti ottenuti dal giudice.

Inoltre è bene sempre verificare che la proprietà dei dati, una volta che vengono trasferiti in uno di questi servizi di cloud computing, rimanga all'utente con apposite clausole di riservatezza e **rivendicazione dei diritti di "proprietà intellettuale"**(o come sarebbe più proprio chiamarli eccezioni al diritto di libertà intellettuale). Devono anche essere indicate le modalità di **restituzione dei dati** stessi al momento della cessazione del contratto.

L'esame attento di questi fattori appena descritti possono limitare oggi gli inconvenienti del cloud computing. In futuro sicuramente ci saranno sempre più leggi specifiche che tuteleranno di più l'utente che fa uso di questi servizi, ma bisogna oggi essere assolutamente consapevoli dei rischi e di conseguenza tutelarsi in modo opportuno.[17]

2.2 Crittografia

2.2.1 Introduzione

La crittografia nasce nell'antichità dall'esigenza di nascondere ai nemici messaggi strategici. Uno dei primi sistemi crittografici che si conosce è stato il **cifrario di Cesare**, oggi ritenuto sicuramente più che obsoleto.

Una primo principio fondamentale che riguarda qualsiasi sistema crittografico viene scritto da Kerckhoffs, tale principio viene comunemente chiamato Legge di Kerckhoffs[3]. Questo principio dice che la sicurezza di un sistema crittografico non dipende dal tenere nascosto l'algoritmo, ma solamente dal tenere nascosta la chiave. Questo è un principio che si ritiene fondamentale, poiché motivo di molti ragionamenti erronei anche nelle persone comuni e in situazioni anche del tutto estranee all'informatica stessa.

Un altro principio fondamentale nella crittografia consiste nel sapere che un algoritmo perfetto, cioè totalmente sicuro, utilizza una chiave lunga quanto l'intero messaggio con ovviamente il grande svantaggio di renderlo poco usabile. Questo principio viene dimostrato da Claude Shannon in seguito ad una proposta avanzata da Gilbert Vernam nel 1918.

Oggi infatti la ricerca nel campo della crittografia si occupa di trovare algoritmi sicuri ed efficienti, che non comportano l'utilizzo di chiavi lunghe quanto il messaggio.[18]

Crittografia simmetrica

La chiave utilizzata per cifrare è identica a quella utilizzate per decifrare. Questa chiave viene comunemente chiamata **chiave privata**, poichè deve essere tenuta nascosta. Uno degli algoritmi storicamente più famosi è il **DES** (Data Encryption Standard), sviluppato dall'IBM degli anni '70. Successivamente questo algoritmo diventa obsoleto, poichè con l'aumentare della velocità degli elaboratori la dimensione della chiave (56 bit) diventa molto piccola e per questo si riescono facilmente a svolgere attacchi di tipo

[3] si trova nel suo libro "La Cryptographie Militaire" del 1883

brute force in breve tempo. Oggi si utilizzano infatti algoritmi come **Triple DES** (si utilizza DES reiterandolo tre volte), **AES** (Avanced Encryption Standard).

Crittografia asimmetrica

Si tratta di un sistema crittografico più recente, consiste nell'avere più di una semplice chiave. Una coppia di chiavi comunemente chiamate **chiave pubblica** e **chiave privata**, vengono utilizzate per cifrare e decifrare il messaggio secondo particolari proprietà. Se si utilizza la chiave pubblica per cifrare il messaggio occorrerà la chiave privata per decifrarlo e viceversa. Questo permette la certezza che solamente le due chiavi possano comunicare fra loro. La chiave pubblica è fatta in modo che non si riesca a risalire a quella privata, e questa proprietà viene ottenuta grazie all'uso di funzioni unidirezionali comunemente conosciute come **funzioni one-way**. Con la chiave privata invece si riesce a risalire a quella pubblica, ed è per questo che deve rimanere segreta. In realtà la sicurezza dei sistemi di crittografia a chiave asimmetrica risiede nell'impossibilità di risalire alla chiave privata conoscendo quella pubblica in un tempo accettabile, attualmente esistono algoritmi che riescono a compiere questa operazione in tempi molto lunghi, tuttavia non è dimostrato matematicamente che questa operazione non si possa fare in breve tempo.

La coppia di chiavi viene generata tramite un algoritmo come **RSA** o **DSA** a partire da numeri casuali. La chiave pubblica deve essere il più possibile a conoscenza di tutti, può essere inserita in appositi spazi chiamati **keyserver**. Più la chiave pubblica è famosa e più il sistema diventa sicuro, un possibile attaccante infatti potrebbe distribuire in giro una sua chiave pubblica spacciandola per quella di un altro. In questo modo il mittente potrebbe cifrare un messaggio utilizzando la chiave pubblica sbagliata e lasciando quindi la possibilità, da parte dell'attaccante, di poter leggere il messaggio. Successivamente l'attaccante può anche cifrare il messaggio utiliz-

zando la chiave pubblica, che sa essere quella originale e spedire il messaggio al reale destinatario, in questo modo nessuno si accorgerà dell'inganno.

Le chiavi possono essere usate per realizzare una **comunicazione confidenziale**: il messaggio viene cifrato con la chiave pubblica del destinatario e successivamente solamente il destinatario può leggere tale messaggio, poichè è l'unico possessore della chiave privata corrispondente.

Inoltre possono essere usate come **firma digitale**: si realizza un fingerprint del messaggio utilizzando funzioni hash come **MD5** o **SHA1** e tale risultato viene cifrato utilizzando la chiave privata del mittente. Il fingerprint cifrato è la firma del messaggio. Il destinatario successivamente realizzerà il fingerprint del messaggio e lo confronterà con quello del mittente dopo averlo decifrato. Il destinatario può quindi avere la certezza che il messaggio non sia stato alterato e che sia stato inviato dal quel particolare mittente.

Motivi dell'introduzione della crittografia asimmetrica

La crittografia asimmetrica è stata introdotta per risolvere molti dei problemi legati ai sistemi crittografici a chiave privata, ma nonostante questo ne introduce degli altri. Per questo motivo i sistemi crittografici a chiave asimmetrica non sono da ritenersi un evoluzione rispetto a quei sistemi a chiave simmetrica, ma semplicemente un'alternativa da utilizzare per tanti casi che hanno bisogno di necessità particolari.

Utilizzando dei sistemi simmetrici la sicurezza di tutto il meccanismo risiede sul fatto che la chiave sia mantenuta segreta, se l'attaccante infatti possiede la chiave privata, nonostante i due si scambino messaggi cifrati, essi potranno essere sempre decifrati rendendo inutile l'uso del sistema crittografico. I due interlocutori inoltre devono preventivamente concordare la chiave che utilizzeranno per cifrare i messaggi, questa chiave dovrà essere scambiata ancora utilizzando un canale che li mette in comunicazione. Tale scambio può essere intercettato dall'attaccante e di conseguenza in questo modo c'è il rischio che la chiave possa essere scoperta. Può essere utilizzato un altro canale in modo che sia più difficile, da parte dell'attaccante, trovare la corri-

spondenza fra il messaggio che transita da un canale e la chiave concordata che è transitata da un altro canale, ma questo comunque non rende il sistema sicuro. In situazioni dove lo scambio di messaggi avviene in continuazione, si necessita sempre di un protocollo per lo scambio della chiave, difficilmente tale protocollo cambia sempre in modo così drastico da diventare imprevedibile, l'attaccante quindi potrebbe essere a conoscenza dei modi di procedere dei due interlocutori e di conseguenza sapere con certezza come avviene lo scambio della chiave. Inoltre bisogna ricordare che questi meccanismi sono comunemente automatizzati e quindi è più che ovvio che l'attaccante sappia già il modo di procedere. Il canale dove transita la chiave dovrebbe essere quindi inattaccabile, ma questo in generale non è garantito. La crittografia asimmetrica elimina appunto la fase in cui i due interlocutori devono concordare la chiave, il mittente infatti sa già la chiave con la quale deve cifrare il messaggio ed è sicuro che solamente il destinatario potrà decifrarlo. [18]

2.2.2 AES

L'Advanced Encryption Standard (AES) è stato inventato da i due crittografi **Joan Daemen** e **Vincent Rijmen**, originariamente chiamato **Rijndael** dall'unione degli inventori. I due inventori hanno presentato al processo di selezione il loro algoritmo chiamato Rijndael e questo è stato adottato dalla National Institute of Standards and Technology (NIST) nel 2001, AES è attualmente un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America e che ha sostituito il vecchio DES (Data Encryption Standard). [19]

AES può essere sviluppato via software che via hardware, è semplice da implementare e risulta poi essere molto veloce e utilizza poca memoria. AES utilizza chiavi che possono essere di 128, 192 o 256 bit. Quando si parla di AES-128, AES-192 ed AES-256 ci si riferisce alla dimensione della chiave che si sta utilizzando. La National Security Agency (NSA) utilizza AES per proteggere documenti confidenziali, viene utilizzata una chiave di 128 bit per i documenti ritenuti SECRET e di 192 o 256 bit per quelli TOP SECRET.

La dimensione del blocco invece è fissa a 128 bit, mentre l'algoritmo originale Rijndael può lavorare anche con blocchi di dimensione diversa, da un minimo di 128 ad un massimo di 256 bit.

L'algoritmo AES consiste in una serie di **ripetizioni** che trasformano l'input in chiaro nell'output cifrato. Ogni ripetizione consiste in diversi **step** tra cui alcuni di essi che dipendono dalla chiave. Si opera utilizzando matrici di 4×4 byte chiamate **State**, l'input viene copiato dentro lo state, elaborato e successivamente ricopiato nell'output.

Per prima cosa viene calcolata la **round key**, essa viene calcolata utilizzando il **Rijndael's key schedule**. Quest'ultima operazione viene chiamata KeyExpansion e dipende strettamente dalla chiave di cifratura.

All'inizio si effettua l'operazione di **AddRoundKey** dove ogni byte dello state viene combinato con la **round key** mediante un operazione di XOR**.**

Successivamente alle ripetizioni seguenti si eseguono questi step:

1. SubBytes
2. ShiftRows
3. MixColumns
4. AddRoundKey

Infine viene svolto un'ultima ripetizione finale che consiste nell'eseguire questi step:

1. SubBytes
2. ShiftRows
3. AddRoundKey

AddRoundKey step

Consiste nell'unione dello state con la round key utilizzando l'operazione di XOR.

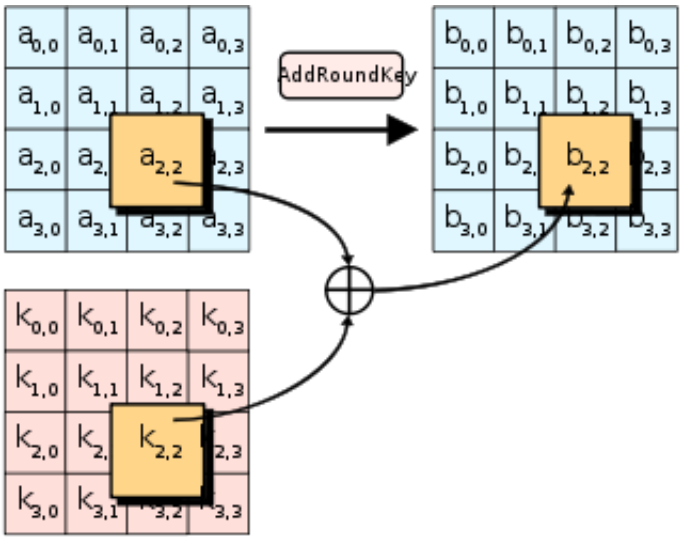


Figura 2.1: AddRoundKey Step (image from: wikipedia.org)

SubBytes step

Ogni byte dello state viene modificato utilizzando una funzione di sostituzione chiamata Rijndael S-box[4]. Con questa operazione si fa in modo che l'algoritmo risulti non lineare, poiché questa S-box viene calcolata dall'inversa nel campo finito di $GF(2^8)$ famosa per essere non lineare[5]. Inoltre per

[4]Le S-Box vengono utilizzate per rimuovere le relazioni tra il messaggio in chiaro e il testo cifrato seguendo il principio della confusione enunciato da Shannon

[5]Per maggiori informazioni consultare informazioni riguardanti il Campo di Galois

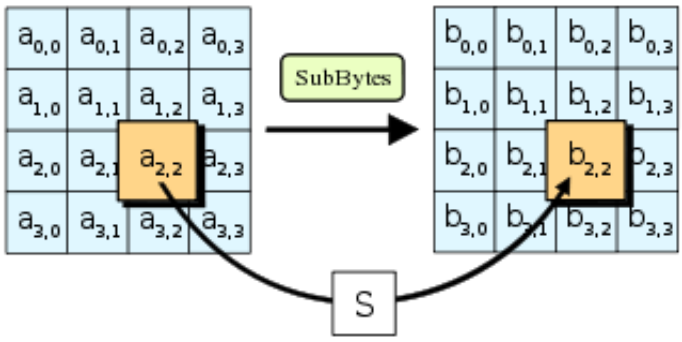


Figura 2.2: SubBytes Step (image from: wikipedia.org)

