FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica Applicata, primo livello

Elaborato finale

Schemi per Network Coding Sicuro basati su RSA

Candidato:
Daniele Licitra

Relatore:
Chiar.mo Prof. Dario Catalano

Anno Accademico 2009–2010

Sommario

Nelle prossime pagine si approfondirà la tematica del Network Coding, delle firme digitali omomorfiche e del progetto a loro correlato.

L'argomento del Network Coding è poco noto al grande pubblico, ma i benefici che esso apporta al networking sono tali da motivarne la scelta nell'ambito di questa tesi di laurea. Inoltre non possono essere ignorate le problematiche che sorgono e le soluzioni ad esse correlate, quali i due schemi di firma omomorfici che sono stati implementati nel progetto software allegato a questo elaborato.

Il testo è stato suddiviso in capitoli per renderne la consultazione più pratica e veloce.
Nel primo capitolo si presenta il Network Coding, si descrivono alcuni schemi ad esso riferiti e il problema del Pollution Attack.
L'utilizzo di notazione matematica e termini mutuati dal mondo della crittografia rende poco agevole la comprensione del testo a chi non ha conoscenze basilari riguardanti questi ambiti. A costoro si consiglia di leggere il secondo capitolo che illustra la notazione matematica utilizzata e fornisce alcuni cenni riguardanti gli strumenti crittografici (ponendo particolare attenzione sull'algoritmo RSA).
Il terzo capitolo presenta gli schemi di firma omomorfici basati su RSA che sono stati implementati (tramite librerie C++) nel progetto software.
L' illustrazione delle varie componenti in cui è suddiviso il progetto software, le scelte implementative effettuate e l'analisi di questi strumenti è trattata nel quarto capitolo.
Si rimanda all'ultimo capitolo per le conclusioni .

Indice

Capitolo 1

Network Coding

Il termine *Network Coding* si riferisce a un metodo generico di routing dove i nodi intermedi modificano i pacchetti di dati in transito.
In una rete un nodo può avere a disposizione più collegamenti in uscita, l'insieme dei metodi e delle procedure attuate per scegliere su quale collegamento inviare le informazioni è il processo di routing (o instradamento).

Il modello di routing più noto è senz'altro lo *Store-and-Forward*, dove i nodi intermedi si limitano a memorizzare e ad inoltrare i pacchetti di dati. In questo modello, il nodo sorgente dividerà il messaggio in pacchetti e invierà quest'ultimi attraverso la rete. Un nodo intermedio che riceve un pacchetto lo memorizzerà in una coda in attesa di verificarne il destinatario. Se il nodo risulta essere il destinatario finale preleverà il pacchetto dalla rete in attesa di essere in possesso di tutte le informazioni per ricostruire il messaggio, altrimenti il messaggio verrà inoltrato sul collegamento in uscita verso il destinatario o verso un altro nodo intermedio. Il nodo intermedio saprà qual è il collegameto migliore su cui inviare il messaggio tramite un algoritmo di routing.

Il *Network Coding* contrasta con lo Store-and-Forward nell'attività dei nodi intermedi.
In una rete che utilizza il Network Coding i nodi intermedi, ricevuti più pacchetti riguardanti il medesimo messaggio, applicheranno delle modifiche per combinare insieme le informazioni contenute nei singoli pacchetti.
Questa tecnica sta ricevendo sempre più attenzione per le sue potenzialità nell'aumentare la quantità di informazioni inviate attraverso la rete (throughput) e per la sua capacità di rendere più resistenti le reti prive di controllo

centralizzato.
Combinando insieme i dati di più pacchetti del medesimo messaggio, si inviano più informazioni con un singolo pacchetto. Il numero di pacchetti in transito per messaggio risulta essere ridotto rispetto al numero di pacchetti necessari nello Store-and-Forward, quindi questo risparmio può essere utilizzato per l'invio di altre informazioni. Un esempio pratico dell'utilizzo del Network Coding per incrementare il throughput di una rete wireless è l'architettura COPE, proposta in [9], che sfrutta lo XOR di pacchetti per limitarne il numero in circolazione attraverso la rete. I suoi ideatori hanno dimostrato che il throughput di una rete che utilizza il protocollo TCP aumenta in media del 38%.
Il Network Coding risulta essere anche una valida alternativa al routing multicast con approccio decentralizzato. Per routing *multicast* si intende l'invio delle medesime informazioni a più destinatari. Un classico esempio di trasmissione multicast è la televisione via cavo.