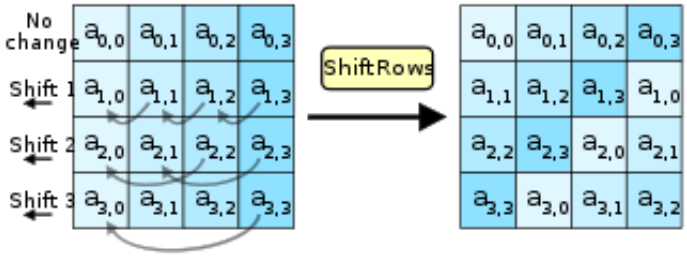


Figura 2.3: ShiftRows Step (image from: wikipedia.org)

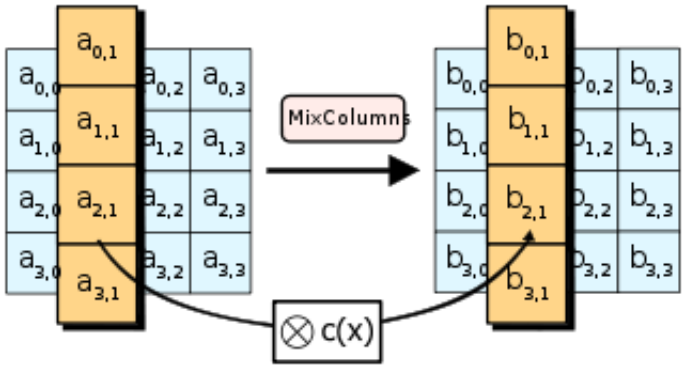


Figura 2.4: MixColumns Step (image from: wikipedia.org)

evitare attacchi basati sulle proprietà algebriche di S-box, quest'ultima viene costruita con cura in modo che non abbia punti fissi e punti opposti. La S-box di Rijndael visto che lavora con dei byte ovviamente è una S-box a 8 bit. Il risultato $b_{i,j}$ viene calcolato come $S(a_{i,j})$.

ShiftRows step

Viene fatta l'operazione di shift su ogni riga, il numero di volte che si compie quest'operazione dipende dal numero della riga. Questo vuol dire che la prima riga resta invariata, alla seconda riga viene fatto un solo shift, alla terza due e alla quarta tre. In questo modo si ottiene che l'ultima colonna dello state di ingresso formerà la diagonale dello state di uscita.

MixColoumns step

Si prendono i quattro byte di ogni colonna e si combinano utilizzando una trasformazione lineare. Tale trasformazione lineare per essere utilizzata anche per decifrare un messaggio ovviamene deve essere invertibile.

Utilizzati insieme gli step ShiftRows e MixColumns fanno in modo che l'algoritmo rispetti il criterio di confusione e diffusione dell'algoritmo.

2.2.3 Block cipher modes

Le modalità di funzionamento degli algoritmi crittografici a blocchi consistono nello specificare come può essere cifrata una sequenza di byte. Infatti tali algoritmi specificano come può essere cifrato un particolare blocco e non come può essere cifrato una sequenza arbitraria di dati. Le prime specifiche di block chiper modes sono state definite in principio dalla National Institute of Standards and Technology (NIST) per poter far funzionare l'algoritmo DES in differenti situazioni, infatti il documento pubblicato che le descrive viene intitolato "DES modes of operations"[20]. Successivamente, con la nascita di nuove esigenze sono state introdotte altre modalità di funzionamento, che vengono successivamente raccolte nel documento Special Publication 800-38A [21].

Electronic CodeBook (ECB)

Viene comunemente utilizzato per la trasmissione sicura di valori singoli, per questo motivo è la più semplice. Il testo in chiaro viene suddiviso in blocchi di 64 bit ed ognuno di essi viene cifrato utilizzando la stessa chiave. Se il messaggio ha una dimensione che non è un multiplo di 64, l'ultimo blocco dovrà essere riempito opportunamente. Con questa modalità di funzionamento blocchi diversi con lo stesso contenuto hanno lo stesso risultato, poiché vengono cifrati utilizzando la stessa chiave. Per questo motivo viene utilizzato per dati di dimensione ridotta, poiché si potrebbero sfruttare le regolarità del messaggio per analisi crittografiche.

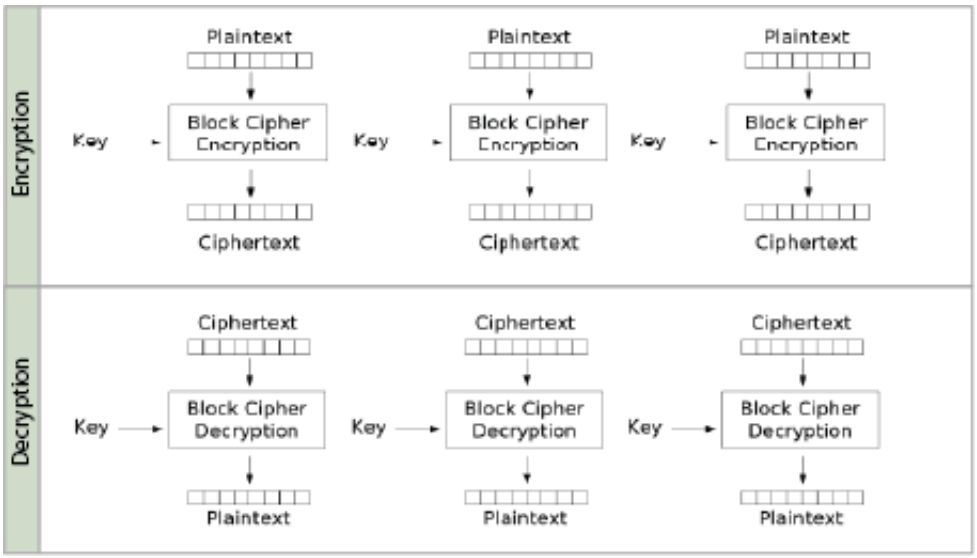


Figura 2.5: ECB Mode (image from: wikipedia.org)

Cipher Block Chaining (CBC)

Nato per risolvere i problemi di sicurezza di ECB, infatti con questa modalità di funzionamento due blocchi con lo stesso contenuto producono due risultati diversi. Questo perché la modalità CBC cifra il blocco tenendo conto anche del blocco precedente, in particolare il blocco in chiaro che deve essere passato come input viene modificato facendo lo XOR con il blocco cifrato precedente. Ogni blocco viene cifrato poi con la stessa chiave. Per il primo blocco viene fatto lo XOR con il vettore di inizializzazione chiamato IV. L'IV viene utilizzato quindi anche in fase di decifratura e quindi deve essere conosciuto dal destinatario che decifrerà il messaggio.

Cipher Feedback (CFB)

Questa modalità è nata per convertire un algoritmo a blocchi in uno a flusso. Ciò vuol dire che non necessita di eseguire dei riempimenti nell'ultimo blocco e può essere utilizzato per cifrare flussi senza dover ragionare a blocchi. La modalità CFB necessita del solito vettore di inizializzazione IV. Necessita anche di un parametro **s** che può essere 1, 8, 64 o 128; si parla di CFB-1

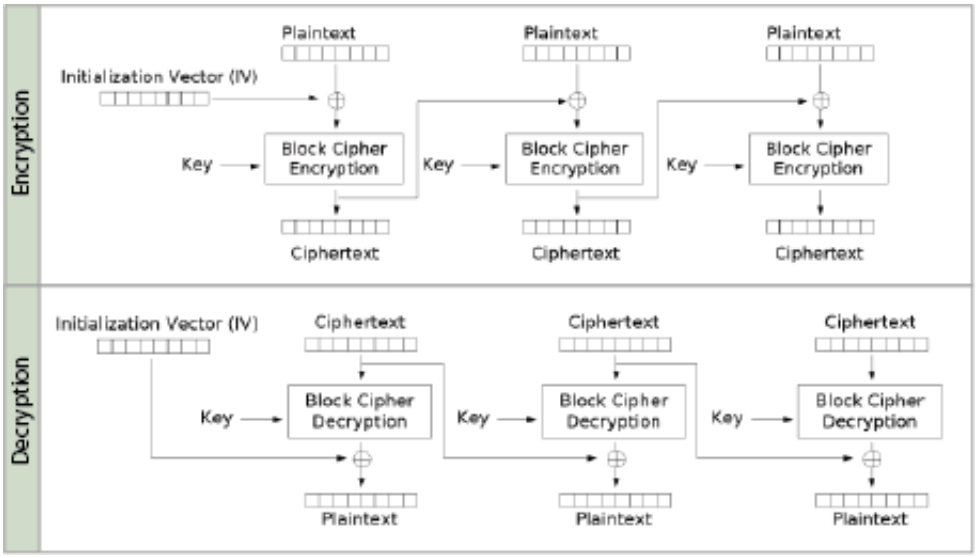


Figura 2.6: CBC Mode (image from: wikipedia.org)

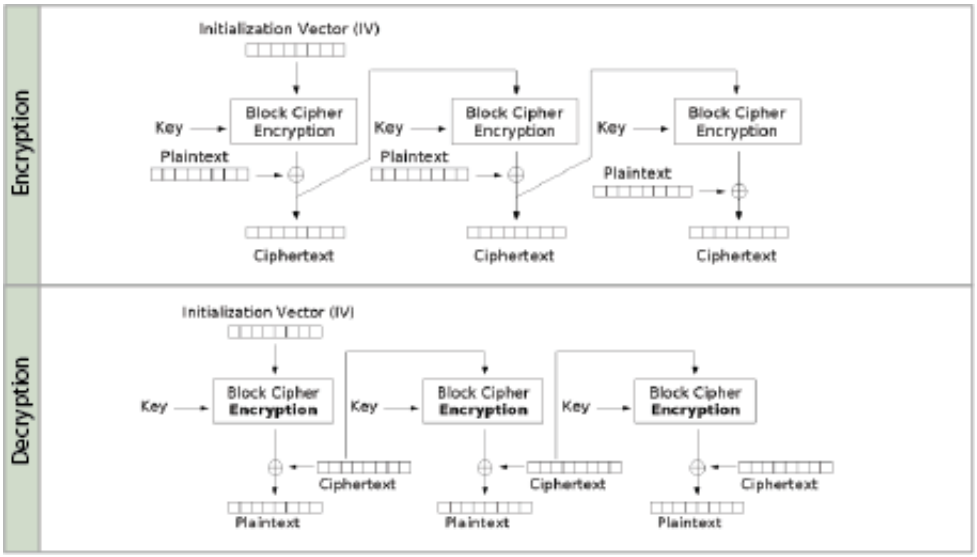


Figura 2.7: CFB Mode (image from: wikipedia.org)

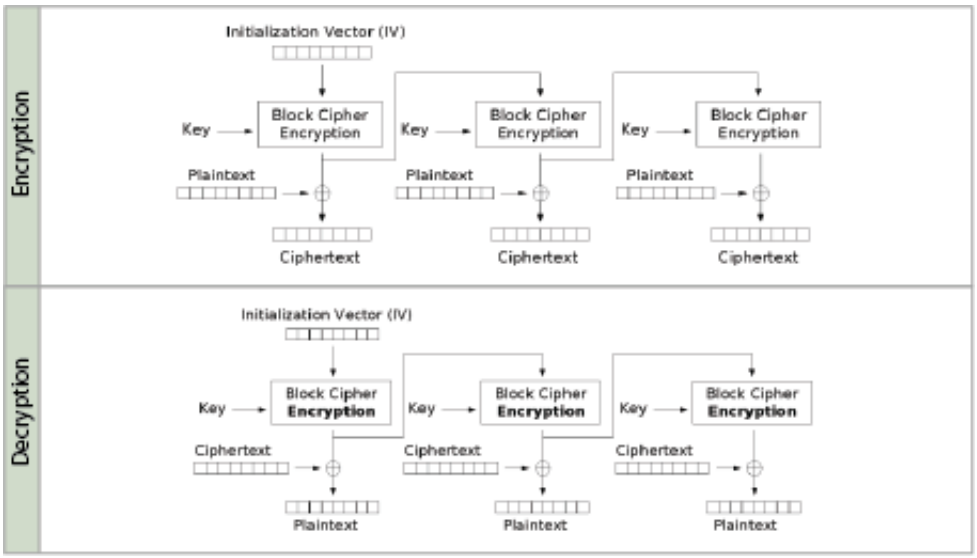


Figura 2.8: OFB Mode (image from: wikipedia.org)

quando $s = 1$, CFB-8 quando $s = 8$, CFB-64 quando $s = 64$ e CFB-128 quando $s = 128$.

Il primo blocco di input è il vettore IV, tale blocco lo chiameremo I1, le prossime operazioni sono applicate a tale vettore per produrre il primo blocco di output. I1 viene cifrato ottenendo un primo risultato intermedio che viene modificato facendo lo XOR fra gli s bit più significativi di tale risultato e i primi s bit del testo in chiaro. Il risultato dello XOR, che ha una dimensione di s bit, forma il primo segmento del testo cifrato, che chiameremo C1. Successivamente il secondo blocco di input I2 viene formato facendo sul primo blocco di input I1 un'operazione di shift verso sinistra di s bit, C1 poi viene ricopiato nelle ultime s posizioni di I2. Il risultato del blocco I2 viene trattato esattamente come per I1.

Output Feedback (OFB)

Questa modalità è simile alla CFB, ma con il vantaggio che gli errori di trasmissione non si propagano. In questo modo si è soggetti a possibili vulnerabilità in caso di attacchi mirati alla modifica del flusso. L'input per

il blocco successivo, che abbiamo precedentemente chiamato I2 infatti non è C1 ma proprio il risultato di I1, in questo modo è facile capire che non ci può essere una propagazione degli errori. OFB quindi si adatta a tutte quelle situazioni in cui il canale potrebbe generare errori di trasmissione e quindi quest'algoritmo è molto utile per fare in modo che questi non si propaghino.

2.3 File system

Il file system è quell'insieme di algoritmi e tipi di dati astratti per la memorizzazione e l'organizzazione di file su un dispositivo di archiviazione. Tale dispositivo può essere sia fisico che virtuale. I file sono un meccanismo di astrazione che da la possibilità a chi lo utilizza di ignorare il modo con cui i dati contenuti vengono effettivamente memorizzati sul dispositivo di archiviazione.

2.3.1 Tipologie

Disk file system

Studiati appositamente per i dispositivi di archiviazione dove è possibile accedere in maniera random ad un particolare indirizzo e in un tempo relativamente breve. Solitamente, a causa dell'implementazione meccanica di tali dispositivi, questi file system sono fatti in modo da tener conto dei tempi necessari affinché la testina raggiunga la posizione richiesta e di conseguenza implementano vari algoritmi, che minimizzano questi tempi. Esempi tipici di file system di questo tipo sono:

- Ext2 (Extended File System 2, utilizzato in ambiente GNU/Linux. Famose anche le successive versioni Ext3 ed Ext4)
- FAT32 (utilizzato su DOS e Windows, è la versione migliorata di FAT)
- ExFAT (conosciuto anche come FAT64, creato da Microsoft per i dispositivi di archiviazione a stato solido)
- NTFS (NT File System, utilizzato su Windows)
- HFS (Hierarchal File System, utilizzato su Mac OS)

Optical discs file system

Studiati appositamente per i dispositivi ottici come CD, DVD e Blu-ray. I più famosi sono ISO 9660 e Universal Disk Format (UDF).

Flash file system

Studiati appositamente per le memorie flash. Quest'ultime infatti hanno proprietà diverse rispetto ai dischi rigidi:

- Ogni blocco deve essere cancellato prima di poterci scrivere nuovamente e il tempo di cancellazione può essere abbastanza rilevante, per questo motivo si tende a cancellare tali blocchi quando il dispositivo non viene utilizzato.

- Non c'è più ritardo per accedere ad una particolare posizione, poiché non c'è più la testina meccanica In questo modo l'ordine delle operazioni può essere svolto in maniera diversa.

- Nelle memorie flash i singoli blocchi tendono a rompersi dopo tante scritture, per questo motivo questi file system tendono a scrivere i dati uniformemente su tutto lo spazio disponibile.

Network file system

Studiati per permettere la condivisione di uno spazio di archiviazione reale su una rete, questi file system vengono chiamati anche **Distributed File System**. Con tali file system è possibile accedere indirettamente a file che si trovano in altri host, utilizzando una rete. Hanno la necessità di avere particolari meccanismi per evitare errori di concorrenza, possono implementare meccanismi di autenticazione e cifratura dei dati. NFS (Network File System) sviluppato da Sun è il file system storicamente più famoso.

Altri file system

Esistono poi anche dei file system specifici che possono essere utilizzati in altro modo. Naturalmente, come spesso accade, non esiste una soluzione perfetta che risolva tutti i problemi nel migliore dei modi, in questo senso esistono soluzioni specifiche che si adattano meglio a situazioni particolari.

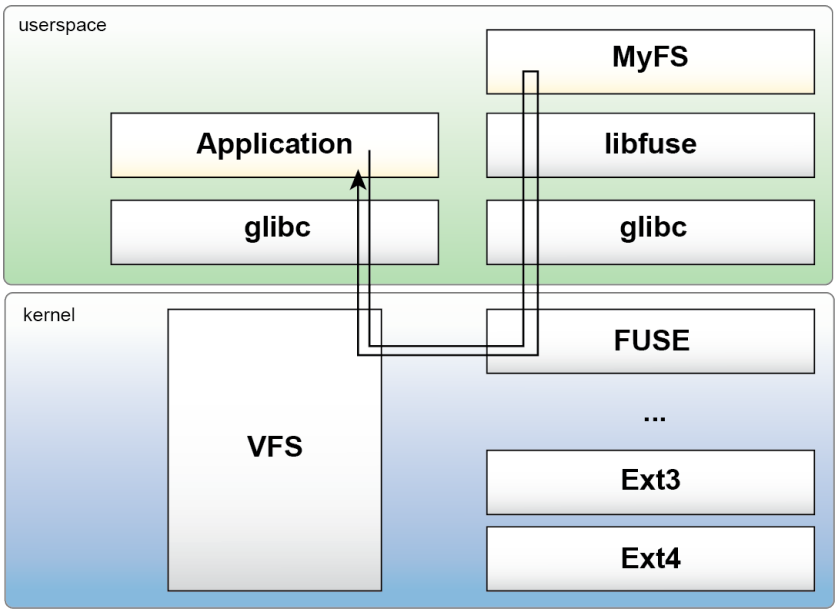


Figura 2.9: FUSE description

2.3.2 FUSE

Introduzione

Filesystem in Userspace (FUSE)[3] è un **modulo del kernel** caricabile in tutti i sistemi operativi considerati "**Unix-like**". Questo modulo permette agli utenti, che non hanno privilegi di root, di poter montare un proprio file system senza editare il kernel. Un utente che vuole scrivere un proprio file system MyFS lo scriverà utilizzando la libreria messa a disposizione da FUSE, tale codice verrà eseguito lato utente con i relativi privilegi e limitazioni. Per fare questo FUSE, che ha il compito di gestire tutte le chiamate fatte al file system MyFS, rimanda tutte le chiamate che sono state fatte al **VFS**[22] al modulo lato kernel di FUSE. Tale modulo poi farà in modo che tali chiamate possano essere gestite dalle funzioni contenute nel file system MyFS, come già detto prima tali funzioni vengono eseguite lato utente.

FUSE viene rilasciato sotto la licenza GNU (General Public License), quindi fa parte della categoria di software che possiamo liberamente utilizzare, poiché liberi. FUSE è disponibile principalmente per Linux, ma anche per

OpenSolaris, Mac OS X e FreeBSD. Dalla versione di Linux 2.6.14, FUSE diventa parte integrante del kernel.[23]

Scopo dell'utilizzo

FUSE viene utilizzato per scrivere file system di ogni genere. In sistemi operativi **Unix-like** si ha il semplice concetto di "**Everything is a File**", in questo modo ci sono dei sotto-alberi del nostro filesystem principale che possono essere gestiti nei modi più disparati. Possiamo pensare anche al sotto-albero "`/proc`", questo viene creato dal kernel dinamicamente e quindi esso non risiede di fatto nel disco ma in memoria. In un'architettura di questo tipo è semplice capire come molte volte è necessario creare veri e propri sotto-alberi gestiti da un programma, che viene eseguito lato utente piuttosto che lato kernel. In questo modo si può virtualizzare qualunque cosa, realizzare visioni semplificate di altri sotto-alberi esistenti già nel file system, avere una visione a file di una struttura dati di un programma, ed altro ancora. Elencheremo adesso una serie di file system che sono fra i più conosciuti:

- Wuala: Un file system distribuito basato su Java.
- WebDrive: Un file system commerciale che implementa WebDAV, SFTP, FTP, FTPS e Amazon S3
- SSHFS: permette di accedere ad un file system remoto tramite SSH
- EncFS: Encrypted virtual filesystem
- NTFS-3G e Captive NTFS: permettono di accedere ai file system NTFS
- WikipediaFS: permette di visualizzare ed editare gli articoli di Wikipedia come se fossero dei file
- TrueCrypt: permette di creare un disco virtuale, che può essere formattato nel modo che si vuole. Tale disco può essere montato per poter avere una reale cifratura on-the-fly dei file.

- s3fs-FuseOverAmazonS3: permette di accedere ad Amazon S3. Si può montare un bucket come un file system, in questo modo in maniera automatica i dati vengono memorizzati in S3.

- fuse-zip: permette di utilizzare un file zip come filesystem dove tutto quello che viene messo all'interno viene compresso.

How it works

FUSE si presenta come un pacchetto che contiene queste cartelle:

- **./doc** Contiene tutta la documentazione di FUSE.

- **./kernel** Contiene tutto il codice sorgente del modulo kernel di FUSE, quindi non serve il contenuto di questa cartella se si intende implementare un filesystem con FUSE.

- **./include** Contiene tutti gli header necessari per implementare un file system con FUSE, l'unico file che interessa in particolar modo è quindi **fuse.h**.

- **./lib** Contiene il codice sorgente usato per creare la libreria di FUSE, quindi contiene i file binari che il proprio file system deve linkare.

- **./util** Contiene il codice sorgente di programmi ausiliari per FUSE come **fusermount.**

- **./example** Contiene alcuni esempi di file system.

Un generico file system MyFS sarà semplicemente un programma, questo avrà la propria funzione `main()`. Tale funzione farà tutti i controlli e procedure che servono e successivamente chiamerà la funzione `fuse_main()` che cederà completamente il controllo a FUSE. `fuse_main()`[6] parsa gli argomenti che sono stati passati e successivamente monta il file system chiamando un

[6]per maggiori informazioni guardare ‘‘`lib/helper.c`’’

opportuna funzione `fuse_mount()`[7]. Quest'ultima funzione `fuse_mount()` crea un UNIX socket pair e in seguito ad un fork lancia in esecuzione il programma **fusermount**[8] che controllerà che il modulo di FUSE è stato caricato per poter essere quindi utilizzabile. A questo punto viene aperto un device in ``/dev/fuse'' e viene ritornato l'handle di tale file alla funzione `fuse_mount()`. Quest'ultima restituisce tale handle alla funzione `fuse_main()` che per prima cosa chiama `fuse_new()` che provvederà a creare tutte le strutture necessarie per il mantenimento del file system e successivamente chiama la funzione `fuse_loop()` che leggerà le modifiche al file ``/dev/fuse''. `fuse_loop()`[9] chiamerà opportunamente le varie funzioni che compongono il file system MyFS che chiameremo **operazioni**. Le operazioni che compongono MyFS vengono descritte dal `main()` utilizzando una struttura di tipo `struct fuse_operations`, tale struttura viene passata come parametro alla chiamata `fuse_main()`. Le varie operazioni possono restituire un valore e tale valore di ritorno viene mandato al modulo lato kernel utilizzando sempre il file ``/dev/fuse''.[23][24]

[7] per maggiori informazioni guardare ``lib/mount.c''

[8] per maggiori informazioni guardare ``util/fusermount.c''

[9] per maggiori informazioni guardare ``lib/fuse.c''

Capitolo 3

Progetto

3.1 Motivazioni

Nel mese di Aprile del 2011, Christopher Soghoian[4] scrive un articolo che prende di mira uno dei servizi più famosi nel campo del cloud storage, Dropbox. Soghoian spiega la necessità di tutelarsi da questo servizio poichè questo non protegge i dati mantenuti nei loro server con sistemi crittografici.

Dropbox infatti non duplica i dati che sono condivisi fra più utenti ma ne viene tenuta solamente una copia. Nella sezione della documentazione che riguarda la sicurezza[14], viene spiegato come i dati vengono memorizzati cifrati con AES-256, utilizzando come chiave la stessa password dell'account, rendendoli quindi inaccessibili anche ai dipendenti della società stessa. Nasce però subito una contraddizione, poiché in questo modo non è possibile implementare il sistema delle cartelle condivise presente in Dropbox. Inoltre, nel mese di Aprile del 2011, Dropbox ha cambiato la sezione della documentazione che riguarda la sicurezza, rimuovendo la frase "and are inaccessible without your account password". Questo fa nascere ulteriori dubbi su come i dati vengono effettivamente memorizzati. Ci sono poi altre considerazioni che vengono discusse nel documento scritto da Soghoian che convincono ancora di più la necessità di una maggiore consapevolezza dei rischi legati all'uso di questi servizi.