Il Network Coding è stato proposto per applicazioni su reti wireless o ad-hoc, dove non sempre si ha a disposizione un dispositivo di controllo centralizzato e gli errori di comunicazione sono elevati.
In questo campo la ricerca progredisce anche grazie agli studi della *Defense Advanced Research Project Agency* (Darpa) e di alcune tra le più prestigiose università americane che hanno sviluppato questo metodo per la trasmissione wireless dei dati tra veicoli militari al fine di creare reti autoconfiguranti affidabili di nodi wireless mobili con un limitato utilizzo di banda. Il loro lavoro si incentra sui progetti ITMANET[1](Information Theory for Mobile Ad-Hoc Networks) e IAMANET(Intrinsically Assurable Mobile Ad hoc Network). Il primo è un programma di 5 anni fondato dalla DARPA per lo sviluppo di una teoria dell'informazione avanzata, orientata alle reti wireless di nodi mobili. Il secondo è la definizione di una rete ad-hoc affidabile che include nella sua definizione integrità, affidabilità, confidenzialità e sicurezza (per maggiori dettagli si rimanda a [10]).

Un'altra importante applicazione sono le reti peer-to-peer (P2P). Un esempio di applicazione P2P che sfrutta il network coding è *Avalanche*[2]di Microsoft, che considera i file come combinazione lineare di vettori.

Con questo sistema non solo si riduce il traffico sulla rete, ma in alcuni

[1]http://www.darpa.mil/Our_Work/I2O/Programs/Information_Theory_for_Mobile_Ad-Hoc_Networks_%28ITMANET%29.aspx
[2]http://research.microsoft.com/en-us/projects/avalanche/default.aspx

schemi il destinatario può anche ricostruire il file originale ricevendo solo parte dei pacchetti. Quindi questo metodo aumenta l'affidabilità di reti per definizione poco affidabili, infatti le reti wireless sono altamente suscettibili ad errori di trasmissione. Si riesce anche a tener traccia di chi ha manipolato il pacchetto, per esempio nelle applicazioni militari, ricevendone uno, si hanno informazioni sull'attività delle stazioni nascoste (quindi si può sapere se un veicolo è in una zona d'ombra o se è stato distrutto).

Consideriamo una rete dove un nodo *sorgente* possiede delle informazioni rappresentate da un file che vuole inviare ad un insieme di nodi *destinatari*. La sorgente suddivide il file in pacchetti che vengono trasmessi attraverso i collegamenti in uscita ai nodi *vicini*. I pacchetti sono visti come vettori di uno spazio lineare su un qualche campo e sono incrementati con delle *coding coordinates* che codificano la storia di tutte le combinazioni lineari avvenute su di esso.
I nodi intermedi calcolano un pacchetto come combinazione lineare dei vettori che ricevono sulle linee in entrata e lo inviano sulle proprie linee in uscita. Questi nodi accorpano più pacchetti per formarne uno solo, così da ridurne il numero in transito ed espandere le dimensioni di quelli che hanno dimensioni ridotte sino al limite superiore gestibile dalla rete.
Il nodo destinatario raccoglierà i pacchetti, effettuerà gli opportuni calcoli e ricostruirà il file originale senza conoscere l'operato dei nodi intermedi. Dall'insieme dei vettori ricevuti, si calcola una matrice che, tramite la sua inversione, produce il file originale.

Un importante fattore che ci permette di valutare la bontà di un protocollo di Network Coding è la *probabilità di decodifica*, ovvero la probabilità che ha un nodo destinatario di ricostruire il file originale. In letteratura molti studi affermano che campi di piccole dimensioni, come 256, offrono alte probabilità di decodifica in reti con un numero sufficiente di connessioni.

1.1 Descrizione dello schema base

Un nodo sorgente S deve inviare un file $\bar{F}$, a cui è associato un identificativo fid, al nodo o ai nodi destinatari T.
Il file $\bar{F}$ è visto come una sequenza ordinata di m vettori n-dimensionali $\vec{v}^{(1)}, ..., \vec{v}^{(m)} \in \mathbb{F}^n$, dove $\mathbb{F}$ è un qualche campo finito. I pacchetti di dati

sono quindi rappresentati dai vettori n-dimensionali.

Prima dell'inizio della trasmissione la sorgente S crea gli m *augmented vectors* $\vec{w}^{(1)}, ..., \vec{w}^{(m)}$ dati da:

$$\vec{w}^{(i)} = (\overbrace{\underbrace{0, .., 0, 1}_{i}, 0, ..., 0}^{m} || \vec{v}^{(m)}) \in \mathbb{F}^{n+m};$$

Ad ogni vettore originale $\vec{v}^{(i)}$ viene anteposto un vettore m-dimensionale contenente il valore 1 alla posizione i-esima e zero in tutte le altre, ovvero una stringa binaria rappresentante il valore 2^{m-i} che viene chiamato *vettore unitario*.
Dunque i vettori $\vec{w}^{(1)}, ..., \vec{w}^{(m)}$ così creati vengono inviati ai nodi vicini.

Il nodo intermedio I riceve i pacchetti $\vec{w}^{(1)}, ..., \vec{w}^{(l)} \in \mathbb{F}^{n+m}$ attraverso i suoi l collegamenti in entrata. Il nodo calcola poi un vettore $\vec{w}$ che invia attraverso i suoi collegamenti in uscita. Questo viene calcolato come

$$\vec{w} = \sum_{i=1}^{l} \alpha_i \vec{w}^{(i)},$$

con $\alpha_i \in \mathbb{F}$ scalare.
Il vettore trasmesso $\vec{w}$ si definisce *corretto* se ricade nello spazio generato (ovvero è combinazione lineare) dei vettori originali incrementati. Se tutti i nodi operano onestamente, allora ogni pacchetto inviato in rete sarà corretto.

A seconda di come vengono scelti i coefficenti $\{\alpha^{(i)}\}_{i=1}^{l}$ da ogni nodo intermedio abbiamo schemi di Network Coding diversi. Nel caso base i coefficenti sono scelti da un autorità centrale per ogni nodo. Quando i coefficenti sono scelti in maniera casuale e indipendente da ogni nodo intermedio, si parla di *Random Linear Network Coding*.

La destinazione, per ricostruire il file originale, deve ricevere m vettori validi $\{\vec{w}^{(i)} = (\vec{u}^{(i)} || \vec{v}^{(i)})\}_{i=1}^{m}$. Da ogni vettore ricevuto, il nodo calcola la matrice U le cui righe sono date dai vettori $\vec{u}^{(i)}$ e una matrice V dove l'i-esima riga è costituita dal vettore $\vec{v}^{(i)}$. La ricostruzione avviene tramite l'operazione:

$$\bar{V} = U^{-1}V$$

dove $\bar{V}$ è la matrice che ha per righe i vettori originali $\vec{v}^{(1)}, ...\vec{v}^{(m)}$.

L'overhead per file equivale a $\theta(m^2)$, ovvero m bit per m vettori. Spesso si sceglie $m \ll n$.

La probabilità di decodifica, quando i coefficenti sono scelti in modo casuale e indipendente dai singoli nodi intermedi, è determinata dalla topologia della rete e dalla dimensione del campo $\mathbb{F}$. Per minimizzare l'overhead è desiderabile mantenere la dimensione di $\mathbb{F}$ più piccola possibile. D'altrocanto, ridurre troppo la dimensione di $\mathbb{F}$ potrebbe diminuire notevolmente la probabilità di decodifica. Nella pratica, le reti usano di solito campi la cui dimensione è di circa 256 e la loro probabilità di decodifica è superiore al 99%, come indicato in [1].

Questo schema è tuttavia soggetto ad un attacco tanto banale quanto efficace, il *pollution attack*.

1.2 Pollution Attack

In questo tipo di attacco, un nodo corrotto o controllato da un avversario (chiamato spesso *Byzantine node*) può inserire pacchetti erronei nella rete. I nodi destinatari che ricevono il pacchetto errato, non riusciranno a ricostruire l'informazione iniziale.
L'errore introdotto in un singolo pacchetto può essere propagato ad altri pacchetti durante il percorso verso la destinazione tramite le modifiche applicate dai nodi intermedi inconsciamente. I nodi onesti, se non hanno strumenti per distinguere un vettore corretto da uno erroneo, combineranno i vettori tra loro secondo lo schema, ma il vettore che produrranno sarà errato e intaccherà la correttezza dei vettori con cui verrà combinato dagli altri nodi.
Questo attacco può essere classificato come DoS (Denial of Service), ovvero diniego di servizio. Questi tipi di attacchi rallentano o impediscono le comunicazioni. Infatti i nodi saranno impegnati anche nel calcolare pacchetti erronei e la destinazione che non riesce a ricostruire l'informazione chiederà nuovamente l'invio dei dati, aumentando il traffico sulla rete fino al possibile collasso della stessa.
Gli schemi di network coding di base non contengono sistemi di isolamento delle falle nel sistema, quindi se uno dei vettori $\vec{w}^{(i)}$ che la destinazione riceve è errato, l'errore si propaga a tutto il file ricostruito e può essere rilevato soltanto quando si è in possesso del file intero.
Per ridurre i danni che questo attacco può arrecare bisogna avere uno strumento che possa dirci se i vettori ottenuti sono corretti o manomessi, quali:

le firme digitali e le funzioni hash.