La motivazione principale, che ha spinto alla realizzazione di questo progetto, è proprio contenuta nelle considerazioni fatte da Soghoian. Ancor prima della lettura di tale articolo si è pensato che il semplice fatto che i dati dovevano in qualche modo essere accessibili da più utenti inevitabilmente conduce a pensare che questi non possono essere accessibili solo con la password dell'account. Inoltre, in una implementazione di questo tipo, il semplice cambio di password dell'account avrebbe provocato una grande mole di lavoro da parte dei server di Dropbox che difficilmente si può pensare sia quella realmente utilizzata. Dopo la lettura dell'articolo di Soghoian, le motivazioni che erano state congetturate, hanno avuto una conferma.

Il progetto quindi prevede come prima cosa la protezione dei dati sensibili su un servizio di cloud storage come Dropbox. Naturalmente per andare

incontro alle esigenze dei servizi offerti dallo stesso bisognava che il progetto diventasse più complesso.

Un primo metodo, per la protezione dei propri dati, potrebbe essere quello di conservarli in un archivio cifrato. Questo metodo però diventa scomodo nel caso in cui ci siano dei cambiamenti frequenti ai file poiché ogni volta bisognerebbe estrarre l'intero archivio, cambiare i file e ricreare l'intero archivio. Inoltre, in questo modo, la velocità di servizi avanzati come Dropbox viene penalizzata poiché l'intero archivio subirà un cambiamento radicale nonostante al suo interno è stato cambiato solamente un piccolo file.

Un altro metodo potrebbe essere quello di utilizzare programmi come TrueCrypt[25] che creano file di dimensione fissa, dove al proprio interno viene scritto un vero e proprio disco virtuale che però viene letto e scritto in modo cifrato. Facendo varie prove si è visto come questi grossi file di circa 200MB al loro interno fossero strutturati in modo che cambiamenti a piccoli file non stravolgevano l'intero file, in questo modo si ha già una velocità più elevata nella sincronizzazione. Inoltre programmi come TrueCrypt hanno un modo semplice di accedere a tali dischi virtuali poiché possono essere montati in una qualunque cartella. Ancora una volta però questo sistema si è rivelato un po limitativo poiché si ha a disposizione uno spazio limitato che poi non può essere modificato.

Un metodo invece che viene molto utilizzato è l'uso di programmi come EncFS che, come TrueCrypt, montano una cartella che chiameremo **MountFolder** e tutto ciò che viene scritto su di essa viene scritta in un'altra cartella che chiameremo **RootFolder** ma in maniera cifrata. Il contenuto dei file infatti viene cifrato on-the-fly e in alcuni programmi come anche EncFS anche il nome dei file viene cifrato. In questo tipo di programmi l'operazione di mount richiede quindi una password che sarà l'effettiva chiave con cui i file vengono cifrati e decifrati. In questo modo non c'è un vero e proprio file che deve essere precedentemente allocato e che limita lo spazio a disposizione, lo spazio utilizzato quindi cresce insieme allo spazio utilizzato dai file. Inoltre un sistema di questo tipo è più tollerante rispetto alla corruzione dei

file poiché in sistemi, come quello di TrueCrypt, la corruzione di una piccola parte del file potrebbe compromettere irrimediabilmente tutto il contenuto del disco virtuale rappresentato da quel file.

Ancora una volta però quest'ultima tipologia di programmi, che sono tutti molto simili fra loro, non si adatta a servizi come Dropbox poiché in situazioni in cui il contenuto che si vuole cifrare sia proprio una cartella condivisa con altri utenti, la password deve essere comunicata facendo insorgere tutti i problemi legati all'introduzione di sistemi crittografici a chiave asimmetrica di cui si è discusso precedentemente.

Per questi motivi si è voluto creare un file system che non utilizzasse più una semplice password ma delle chiavi RSA che permettessero di sfruttare le cartelle condivise di servizi di cloud storage come Dropbox. [4]

3.2 Contesto tecnologico

3.2.1 EncFS

EncFS[5] è un progetto sotto licenza GPL, un file system realizzato in C++ mediante FUSE che cifra i file on-the-fly. Tutte le operazioni che vengono effettuate sulla cartella MountFolder vengono riportate su una cartella RootFolder, in particolare le operazioni come write e read vengono effettuate cifrando e decifrando opportunamente i dati utilizzando una chiave. EncFS viene a conoscenza della chiave da utilizzare al momento del mount, viene infatti chiesta una password che decifra un file di configurazione dentro il quale è contenuta la **Volume Key** che serve per l'algoritmo di cifratura. In questo modo è possibile cambiare la password con la quale si effettua il mount senza dover re-cifrare tutto il contenuto. EncFS utilizza OpenSSL come libreria, questa infatti mette a disposizione degli algoritmi crittografici che possono essere utilizzati nel modo più opportuno. Possiede due metodi di cifratura dei contenuti. Il primo è lo **stream encryption** che provvede ad utilizzare la modalità CFB per cifrare i file, il secondo invece è il **block encryption** che cifra utilizzando la modalità CBC. Nel secondo metodo, la dimensione dei blocchi deve essere un multiplo della dimensione con cui l'algoritmo di cifratura a blocchi è in grado di lavorare. I nomi dei file vengono cifrati e la dimensione dei file viene parzialmente nascosta quando viene utilizzata la modalità a blocchi.

3.2.2 MetFS

MetFS[26] è un progetto libero realizzato da Metin KAYA in C utilizzando la libreria FUSE. Come EncFS, cifra i file on-the-fly riportando opportunamente tutte le chiamate fatte alla una cartella MountFolder ma in questo caso il contenuto non viene salvato in un altra cartella ma in un unico file che non ha una dimensione fissa, ma che varia in base ai contenuti. Anche in

questo caso per montare il file in una cartella MountFolder è necessaria una password. Questi tipi di file hanno un estensione ".mfs".

3.2.3 CryptoFS

CryptoFS[27] è un progetto libero realizzato da Christoph Hohmann in C utilizzando la libreria FUSE. Come EncFS cifra i file on-the-fly riportando opportunamente tutte le chiamate fatte alla una cartella MountFolder ad un altra reale RootFolder. Il progetto è molto simile ad EncFS con sottilissime differenze. CryptoFS infatti è realizzato con un core unico che astrae il funzionamento, tale core può poi essere utilizzato sia da FUSE che da Linux Userland FileSystem (LUFS). Anche CryptoFS richiede una password per cifrare e decifrare il contenuto dei file, al momento del mount però c'è una gestione più complessa che garantisce una buona sicurezza. Infatti, al momento del mount, vengono generate N chiavi utilizzando quella fornita, queste sotto-chiavi poi vengono utilizzate ciclicamente nei vari blocchi. In questo modo un file con lo stesso contenuto per tutti i blocchi presenterà risultati identici solamente dopo N blocchi. Come per EncFS vengono cifrati i nomi dei file.

3.2.4 Magikfs

Magikfs[28] è un progetto realizzato per avere un file system che avesse una sicurezza utilizzando la steganografia. La steganografia che è sempre stata utilizzata fin dai tempi più antichi, infatti si propone di nascondere il contenuto di un messaggio, piuttosto che renderlo incomprensibile. Nel caso particolare dei file, questi vengono mascherati dentro altri file come immagini e file audio. Con un file system che utilizza la steganografia si può sempre negare che determinati file siano realmente presenti, inoltre se i file vengono anche cifrati si introduce un ulteriore sicurezza poiché un attaccante deve per prima cosa trovare dove sono nascosti i file e poi deve decifrarli.

3.2.5 LessFS

LessFS[29] è un progetto sotto licenza GPLv3 realizzato in C da Mark Ruijter, anche questo utilizza la libreria FUSE. Viene sviluppato per fornire una soluzione flessibile alla deduplicazione dei dati su sistemi Linux. Per fornire la deduplicazione dei dati viene calcolato un hash di 192 bit per ogni blocco dati che viene scritto, in questo modo lessFS può sapere se tale blocco deve essere scritto o se deve creare un puntatore ad un blocco già esistente con lo stesso hash. I blocchi inoltre vengono compressi utilizzando LZP, QuickLZ o BZip. Inoltre è possibile cifrare tali blocchi realizzando quindi anche una protezione dei contenuti. LessFS si presta molto in tutti i casi in cui si vuole ridurre lo spazio di backup, mantenendo comunque la segretezza dei contenuti.

3.3 Descrizione

Il progetto viene chiamato **pubcFS**, viene realizzato utilizzando come linguaggio il C e la libreria FUSE. pubcFS è un file system che, grazie a FUSE, è in grado di montare una cartella che abbiamo chiamato **MountFolder**. Tutte le operazioni che vengono svolte in questa cartella vengono elaborate al fine di cambiare il contenuto di una cartella **RootFolder**. In particolare le operazioni di scrittura cifrano i dati in modo che in RootFolder vengano memorizzati in maniera sicura, successivamente questi dati cifrati verranno decifrati durante le operazioni di lettura. Il risultato è che l'utente avrà l'impressione di avere una cartella MountFolder dove i dati sono salvati in chiaro mentre in realtà tali dati sono solamente il risultato di una decifratura on-the-fly dei dati presenti nella cartella RootFolder.

3.3.1 Progettazione pensata per i cloud storage

La RootFolder può essere creata ovunque si voglia, un gruppo di utenti che condividono una cartella, che chiamiamo **SharedFolder** , possono utilizzare tale cartella come RootFolder in modo che tutto il contenuto della cartella condivisa venga cifrato. Eventualmente possono creare ed utilizzare una sotto-cartella di SharedFolder come RootFolder, in modo che solo alcuni dati rimangono cifrati. Citando come esempio Dropbox, questo crea nella propria home una cartella dove tutto il contenuto viene periodicamente sincronizzato con il server, è possibile poi che alcune sue sottocartelle siano condivise con altri utenti, cioè che siano delle SharedFolder.

Il progetto, come già detto in precedenza, si prefigge l'uso di sistemi crittografici a chiave asimmetrica per poter realizzare un file system che potesse essere montato da più persone, senza la necessità che queste debbano concordare una chiave.

Per quanto riguarda la gestione degli utenti si hanno queste semplici linee guida:

- Esiste sempre un creatore della cartella RootFolder, quest'ultimo viene chiamato R. Al momento della sua creazione R dovrà inizializzarla in modo che possa essere utilizzata da pubcFS.

- R può decidere quali utenti possono decifrare il contenuto di RootFolder abilitandoli in modo opportuno.

- R non deve essere in alcun modo favorito rispetto agli altri utenti che sono stati abilitati, in questo modo viene garantita la parità degli utenti. Gli altri utenti abilitati potranno infatti aggiungere o rimuovere altri utenti.

Servizi come Dropbox sono costruiti in modo che il contenuto delle SharedFolder possa essere modificato da ogni utente abilitato senza restrizioni, in maniera brutale ogni utente abilitato può ovviamente anche rimuovere file creati da altri. Il gruppo di utenti che condividono una SharedFolder sono quindi tutti uguali fra loro ed il creatore di tale cartella non ha privilegi speciali rispetto agli altri. PubcFS segue esattamente queste linee guida garantendo questa parità. Altra nota importante è quella che pubcFS gestisce gli utenti che possono decifrare il contenuto di RootFolder ma non gestisce assolutamente chi può leggere il contenuto di RootFolder poiché quest'ultima viene gestita esclusivamente dal servizio di cloud storage utilizzato.

Seguendo l'esempio di altri progetti come EncFS nasce l'esigenza che il nome dei file venga cifrato, ma per l'utilizzo che deve essere fatto di pubcFS nasce l'esigenza che solamente alcuni nomi di file vengano cifrati, in questo modo la RootFolder avrà sia file con un nome in chiaro che file con un nome cifrato. Questo per il semplice motivo che, in situazioni in cui si lavora con cartelle condivise con altri utenti, servizi come Dropbox avvisano con dei messaggi il cambiamento di un file citandone il nome, se il nome fosse cifrato il gruppo di utenti che condividono la SharedFolder avrebbe problemi nella loro coordinazione. Il progetto infatti nasce con la volontà che tutti i nomi dei file vengano lasciati in chiaro, ma in seguito si è deciso comunque di

riservare la possibilità che l'utente possa cifrare il nome di file specifici dove anche il nome stesso è un dato sensibile.

3.3.2 Funzionamento

L'utente **R** crea la cartella RootFolder, tale cartella, per essere utilizzata come tale, deve essere per prima cosa inizializzata da pubcFS. Quest'ultimo inizializza RootFolder creando una speciale cartella di configurazione che viene chiamata `‘‘.pubcfs’’`. PubcFS utilizza **AES** sia per cifrare il contenuto dei file che per cifrare il nome dei file. Questo perché, per ottenere buone performance, deve essere utilizzato un sistema crittografico a chiave simmetrica. La chiave per la cifratura, che chiamiamo **Key**, viene decisa al momento dell'inizializzazione della RootFolder e viene cifrata utilizzando la chiave pubblica di R, tale chiave cifrata la chiameremo **R-Key**. R-Key viene salvata nella cartella di configurazione `‘‘.pubcfs/keys’’` in un file che viene nominato con il nome dell'utente R. Per questo motivo R al momento dell'inizializzazione deve fornire la propria chiave pubblica e il nome con il quale vuole essere identificato in RootFolder, quest'ultimo nome lo chiameremo **UserName**. Successivamente R può chiedere a pubcFS di montare RootFolder in MountFolder, per fare questo occorre che R fornisca la propria chiave privata e il proprio UserName, in seguito viene decifrata R-Key utilizzando la chiave privata di R in modo da ottenere la chiave Key. In questo modo pubcFS può cifrare e decifrare utilizzando Key come chiave fino a quando la cartella non viene smontata. Per la cifratura e decifratura di Key viene utilizzato **RSA**.