1.3 Schemi di firma per Network Coding

Nel Network Coding, le firme sono un utile strumento per limitare i danni subiti da un pollution attack. Anche un singolo vettore erroneo può causare una errata ricostruzione del file originale. Porre la firma su un pacchetto (o vettore) ci permette di capire se esso è corretto. Infatti con le firme digitali, solo il nodo sorgente può apporre una firma valida utilizzando la propria chiave privata, mentre gli altri nodi potranno utilizzare la chiave pubblica per verificarne la correttezza.

Nel sistema più semplice il nodo sorgente applica la firma con la sua chiave privata a tutto il file, lo scompone in vettori e lo invia. La destinazione riceve i vettori, ricostruisce il file e ne verifica l'autenticità e correttezza tramite la chiave pubblica della sorgente. I nodi intermedi, non avendo tutti i vettori, non potranno verificare l'autenticità delle informazioni che stanno trattando, un vettore erroneo non può essere fermato e quindi questo metodo non è sufficiente.

Dare la possibilità ad ogni nodo intermedio di poter firmare e autenticare il pacchetto in transito, sia tramite firma digitale che tramite Message Authentication Code, potrebbe risolvere il problema ma ridurrebbe drasticamente le prestazioni dello schema. Infatti ogni nodo dovrebbe impegnare risorse per verificare e autenticare i pacchetti in transito. Inoltre nello schema simmetrico, tutti i nodi della rete dovrebbero condividere la stessa chiave privata, il che spesso non è possibile perchè non tutti i nodi godono della stessa fiducia.
Nello schema asimmetrico invece ogni nodo dovrebbe conoscere le chiavi pubbliche di tutti i nodi vicini da cui riceve trasmissioni in entrata. Per ogni nodo dovrebbe prima verificare la veridicità del pacchetto tramite la chiave pubblica, poi firmare i pacchetti con la propria chiave privata ed inviarli sui collegamenti in uscita. Ma le primitive asimmetriche impiegano più risorse di quelle simmetriche, quindi i nodi intermedi utilizzerebbero molte risorse dilatando i tempi di trasmissione e si potrebbero verificare problemi con le code dei messaggi, che si riempiono in caso di nodi più lenti rispetto ad altri.

Da quanto detto finora, gli strumenti classici non garantiscono buone prestazioni in termini di integrità e di tempi di esecuzione; servono strumenti nuovi che possano garantire integrità senza intaccare i tempi di esecuzione. Le ultime ricerche hanno sviluppato strumenti crittografici il cui unico vincolo alla dimostrazione di sicurezza è dato dalla limitazione delle risorse che l'avversario ha a disposizione. Questi approcci utilizzano le firme digitali e permettono a chiunque di possedere la chiave pubblica della sorgente e quindi ogni nodo intermedio può verificare la correttezza dei vettori.

Possiamo suddividere gli schemi di firma per il Network Coding in due categorie: gli schemi che si basano sull'hash omomorfico e quelli che usano le firme omomorfiche.

Schemi basati sull'hashing omomorfico. Una funzione omomorfica H è una funzione hash resistente alle collisioni con la seguente proprietà: per ogni vettore a e b e scalari α e β, si ha che $H(\alpha a + \beta b) = H(a)^{\alpha} H(b)^{\beta}$. La resistenza alle collisioni ci assicura che, dati dei vettori a, b e c per cui $H(c) = H(a)^{\alpha} H(b)^{\beta}$, allora $c = \alpha a + \alpha b$.

Un esempio concreto di hashing omomorfico è dato dallo schema *Exponential Homomorphic Hash (EHH)*. Sia $\mathbb{G}$ un gruppo ciclico di ordine p e supponiamo che la chiave pubblica contenga una descrizione di $\mathbb{G}$ su dei generatori $g_1, ..., g_n \in \mathbb{G}$. Definiamo una funzione H di vettori $v = (v_1, ..., v_n) \in \mathbb{Z}_p^n$ come :

$$H(v) = \prod_{j=1}^{n} g_j^{v_j}$$

La proprietà omomorfica è facilmente verificata mentre, la resistenza alle collisioni è garantita dalle assunzioni del logaritmo discreto in $\mathbb{G}$.