R poi potrà aggiungere un altro utente **A** fornendo il suo UserName e la sua chiave pubblica. PubcFS, per poter aggiungere l'utente, dovrà decifrare R-Key utilizzando la chiave privata di R e, successivamente, dovrà cifrarla nuovamente utilizzando la chiave pubblica di A in modo da ottenere **A-Key**. A-Key viene poi salvata nella cartella di configurazione della RootFolder in un file chiamato come l'UserName di A. In seguito A ed R potranno aggiungere o rimuovere altri utenti.

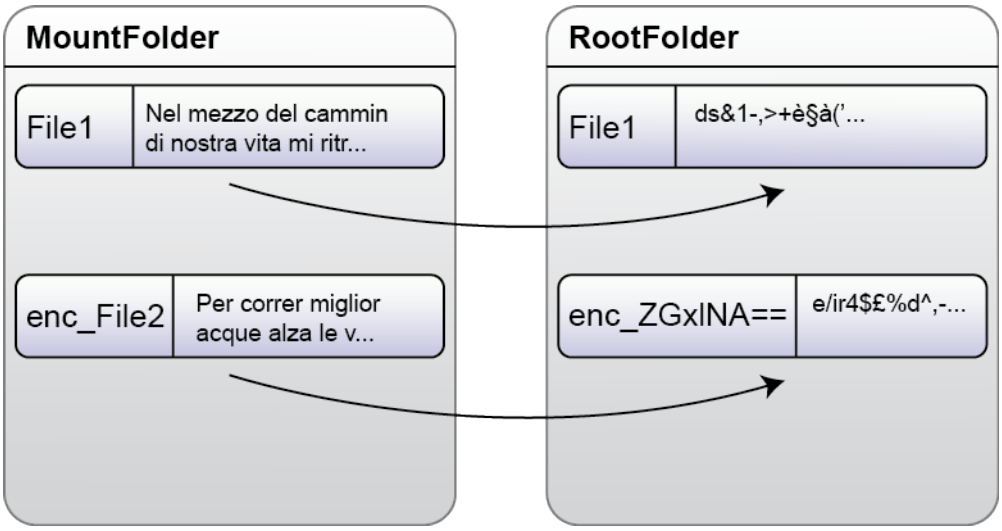


Figura 3.1: Funzionamento

Il nome di un determinato file (o cartella) può essere memorizzato in RootFolder in maniera cifrata, per fare ciò è necessario che il file (o cartella) venga nominato in modo che all'inizio ci sia la dicitura ‘‘enc_’’. PubcFS provvederà a cifrare il nome del file utilizzando sempre AES e la chiave Key, successivamente il risultato viene trasformato in un dialetto di base64[6] chiamato **base64url**[1] in modo che possa essere scritto in RootFolder. Facciamo un esempio: l'utente crea in

```
/path/to/MountFolder/miacartella
```

un file chiamandolo ‘‘enc_mionome’’, pubcFS cifrerà "mionome" con AES e successivamente lo trasformerà con base64url[6] ottenendo come risultato ‘‘ZG9lcnVtZg==’’, pubcFS a questo punto creerà il corrispondente file in

```
/path/to/RootFolder/miacartella/enc_ZG9lcnVtZg==
```

[1]questo dialetto è stato fatto proprio per permettere che il risultato possa essere posto in un url, in particolare il dialetto prevede che non venga utilizzato il carattere "/" e che al suo posto venga utilizzato il carattere "_"

3.4 Utilizzo

3.4.1 Cenni sulla creazione di chiavi RSA

Creazione chiave privata senza protezioni

Per poter creare una chiave RSA utilizzando i comandi messi a disposizione da OpenSSL:

```
openssl genrsa -out key.priv.pem 1024
```

Quest'ultimo comando genera una chiave RSA privata di 1024 bit e la scrive in un file chiamato `key.priv.pem` in formato PEM.

Creazione chiave privata con protezioni

Nel caso invece si voglia generare la chiave proteggendola con un algoritmo crittografico a chiave simmetrica si può lanciare il comando:

```
openssl genrsa -des3 -out key.priv.pem 1024
```

Nell'ultimo comando la chiave viene protetta utilizzando Triple DES, in generale è possibile comunque utilizzare l'algoritmo che si vuole come AES (‘‘`-aes128`’’,‘‘`-aes192`’’, ‘‘`-aes256`’’) oppure il DES normale (‘‘`-des`’’)

Creazione chiave pubblica derivandola dalla privata

Per generare la chiave pubblica è necessario avere prima quella privata, si può quindi generare con il comando:

```
openssl rsa -in key.priv.pem -pubout > key.pubb.pem
```

Quest'ultimo comando scrive in un file chiamato `key.pubb.pem` la corrispondente chiave pubblica in formato PEM relativa a quella privata contenuta nel file `key.priv.pem`.

3.4.2 Gestione della RootFolder

Viene spiegato in seguito come è possibile creare e gestire la RootFolder. Viene anche spiegato in modo dettagliato quello che viene fatto da pubcFS, questo perchè in seguito spiegheremo il motivo di tale implementazione.

Help

Nel caso si voglia richiedere l'help direttamente da pubcfs-config basta lanciare il comando ‘‘`pubcfs-config`’’ in questo modo:

```
pubcfs-config help
```

ricordiamo che l'help viene mostrato anche nel caso il numero di argomenti non sia corretto o nel caso in cui non ce ne siano del tutto.

Inizializzazione RootFolder

L'utente R che vuole inizializzare la RootFolder deve lanciare il comando ‘‘`pubcfs-config init`’’ in questo modo:

```
pubcfs-config init "/path/to/RootFolder" "R-UserName" \
                   "/path/to/R-PublicKey"
```

Viene inizializzata la RootFolder, che risiede in

```
"/path/to/RootFolder"
```

, creando la cartella di configurazione ‘‘`.pubcfs`’’. La chiave per la cifratura **Key**, viene generata e cifrata utilizzando la chiave pubblica di R che risiede in `"/path/to/R-PublicKey"`, tale chiave cifrata chiamata **R-Key** viene salvata in

```
"/path/to/RootFolder/.pubcfs/keys/R-UserName"
```

Viene anche creato un file

```
"/path/to/RootFolder/.pubcfs/config"
```

Al suo interno viene salvata la configurazione standard per la RootFolder, tale configurazione contiene vari parametri che possono essere modificati.

Inserimento di un altro utente

L'utente R che vuole aggiungere l'utente A deve lanciare il comando ‘‘`pubcfs-config add`’’ in questo modo:

```
pubcfs-config add "/path/to/RootFolder" "R-UserName" \
                  "/path/to/R-PrivateKey" "A-UserName" \
                  "/path/to/A-PublicKey"
```

In questo caso viene decifrata la R-Key utilizzando la chiave privata di R, tale R-Key ricordiamo essere contenuta in

```
"/path/to/RootFolder/.pubcfs/keys/R-UserName"
```

, ottenendo la chiave Key. Viene poi cifrata utilizzando la chiave pubblica di A ottenendo quindi l'A-Key. A-Key viene salvata poi in

```
"/path/to/RootFolder/.pubcfs/keys/A-UserName"
```

Rimozione di un utente

L'utente R che vuole rimuovere un utente B deve lanciare il comando ‘‘`pubcfs-config delete`’’ in questo modo:

```
pubcfs-config delete "/path/to/RootFolder" "B-UserName"
```

In questo caso viene rimossa la B-Key eliminando il file in

```
"/path/to/RootFolder/.pubcfs/keys/B-UserName"
```

Da notare che se pubcFS si accorge che B è l'ultimo utente che è abilitato alla decifratura della RootFolder, non rimuove la B-Key ed avvisa opportunamente l'utente che ha lanciato il comando. Questo perché eliminando la B-Key non si potrebbe più derivare la chiave Key rendendo inaccessibile RootFolder.

Elenco degli utenti abilitati

L'utente C può richiedere l'elenco degli utenti abilitati lanciando il comando ‘‘pubcfs-config list’’ in questo modo:

```
pubcfs-config list "/path/to/RootFolder"
```

In questo caso viene mostrata la lista di tutti gli utenti abilitati.

3.4.3 Mount e Unmount

Viene spiegato in seguito come è possibile montare e smontare la RootFolder in una cartella MountFolder, anche in questo caso spiegheremo le operazioni svolte da pubcFS per poterne poi discutere i motivi della sua implementazione.

Help

Nel caso si voglia ottenere l'help direttamente da pubcFS basta lanciare il comando ‘‘pubcfs’’ in questo modo:

```
pubcfs -h
```

ricordiamo che l'help viene mostrato anche nel caso in cui il numero di argomenti non sia corretto o nel caso in cui non ce ne siano del tutto.

Mount

L'utente R che vuole montare la RootFolder in una cartella MountFolder deve lanciare il comando ‘‘pubcfs’’ in questo modo:

```
pubcfs "R-UserName" "/path/to/R-PrivateKey" \
       "/path/to/RootFolder" "/path/to/MountFolder" \
       fuseOptions
```

Per prima cosa viene letta la R-Key memorizzata in

```
"/path/to/RootFolder/.pubcfs/keys/R-UserName"
```

e viene decifrata utilizzando la chiave privata di R in modo da ottenere la chiave Key. Successivamente viene montata in

```
"/path/to/MountFolder"
```

grazie alla libreria FUSE la cartella

```
"/path/to/RootFolder"
```

. Come spiegato precedentemente pubcFS elaborerà le operazioni che l'utente svolgerà nella MountFolder e modificherà opportunamente la cartella RootFolder in modo da memorizzarne i dati.

Unmount

L'utente R che vuole smontare la MountFolder deve lanciare il comando ‘‘`fusermount`’’ in questo modo:

```
fusermount -u "/path/to/MountFolder"
```

Capitolo 4

Architettura del progetto

4.1 Scelte progettuali

4.1.1 Algoritmo di cifratura per i contenuti

Come algoritmo crittografico per il contenuto e il nome dei file è stato scelto **AES**.

Per prima cosa bisogna dire che è stato scelto un algoritmo a chiave simmetrica per fare in modo che la cifratura dei file fosse molto veloce, inoltre in contesti dove il file deve essere disponibile a più utenti sarebbe stato molto difficile da implementare utilizzando direttamente un algoritmo crittografico a chiave asimmetrica. L'unica chiave Key per la cifratura dei contenuti viene condivisa e viene cifrata per ogni utente ottenendo N risultati diversi, mentre nel caso si fosse usato in modo diretto un algoritmo crittografico a chiave asimmetrica sarebbe stato il contenuto dei file a dover essere cifrato ottenendo N risultati diversi, questo avrebbe introdotto non pochi problemi di coerenza, concorrenza e sopratutto uno spreco inutile di risorse.

La scelta di un particolare algoritmo crittografico (AES) è stata fatta poiché come spiegato in precedenza viene ritenuto ormai uno standard a livello internazionale e garantisce una confidenzialità dei dati pari a quella dei documenti militari statunitensi. In particolare viene utilizzato AES-256 utilizzando la modalità di funzionamento CFB-8.

Per implementare la crittografia dei contenuti e dei nomi dei file è stata utilizzata la libreria **openssl** che ha al suo interno una serie di funzioni per la crittografia.

4.1.2 Lettura e scrittura

Un primo modo per poter implementare le operazioni di lettura e scrittura dei file potrebbe essere quello di leggere tutto il contenuto di un file quando viene richiesta una lettura con l'operazione `read`. In questo modo se si richiedono ad esempio 20 scritture di 2 byte queste dovranno per 20 volte decifrare tutto il file, modificarlo nei 2 byte che interessano e cifrarlo

nuovamente per intero, naturalmente più è grande il file e più le operazioni di read diventerebbero lente. Inoltre in questo modo si avrebbe l'ulteriore svantaggio che la modifica di due byte in un file molto più grande stravolgerebbe comunque tutto il file rendendo servizi di cloud storage come Dropbox molto più lenti.

Le funzioni relative alle operazioni di lettura e scrittura dei file sono state invece implementate in modo che il file venga visto come se fosse diviso in blocchi di dimensione `blockSize`. Quest'ultimo parametro viene deciso al momento dell'inizializzazione della RootFolder e viene memorizzato nel file di configurazione della stessa. In questo modo se si vuole leggere 1 byte, ad essere decifrato deve essere solamente il blocco che lo contiene, se il file è molto grande non ci sono rallentamenti. Inoltre, una modifica di due byte in un file molto più grande, stravolgerebbe solamente il blocco che li contiene, facendo in modo che servizi di cloud storage come Dropbox mantengano le stesse performance.

Per i motivi appena discussi il parametro `blockSize`, che può comunque essere modificato, non deve avere valori molto alti. Di default viene impostato a 64. Con un valore troppo basso invece non si garantirebbe la stessa sicurezza, poiché lo stravolgimento a seguito di una modifica di 2 byte sarebbe più contenuta.

4.1.3 Gestione utenti

Per garantire che gli utenti potessero gestire la chiave simmetrica Key è stato necessario l'utilizzo di un sistema crittografico a chiave asimmetrica. La chiave simmetrica Key viene cifrata e decifrata utilizzando **RSA**. Anche in questo caso è stato utilizzata openssl come libreria.

La chiave Key viene cifrata in modo diverso per ogni utente, per l'utente A infatti la chiave Key viene cifrata utilizzando la chiave pubblica di A, successivamente tale risultato viene memorizzato in

```
"/path/to/RootFolder/.pubcfs/keys/A-UserName"
```

dove A-UserName, come già spiegato in precedenza, è il nome utente con il quale A ha deciso di identificarsi all'interno della RootFolder. Se poi l'utente A vuole montare la RootFolder dovrà specificare la propria chiave privata in modo che pubcFS possa decifrare la chiave Key.

Si è deciso di non implementare la gestione delle chiavi mediante l'utilizzo di un piccolo database. Bisogna però considerare che con un database una situazione in cui due utenti A e B aggiungono rispettivamente gli utenti C e D, durante la sincronizzazione possono molto facilmente verificarsi dei conflitti[1]. Con la gestione degli utenti mediante file invece i conflitti possono verificarsi solamente in situazioni in cui A e B aggiungono due utenti C e D chiamandoli con lo stesso UserName, in quel caso però servizi come Dropbox rinominano il file più recente lasciando quindi la possibilità agli utenti di accorgersi dell'errore e lasciando soprattutto la possibilità di risolverlo facilmente con una semplice rinomina del file.

4.1.4 FUSE ed UMView

UMview è un progetto di Renzo Davoli che permette la virtualizzazione di macchine in modo parziale, consiste essenzialmente nel virtualizzare solamente alcune risorse, fra cui anche un file system. UMfuse, un particolare modulo di UMview, infatti permette con la stessa interfaccia di FUSE la possibilità di montare file system. L'unica differenza fra FUSE e UMfuse è che mentre in FUSE il file system viene implementato come un eseguibile che utilizza la libreria di FUSE, in UMfuse il file system viene implementato come libreria dinamica. Quasi tutti i moduli realizzati per FUSE sono compatibili con UMfuse, infatti i moduli compatibili con UMfuse devono essere implementati in modo che siano gestite situazioni in cui lo stesso file system viene montato due o più volte, questo perché UMfuse può montare e gestire più file system. In particolare è semplicemente l'uso di variabili globali che

[1] ricordiamo che prendendo l'esempio di Dropbox, i cambiamenti possono essere fatti offline e la sincronizzazione essere fatta in un secondo momento. Questo aumenta enormemente le probabilità che possa verificarsi un conflitto.

fa in modo che il file system non possa essere montato più di una volta. Una recente versione di FUSE ha infatti introdotto un parametro aggiuntivo nella funzione `fuse_main` chiamato `user_data`, con questo parametro il main di un file system può memorizzare il proprio "contesto" passandolo direttamente a FUSE, in questo modo ogni esecuzione dello stesso file system viene gestita con contesti diversi rendendolo compatibile con UMfuse.

Per rendere compatibile il file system con UMfuse, pubcFS memorizza tutti i dati in una particolare struttura chiamata `pubcfs_context`, al suo interno sono presenti tutti i dati necessari come il path della RootFolder, l'username dell'utente che ha montato il file system ed altro ancora.

4.2 Implementazione

4.2.1 Fasi di implementazione

L'implementazione di pubcFS comincia con lo studio approfondito di FUSE guardando documentazioni, guide ed esempi. Successivamente si realizza un primo esempio prendendo spunto dall'hello world disponibile negli esempi di FUSE e da quello si realizzano varie prove per apprenderne il funzionamento.

Una prima versione del progetto consiste nella realizzazione di un file system "identità", ovvero di un file system dove tutte le operazioni che vengono svolte sulla MountFolder vengono riportate esattamente identiche nella RootFolder. In questa versione tutte le funzioni già richiedono la cifratura della path alla funzione `pubcfs_encodePath` nonostante questa realizzi nient'altro che una concatenazione della path della RootFolder con la path relativa alla MountFolder senza eseguire nessuna cifratura. Inoltre la `readdir` richiamata la funzione `pubcfs_decryptName`, quest'ultima come la `pubcfs_encodePath` non realizza nient'altro che semplici copie senza l'utilizzo di algoritmi crittografici.

Successivamente questo file system viene modificato per fare in modo che le operazioni di lettura e scrittura vengano fatte a blocchi senza l'utilizzo di funzioni crittografiche. La dimensione dei blocchi è fissa e il suo valore memorizzato in una costante. In questa versione l'ultimo blocco, che potrebbe essere pieno solo per metà, viene comunque scritto tutto e riempito in fondo con dei byte con il valore 0. All'inizio del file viene memorizzato un intero di 2 byte che determina il numero di byte memorizzati nell'ultimo blocco. Con questa implementazione funzioni semplici, come la truncate, devono prendere il blocco corrispondente, riempire di 0 la parte del blocco che non interessa, fare la truncate in modo da escludere i blocchi successivi e modificare i primi due byte del file per informare lo spazio utilizzato dal nuovo ultimo blocco.

Un altra versione del progetto poi realizza tutte le funzioni che cifrano e decifrano i contenuti e vengono risolti i problemi di concorrenza con l'in-

troduzione del crypto context. In questa fase la chiave Key è predefinita. Solo in questa fase è stata poi scoperta la modalità CFB di cui non si era a conoscenza e sono state quindi semplificate molte funzioni. Con la modalità CFB infatti nell'ultimo blocco viene scritto solamente il necessario e quindi, se questo è riempito solamente per metà, verrà scritto solamente metà blocco. In questo modo vengono semplificate molte funzioni, la truncate ad esempio si trasforma in una semplicissima chiamata alla truncate reale, vengono quindi rimossi i due byte iniziali. Inoltre in questo modo la dimensione del file cifrato non è più un multiplo della dimensione del blocco ma è la dimensione effettiva del file.

Successivamente vengono implementate tutte le funzioni crittografiche per la gestione delle chiavi con tutti i comandi per l'inizializzazione della Root-Folder ed altre funzioni per la gestione degli utenti, viene quindi migliorata tutta la parte che riguarda l'interfaccia del programma introducendo anche messaggi di errore ed help. Questa fase è sicuramente la più corposa, quella che ha preso molto tempo per vari problemi tecnici che comunque non erano di progettazione.

L'ultima fase è stata implementare le funzioni di traduzione delle path utilizzando le funzioni crittografiche precedentemente implementate, in seguito poi sono state implementate le operazioni che gestiscono i link simbolici. Quest'ultima fase viene fatta per ultimo poiché non essenziale per il funzionamento del progetto, in questo senso si sono voluti ridurre i rischi legati a problemi che potevano sorgere nelle fasi precedenti. Durante quest'ultima fase si sono realizzate anche delle funzioni per la codifica e decodifica in base64url[6], è stata aggiunta anche la possibilità in queste funzioni di essere utilizzate per la codifica e decodifica anche nell'usuale base64[2].

Infine è stata scritta una piccola libreria per la lettura e scrittura della configurazione chiamata mConfig. Tale libreria mette a disposizione una serie di funzioni per la lettura e scrittura di parametri utili per la configurazione.

[2]si è preferito implementare da zero le funzioni poiché altrimenti si sarebbe dovuto modificare funzioni in base64 già esistenti da altri con i relativi problemi di licenze

Durante quest'ultima fase si è utilizzata questa libreria durante l'inizializzazione della RootFolder e in seguito durante il mount generico per leggere i parametri di configurazione della stessa.

4.2.2 Main

Il main consiste sostanzialmente in una parte di parsing degli argomenti, una di elaborazione e creazione del contesto, ed un ultima parte di chiamata alla libreria FUSE.

Per prima cosa viene creato il contesto di pubcFS con una `malloc`, il puntatore a questa struttura viene memorizzato in una variabile locale `ctx`. Durante la fase di parsing degli argomenti vengono presi tutti gli argomenti specifici di pubcFS e memorizzati dentro il contesto, successivamente gli altri argomenti vengono passati a FUSE utilizzando la funzione `fuse_opt_add_arg`. In questa fase può risultare che il numero di parametri non sia corretto o che venga richiesto esplicitamente di mostrare l'help, in questi casi quello che viene fatto è mostrare l'help dei comandi di pubcFS e in seguito anche l'help dei comandi relativi a FUSE.

Viene letta poi la configurazione della RootFolder chiamando la funzione `pubcfs_main_readConfig` e viene memorizzato tutto il necessario nel contesto. Successivamente vengono anche memorizzati nel contesto tutti i valori deducibili dai parametri passati in input. Mostriamo adesso la struttura che memorizza le informazioni del contesto:

```
typedef struct {
  char *rootPath;
  char *privateKeyPath;
  char *userName;
  ubyte *key;
  size_t keyLen;
  size_t blockSize;
  pthread_key_t cryptCtxKey;
} pubcfs_context;
```

Successivamente vengono inizializzate le funzioni che gestiranno la crittografia asimmetrica mediante la chiamata alla funzione `pubcfs_initRSAModule`. Una volta inizializzata la parte relativa alla crittografia asimmetrica viene letta la chiave privata dell'utente chiamando la funzione `pubcfs_readPrivateKey`. Quest'ultima funzione provvede anche a gestire le situazioni in cui ci siano chiavi private protette con password utilizzando una funzione standard della libreria openssl che fa inserire la password all'utente. Successivamente viene letta la chiave Key chiamando la funzione `pubcfs_readSimmetricKey`, tutti gli errori relativi a questa parte possono essere molti[3] e vengono tutti notificati all'utente. Una volta che viene letta la chiava Key questa viene memorizzata nel contesto ed infine viene chiamata la funzione `fuse_main` passando come argomento `user_data` il contesto `ctx` che è stato creato precedentemente. Ricordiamo inoltre che la funzione `fuse_main` ha bisogno anche che gli vengano passati gli argomenti che riguardano strettamente FUSE, uno fra i più importanti è la path che identifica la MountFolder. Altri argomenti specifici di FUSE sono quelli che realizzano opzioni particolari come l'esecuzione di FUSE in mono-thread ed altro ancora.

4.2.3 Gestione chiavi

Per la gestione delle chiavi asimmetriche bisogna prima di tutto cominciare a descrivere le funzioni `pubcfs_readPrivateKey` e `pubcfs_readPublicKey`. Entrambe le funzioni sono molto semplici poichè la gestione viene molto semplificata dalla libreria openssl. La prima apre il file che contiene la chiave privata in formato PEM utilizzando la funzione `fopen`, successivamente chiama la funzione `PEM_read_RSAPrivateKey` e chiude il file utilizzando `fclose`. Alla funzione `PEM_read_RSAPrivateKey` non viene specificata nessuna password e nessuna funzione che la richiede, in questo modo se la chiave è protetta è direttamente la libreria a svolgere le attività di inserimento della password.

[3] ad esempio all'username specificato non corrisponde nessuna chiave, oppure la chiave privata non riesce a decifrare la chiave Key per quel determinato utente poiché relativa ad una chiave pubblica diversa, ecc...

La seconda invece legge la chiave pubblica nel medesimo modo utilizzando la funzione `PEM_read_RSA_PUBKEY`.

La funzione `pubcfs_readSimmetricKey` invece provvede a identificare il file relativo all'user name dell'utente A, leggerne il contenuto in modo da ottenere la A-Key e successivamente decifrare la A-Key in modo da ottenere la chiave Key. Per fare questo viene prima di tutto determinata la dimensione della chiave privata di A utilizzando la funzione `RSA_size`, in questo modo si è a conoscenza di quanto deve essere letto dal file per ottenere la A-Key. Successivamente viene decifrata la A-Key mediante l'uso della funzione `RSA_private_decrypt`, da notare che prima della chiamata di quest'ultima viene chiamata la funzione `RSA_blinding_on` che protegge da attacchi basati sulla misurazione del tempo di decifratura. Nel caso la chiave Key sia più piccola rispetto a quello che è possibile cifrare e decifrare con la chiave RSA, viene utilizzato come padding la modalità PKCS #1, viene infatti specificata alla funzione `RSA_private_decrypt` un parametro con il valore `RSA_PKCS1_PADDING`.

La funzione `pubcfs_writeSimmetricKey` viene utilizzata per scrivere la chiave A-Key di un utente A su un file avente come nome l'user name specificato da A. Questa funzione ha prima di tutto il compito di cifrare la chiave Key utilizzando la chiave pubblica di A, per fare ciò è necessario conoscere prima la dimensione della chiave pubblica utilizzando la funzione `RSA_size` in modo da poter allocare lo spazio necessario per poter mantenere in memoria la A-Key. Successivamente viene fatto il controllo che esista la cartella di configurazione della RootFolder con la funzione `pubcfs_checkCreateConfigFolder`[4] e successivamente viene scritta la A-Key in un file che ha come nome l'user name specificato da A.

Come accennato nella descrizione del main vengono inizializzate le funzioni che gestiranno la crittografia asimmetrica con una funzione chiamata `pubcfs_initRSAModule`. Tale funzione è molto semplice poiché consiste solamente nella chiamata alla funzione `OpenSSL_add_all_algorithms`, questo

[4]questa funzione crea la cartella se non esiste

perché, ad esempio, le funzioni di lettura e scrittura delle chiavi non gestiscono per default chiavi in formato PEM facendo quindi andare in errore funzioni come `PEM_read_RSA_PUBKEY`[5].

Successivamente vengono inizializzate le funzioni che gestiranno la crittografia asimmetrica mediante la chiamata alla funzione `pubcfs_initRSAModule`.

4.2.4 Crypto Context

Dietro al Crypto Context si nasconde una fra le più complesse fasi dell'implementazione di pubcFS. FUSE infatti gestisce le operazioni che vengono fatte su un file system in multi-thread ed opzionalmente si può configurare in modo che possano essere gestite soltanto da un singolo thread. Durante i test infatti alcuni file presentavano delle anomalie molto insolite: erano per metà decifrati in modo corretto e per l'altra metà invece in maniera totalmente casuale pur mantenendo la dimensione complessiva del file identica a quella che avrebbe avuto se fosse stato decifrato in modo corretto. Dopo molte prove si è scoperto che il problema era semplicemente un problema di concorrenza poiché in modalità mono-thread i test non facevano risultare errori di nessun tipo, vista poi la composizione di questi file anomali si è scoperto che era un problema della cifratura e decifratura.

Le funzioni per la cifratura e decifratura infatti hanno bisogno di un contesto che deve essere salvato in una struttura EVP_CIPHER_CTX, tale contesto non rimane invariato dopo l'uso di funzioni per la cifratura e decifratura come `EVP_EncryptInit_ex`, `EVP_EncryptUpdate`, `EVP_EncryptFinal_ex`, `EVP_DecryptInit_ex`, `EVP_DecryptUpdate`, `EVP_DecryptFinal_ex`. Tale contesto invece viene continuamente modificato e in situazioni in cui FUSE lavora in multi-thread ogni thread ha bisogno di un proprio contesto per evitare errori come quello appena descritto.

[5] l'errore è abbastanza generico e la documentazione molto povera, durante l'implementazione infatti è stato complesso capire che l'errore era relativo al fatto che PEM non veniva letto per default e bisognava caricarlo appositamente con la chiamata ad una funzione specifica.

Per questo motivo vengono utilizzate delle **thread local storage key** che permettono di salvare dei dati in modo che siano locali per un singolo thread. Durante il `main` infatti viene chiamata la funzione `pthread_key_create` in questo modo:

```
pthread_key_create(&(ctx->cryptCtxKey),
   pubcfs_destroyCryptCtx);
```

con ciò viene creato un thread local storage key, il riferimento a tale chiave viene memorizzato dentro il contesto generale di pubcFS. Notare che viene anche specificata una funzione `pubcfs_destroyCryptCtx` che consiste essenzialmente nella funzione che distruggerà tutto il contenuto relativo alla thread local storage key nel momento in cui un thread cesserà di esistere. All'interno di questa thread local storage key viene memorizzata una struttura `pubcfs_cryptoCtx` contenente un contesto per la decifratura `en` ed un altro per la cifratura `de`, entrambi sono di tipo `EVP_CIPHER_CTX`.

```
typedef struct {
  EVP_CIPHER_CTX en;
  EVP_CIPHER_CTX de;
} pubcfs_cryptoCtx;
```

La struttura `pubcfs_cryptoCtx` però non può essere allocata e memorizzata all'interno della thread local storage key direttamente nel `main` poiché sarebbe una chiave relativa al solo thread che ha eseguito il `main`, deve essere invece allocata in modo dinamico. In particolare ogni operazione del file system che richiede di compiere operazioni crittografiche richiede sempre il crypto context utilizzando la funzione `pubcfs_getCryptoCtx`, quest'ultima recupera il contesto generale di pubcFS memorizzandolo in `ctx` e controlla se la thread local storage key è `NULL` oppure no, per fare ciò chiama la funzione `pthread_getspecific`.

```
pubcfs_cryptoCtx* pubcfs_getCryptoCtx(pubcfs_context*
   ctx){
  pubcfs_cryptoCtx* cctx;
```

```
  cctx = (pubcfs_cryptoCtx*)pthread_getspecific(ctx->
     cryptCtxKey);
  if(cctx == NULL){
    cctx = pubcfs_createCryptoCtx(ctx);
    pthread_setspecific(ctx->cryptCtxKey, cctx);
  }
    return cctx;
}
```

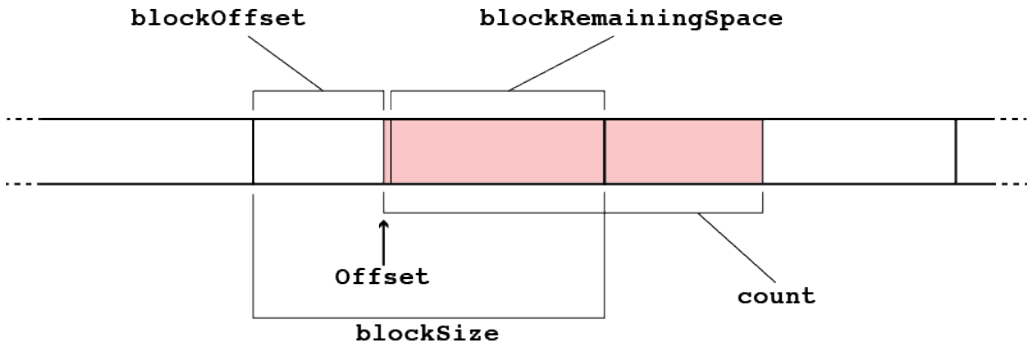
Nel caso questo sia `NULL` viene creato il contesto richiamando la funzione `pubcfs_createCryptoCtx` e successivamente viene memorizzato tale contesto nella thread local storage key utilizzando la funzione `pthread_setspecific`. Successivamente la funzione `pubcfs_getCryptoCtx`, restituisce il crypto context all'operazione che lo ha richiesto per poterlo utilizzare.

`pubcfs_createCryptoCtx`, come appena detto, crea il contesto che poi viene utilizzato. Per fare ciò viene chiamata la funzione EVP_BytesToKey specificando che si vuole utilizzare AES-256 con la modalità di funzionamento CFB, quest'ultima crea un vettore iniziale IV e una chiave derivata necessaria alle funzioni di inizializzazione. Successivamente vengono inizializzati i due contesti per la cifratura e decifratura richiamando le funzioni `EVP_EncryptInit_ex` e `EVP_DecryptInit_ex` e vengono memorizzati all'interno del crypto context.

4.2.5 Write & Read

Le funzioni relative alle operazioni di `write` e `read` funzionano ragionando sul file come se fosse diviso in blocchi di dimensione `blockSize`. Quest'ultimo viene deciso al momento dell'inizializzazione della RootFolder e viene memorizzato nel file di configurazione della stessa.

Come sappiamo da FUSE la `read` e la `write` hanno un parametro `offset`, infatti per questo motivo queste ultime operazioni in realtà sono delle `pread` e `pwrite`. Quando viene richiesta una `read` viene prima di tutto calcolato `block` che è il numero del blocco che contiene il byte puntato da `offset`,



successivamente viene calcolato il `blockOffset` che non è altro che l'offset del byte puntato dall'`offset` ma relativo all'inizio del blocco corrente `block`. Successivamente viene letto il blocco `block`, poi questo viene decifrato e successivamente viene copiato nel buffer di output `buf` il quale puntatore viene passato come parametro. I blocchi successivi vengono letti fino a quando è necessario, in totale bisogna leggere `count` byte, anche `count` viene passato come parametro.

```
/*...*/
endOffset = offset + count;
block = offset / blockSize;
while((block * blockSize) < endOffset){
/*...*/
```

Ovviamente è anche possibile che `count` sia molto piccolo rispetto al `blockSize` e che quindi venga letto e decifrato un blocco di `blockSize` byte per poi dover prendere molti meno byte. Per questo motivo un `blockSize` troppo grande rallenta troppo le performance mentre uno troppo piccolo garantisce minore sicurezza. Un altra variabile che viene in aiuto soprattutto in queste situazioni è la `blockRemainingSpace` che è il numero di byte da leggere in quel particolare blocco. Con ad esempio dei blocchi di 10 byte, un `blockOffset` di 2 ed un `count` di 50 allora il `blockRemainingSpace` relativo al primo blocco sarà di 8 byte mentre per i blocchi successivi sarà di 10 byte tranne per l'ultimo che sarà di 2 byte. Nel caso precedente se `count` fosse stato 5 il `blockRemainingSpace` si sarebbe ridotto a 5 byte e non ci

sarebbero state letture di altri blocchi.

```
/*...*/
if (((block + 1) * blockSize) <= endOffset){
  blockRemainingSpace = blockSize - blockOffset;
}else{
  blockRemainingSpace = (endOffset % blockSize) -
     blockOffset;
}
/*...*/
```

Durante questo ciclo che calcola gli indici dei vari blocchi viene chiamata la funzione `pubcfs_readBlock` che legge e decifra effettivamente un blocco. Quest'ultima funzione memorizza semplicemente in un buffer temporaneo di dimensione `blockSize` il risultato della lettura dalla posizione `(block * blockSize)` di `blockSize` byte, successivamente questo buffer viene decifrato chiamando la funzione generica `pubcfs_decrypt`.

La scrittura invece funziona in modo analogo con l'unica differenza che dopo essere stato letto il blocco, questo non viene copiato nel buffer di output `buf` ma viene riscritto utilizzando la funzione `pubcfs_writeBlock` dopo essere stato opportunamente modificato con i dati presenti nel buffer di input `buf`.

4.2.6 Percorsi cifrati

Alcune operazioni del file system vengono fatte su file specificandone il percorso, in FUSE il percorso è relativo al punto di mount, ovvero quella che abbiamo sempre chiamato MountFolder.

Una prima operazione che viene svolta da queste operazioni è richiedere la path relativa al file reale memorizzato nella RootFolder, questa path la chiameremo `e_path`. Essa può essere cifrata in alcuni punti se contiene nomi di cartelle che cominciano per ``enc_'', inoltre può essere cifrata anche l'ultima parte poiché anche il nome del file stesso può essere cifrato se

comincia per ‘‘enc_’’. Per ottenere la `e_path` viene chiamata la funzione `pubcfs_encodePath`. Tale funzione costruisce la `e_path` scrivendo per prima cosa la path della RootFolder, successivamente viene eseguito un loop sulla path in input in modo da ottenere i vari token delimitati dal simbolo ‘‘/’’. Per realizzare questo loop viene utilizzata `strtok_r`[6].

```
char *strtok_ctx, *token;
/*...*/
token = strtok_r(path_to_encrypt, "/", &strtok_ctx);
while(token != NULL){
  /*...*/
  token = strtok_r(NULL, "/", &strtok_ctx);
}
/*...*/
```

Un token rappresenta quindi un nome di una cartella o di un file, questo se inizia per ‘‘enc_’’ viene cifrato utilizzando la funzione generica `pubcfs_encrypt`[7]. Successivamente il risultato della cifratura viene trasformato in base64url[6] utilizzando la funzione `base64_encode` per essere utilizzabile in una path. Ricordiamo che se il token è nella forma “enc_nomeInChiaro” il risultato sarà nella forma “enc_nomeCodificatoInBase64url”. Una volta tradotto il token questo viene aggiunto alla stringa che formerà la `e_path`.

In modo analogo altre funzioni come la `readdir` richiedono invece di poter decifrare i nomi dei file compresi quelli che cominciano per ‘‘enc_’’. Per fare questo esiste una funzione chiamata `pubcfs_decryptName` che svolge proprio il compito di decifrare, se necessario, un token. Da notare che in questo caso non vengono tradotte path complete ma solamente singoli token che rappresentano nomi di file o nomi di cartelle.

Un altro punto dove vengono utilizzate in modo specifico le path è nelle operazioni per la gestione dei link. Quando viene creato un link, viene

[6] viene utilizzata la `strtok_r` poiché è la versione reentrant della `strtok`, questo per fare in modo che in situazioni di multi-thread questa non dia risultati errati.

[7] utilizzata anche durante la lettura di un blocco

eseguita l'operazione `link`, a questa vengono passati due parametri che rappresentano il link di origine e il link di destinazione. Tali link vengono cifrati entrambi utilizzando la funzione `pubcfs_encodePath` e successivamente viene creato un link fra queste due path cifrate utilizzando la reale funzione `link`. Durante la lettura invece, viene eseguita l'operazione `readlink`, tale operazione legge il link utilizzando la reale funzione `readlink`, successivamente la path letta viene tutta decifrata utilizzando sempre il solito loop che analizza tutti i token.

Capitolo 5

Conclusioni

5.1 Considerazioni sul progetto

Il progetto è stato una grande fonte di soddisfazioni, quando si utilizzano sistemi operativi dove tutto è un file è molto utile imparare strumenti come FUSE. Inoltre la passione incontenibile verso la crittografia e verso il funzionamento di sistemi operativi come GNU/Linux ha dato la giusta spinta per fare in modo che si potesse portare avanti il progetto a grande velocità.

Un parere negativo sull'utilità di questo progetto potrebbe essere quello che non ci fosse bisogno di una gestione delle chiavi con RSA e che era già abbastanza sufficiente una singola password. Una gestione con un'unica password dovrebbe però essere comunicata a tutti i membri che partecipano alla condivisione, per piccoli gruppi questo può essere fatto in modo banale utilizzando altri canali di comunicazione, ma ovviamente il progetto è stato pensato per essere opportunamente scalabile e quindi utilizzabile in contesti dove lo scambio deve essere fatto in un modo più formale e sicuro.

Un altro parere negativo potrebbe essere quello che il sistema non protegge i casi in cui alcuni utenti cancellano la configurazione della cartella. Naturalmente pubcFS, come molti altri programmi di crittografia, ha lo scopo di rendere incomprensibile i dati e non di certo di proteggerli da cancellazioni. Ci sono ad esempio alcuni programmi come TrueCrypt che possono cifrare tutto il contenuto di uno spazio di archiviazione ma anche in quel caso non c'è niente che protegga da una brutale formattazione che cancella tutti i dati. In pubcFS cancellando la cartella di configurazione infatti viene persa la possibilità di poterne decifrare il contenuto. Questo succede anche in altri file system cifrati come EncFS dove la password viene salvata cifrata esattamente come fa pubcFS con l'unica differenza che EncFS utilizza una crittografia simmetrica mentre pubcFS una asimmetrica in modo da avere tutta quella gestione degli utenti di cui si è parlato.

Per assicurare la coerenza delle operazioni che vengono svolte sono stati implementati alcuni test. Questi generano file di lunghezza casuale e con contenuti casuali, questi vengono poi scritti, riletti e confrontati byte dopo byte per verificare che i dati vengano cifrati, scritti, riletti e decifrati in

modo corretto. Anche sui singoli componenti, come il modulo per la codifica in base64[6], sono stati condotti dei test molto simili. Per quanto riguarda la gestione degli utenti, sono state fatte delle prove con un numero limitato di utenti, ogni utente aveva diverse chiavi pubbliche e private ed alcuni anche di dimensione diversa. Anche per i link sono stati condotti dei test per verificare il corretto funzionamento. Inoltre il sistema è attualmente utilizzato da un ristretto gruppo di persone senza riscontrare incongruenze di nessun tipo.

Il progetto trova la sua utilità nei casi in cui più persone condividono una cartella su servizi di cloud storage come Dropbox, è molto utile in questi casi che tutta la gestione dei dati sensibili venga svolta da un programma come pubcFS in modo trasparente utilizzando le proprie chiavi private e pubbliche, che permettono di risolvere tutti quei problemi legati allo scambio necessario di una password fra tutti gli utenti. In conclusione si sostiene che il progetto è molto interessante, poiché ha contribuito all'espansione del set di progetti riguardanti la protezione dei dati mediante l'utilizzo di strumenti come FUSE. Come si è precedentemente parlato infatti esistono vari progetti incentrati su questo tema ed alcuni sono anche molto simili fra loro. Questo progetto invece differisce in tutta quella parte che riguarda il mount della directory con tutti quei meccanismi che prevedono l'utilizzo di chiavi pubbliche e private, pubcFS tuttavia in questo modo rende sicuramente più complesso l'utilizzo rispetto ai sistemi come EncFS che chiedono una semplice password. Tuttavia questo gap potrà essere colmato mediante l'implementazione di interfacce che ne miglioreranno l'usabilità.

5.2 Limitazioni attuali e Sviluppi futuri

5.2.1 Sistemi operativi

PubcFS è stato progettato e implementato per essere utilizzabile su GNU/-Linux. Sarebbe comunque interessante adattare il progetto per essere utilizzabile in altri sistemi operativi come Mac OS. Per quest'ultimo sistema operativo, è disponibile la libreria di FUSE che viene chiamata MacFUSE. Questa permetterebbe l'evoluzione di questo progetto in modo da essere disponibile anche per Mac OS.

5.2.2 Cifratura dei nomi file

La progettazione ha portato a considerare la cifratura dei nomi dei file come una cosa estremamente rara e ha portato a progettare il sistema senza tener conto di specifiche esigenze. Infatti, se si volessero cifrare i nomi di tutti i file e sotto-cartelle contenute all'interno di una cartella, bisognerebbe cambiare i nomi dei file di tutto il sotto-albero. Per questo motivo il sistema dovrà avere la possibilità che particolari suffissi nel nome estendano la cifratura dei nomi per tutto il suo sotto-albero.

5.2.3 Esigenze specifiche per altri servizi

Nonostante pubcFS sia stato pensato per poter essere utilizzabile in qualunque servizio di cloud storage, sicuramente ha subito delle influenze da parte dei servizi più famosi come Dropbox e Ubuntu One. Naturalmente considerando altri servizi di cloud storage è possibile che nascano ulteriori esigenze e che siano necessari ulteriori feature. Per il momento i requisiti sono stati descritti in modo che pubcFS si adatti perfettamente su qualunque servizio di cloud storage conosciuto dagli sviluppatori.

5.2.4 Crittografia

PubcFS utilizza AES per cifrare e decifrare i contenuti dei file e i nomi dei file. Potrebbe essere interessante che l'algoritmo da utilizzare possa essere specificato all'interno delle impostazioni contenute nella RootFolder. Per fare questo bisognerebbe semplicemente modificare la parte di inizializzazione del crypto context. Questa dovrebbe quindi considerare i possibili valori della configurazione e inizializzare di conseguenza i contesti delle funzioni crittografiche in modo che utilizzino l'algoritmo crittografico specificato. Naturalmente gli algoritmi crittografici utilizzabili devono comunque essere disponibili sulla libreria openssl.

Tuttavia non si esclude la possibilità, facendo però molte più modifiche, di poter decidere direttamente la libreria crittografica da utilizzare. Con questa feature l'utente potrebbe linkare dinamicamente una libreria che implementi delle funzioni di cifratura e decifratura e pubcFS utilizzerebbe tali funzioni. Queste ovviamente dovrebbero rispettare determinati requisiti come ad esempio quello di cifrare il contenuto con la modalità di funzionamento CFB. Inoltre bisognerebbe apportare numerosi accorgimenti per fare in modo che queste funzioni rispettino determinati requisiti e abbiano la possibilità di poter essere eseguiti in maniera concorrente.

5.2.5 Gestione delle chiavi

La chiave simmetrica Key può variare modificando la configurazione della RootFolder. Le chiavi asimmetriche dei vari utenti però potrebbero non essere abbastanza lunghe da permettere di cifrare la chiave Key. pubcFS attualmente non esegue nessun warning nel caso si tenta di impostare la dimensione della chiave Key ad un valore troppo alto che nessuna chiave RSA abituale riuscirebbe a cifrare. Si ricorda che quando viene inserito un nuovo utente, che ha una chiave molto piccola, viene notificato un errore che semplicemente dice che non è stato possibile eseguire la cifrazione, non c'è quindi un check che identifichi che l'errore di cifratura è causato dalla

dimensione troppo piccola della chiave. Per causare questo errore insolito, con i parametri standard, occorre comunque una chiave molto molto piccola, ormai nessuno ha chiavi più piccole di 1024 bit.

5.2.6 Interfaccia

PubcFS potrebbe essere inoltre migliorato nell'interfaccia, in modo che possa essere utilizzabile anche da persone meno esperte, infatti come requisito fondamentale si richiede il saper utilizzare un terminale. Utenti che utilizzano Dropbox molto spesso sono persone molto poco esperte e per queste sarebbe interessante realizzare un interfaccia GUI. Inoltre, proprio per questo motivo, le persone potrebbero non essere a conoscenza del funzionamento dei sistemi crittografici a chiave asimmetrica e sarebbe quindi molto utile poter nascondere tutta la gestione delle chiavi creandole direttamente secondo le necessità e recuperando le chiavi pubbliche degli altri utenti utilizzando servizi conosciuti per la distribuzione di chiavi, come ad esempio il keyserver di PGP.

Bibliografia

[1] Bobbie Johnson. Cloud computing is a trap, warns GNU founder Richard Stallman. *The Guardian*, Settembre 2008.

[2] Neelie Kroes. Towards a european cloud computing strategy. In *World Economic Forum di Davos*, Gennaio 2011.

[3] FUSE filesystem in userspace official website. `http://fuse.sourceforge.net/`. Acceduto il 30 Settembre 2011.

[4] Christopher Soghoian. In the matter of Dropbox Inc. request for investigation and complaint for injunctive relief. 2011.

[5] Valient Gough. EncFS official webpage. `http://www.arg0.net/encfs`. Acceduto il 23 Settembre 2011.

[6] The Internet Society. RFC 4648: The Base16, Base32, and Base64 Data Encodings. 2006.

[7] Simone Brunozzi. Cloud computing. In *XX Congresso AIP (Associazione Informatici Professionisti)*, Assisi, Maggio 2009.

[8] Alessandro Mantelero. *Processi di Outsourcing Informatico e Cloud Computing*. 2010.

[9] Rean Griffith Anthony D. Joseph Randy H. Katz Andrew Konwinski Gunho Lee David A. Patterson Ariel Rabkin Ion Stoica Matei Zaharia Michael Armbrust, Armando Fox. Above the Clouds: A Berkeley view of cloudcomputing. 2009.

[10] Storage Networking Industry Association. *Overview of Cloud Storage*. 2011.

[11] Amazon Web Services. Amazon Web Services: Overview of security processes. September 2008.

[12] Amazon Web Services. Amazon Simple Storage Service (S3). `http://aws.amazon.com/s3/`. Acceduto il 20 Settembre 2011.

[13] Macfuse official website. `http://code.google.com/p/macfuse/`. Acceduto il 3 Ottobre 2011.

[14] Dropbox help. `http://www.dropbox.com/help`. Acceduto il 28 Settembre 2011.

[15] Dropbox terms of service. `http://www.dropbox.com/terms`. Acceduto il 28 Settembre 2011.

[16] 107th Congress (2001-2002). H.R.3162.ENR. 2001. Patriot Act Law.

[17] Garante per la protezione dei dati personali. Cloud computing : indicazioni per l'utilizzo consapevole dei servizi.

[18] Alessandro Capodaglio. Firma digitale and crittografia. 2011.

[19] John Schwartz. U.S. selects a new encryption technique. *The New York Times*, Ottobre 2000.

[20] National Institute of Standards and Technology. DES modes of operation. *Federal Information Processing Standard*, Dicembre 1980.

[21] National Institute of Standards and Technology. Special publication 800-38A. Recommendation for Block Cipher Modes of Operation. *NIST Special Publication*, 2001.

[22] Alessandro Rubini. The Virtual File System in Linux. 1997.

[23] Terje Oseberg. How FUSE works. Febbraio 2011. from FUSE documentation.

[24] Sumit Singh. Develop your own filesystem with FUSE. Febbraio 2006.

[25] TrueCrypt official webpage. `http://www.truecrypt.org/`. Acceduto il 23 Settembre 2011.

[26] Metin KAYA. MetFS official webpage. `http://www.enderunix.org/metfs/`. Acceduto il 23 Settembre 2011.

[27] Christoph Hohmann. CryptoFS official webpage. `http://reboot78.re.funpic.de/cryptofs/`. Acceduto il 23 Settembre 2011.

[28] Anoop.S Vivek.K.P. Varun Suresh, Shibin.K. Magikfs official webpage. `http://magikfs.sourceforge.net/`. Acceduto il 23 Settembre 2011.

[29] lessfs official webpage. `http://www.lessfs.com/wordpress/`. Acceduto il 23 Settembre 2011.

[30] *Virtual Square: Users, Programmers and Developers Guide.* Renzo Davoli, Michael Goldweber, 2011.

[31] Valerio Vertua Yvette Agostini. Il cloud data storage: stato dell'arte, rischi e tutela della privacy. In *e-privacy 2011 a Firenze*, Giugno 2011.

[32] National Institute of Standards and Technology. Advanced encryption standard (AES). *Federal Information Processing Standards Publication*, Novembre 2001.

[33] Vincent Rijmen Joan Daemen. Rijndael specification. Aprile 2003.

[34] The Internet Society. RFC 2045: Format of Internet Message Bodies. 1996.

[35] Dropbox official website. `http://www.dropbox.com/`. Acceduto il 28 Settembre 2011.

[36] Amazon Web Services. Client-side data encryption with the AWS SDK for Java and Amazon S3. Aprile 2011.