Le funzioni hash omomorfiche nel Network Coding sono usate nel modo seguente: per ogni vettore $\vec{v}^{(i)}$, la sorgente S calcola $h_i = H(\vec{v}^{(i)})$; dopo firma, mediante uno schema di firma standard, i valori $h_1, ..., h_m$ utilizzando anche l'identificativo del file fid. I valori h_i e le loro firme sono concatenati ad ogni pacchetto inviato sulla rete. Un nodo può determinare se il vettore $w = (u||v)$ è corretto verificando la firma sui valori $\{h_i\}$ e verificando se $\prod_{i=1}^{m} h_i^{u_i} = H(v)$.

Nel caso particolare di EHH, il controllo avviene verificando se

$$\prod_{i=1}^{m} h_i^{u^{(i)}} = H(v) \stackrel{def}{=} \prod_{j=1}^{n} g_j^{v_j}.$$

La sicurezza di questa categoria di schemi può essere provata basandosi sull'assunzione della difficoltà del logaritmo discreto. Usando questi schemi, l'unica aggiunta allo schema base è che un nodo intermedio deve verificare la correttezza dei vettori che riceve prima di operare su di essi e che il campo utilizzato è $\mathbb{F} = \mathbb{Z}_n$.

Schemi basati sulle firme omomorfiche. Gli schemi di firma omomorfici hanno la proprietà che, per ogni vettore $\vec{a}$ e $\vec{b}$ e scalari α e β, si ha $Sign(\alpha a + \beta b) = Sign(a)^{\alpha} Sign(b)^{\beta}$. La sorgente firma ogni vettore $\vec{w}^{(i)}$ e lo invia insieme alla sua firma $Sign(\vec{w}^{(i)})$. Un nodo intermedio che riceve i vettori, verifica la firma utilizzando la chiave pubblica, scarta i vettori errati e calcola una firma valida su ogni vettore in uscita utilizzando la proprietà omomorfica. Dato che la verifica della firma sarà svolta da ogni nodo, questa operazione deve essere ben progettata in quanto potrebbe costituire il collo di bottiglia del sistema.
Un esempio concreto di questi schemi è BFKW presentato in [2].

1.3.1 Definizioni formali

Definizione 1. *Gli schemi di firma per Network Coding sono costituiti da una terna di algoritmi randomizzati, eseguibili in tempo polinomiale,* (Setup,Sign,Vrfy) *definiti come:*

- *Setup*$(1^k, N)$ *: Dati i parametri* 1^k *e* N*, rispettivamente un parametro di sicurezza e la lunghezza dei vettori da firmare, l'algoritmo restituisce in output una chiave pubblica* PK *e una chiave privata* SK*. Inoltre definisce un campo* $\mathbb{F}$*.*

- *Sign*(SK, fid, V)*: L'algoritmo riceve in input la chiave privata* SK*, l'identificativo del file* $fid \in \{0,1\}^*$ *e un sottospazio* $V \subset \mathbb{F}^N$ m*-dimensionale, con* $0 < m \leq N$*, composto dai vettori di base* $v_1, ..., v_m$ *.Ritorna la firma* σ*.*

- $Vrfy(PK, fid, y, \sigma)$ *: Data la chiave pubblica* PK*, l'identificativo del file* fid*, un vettore* $y \in \mathbb{F}^N$ *e la firma* σ*, l'algoritmo ritorna 0 (rifiuta) o 1 (accetta).*

Definizione 2. *Sia* $S = (Setup, Sign, Vrfy)$ *uno schema di firma per Network Coding. Sia* A *un avversario polinomialmente limitato che conosce lo schema ed ha accesso ad un oracolo di firma. Sia* k *un parametro di sicurezza e* V_i *un sottospazio di* $\mathbb{F}_p^N$*. Definiamo l'esperimento:*

Algorithm 1 $Esp_S(A)$

$N \leftarrow_R \mathbb{N}$;
$(p, Sk, Pk) \leftarrow Setup(1^k, N)$;
$id_i \leftarrow_R I; V_i \subset \mathbb{F}_p^N$;
$\sigma_i \leftarrow Sign(Sk, id_i, V_i)$;
$(id^*, \sigma^*, y^*) \leftarrow A^{Sign(Sk,\ )}(Pk)$;
if $Vrfy(Pk, id^*, y^*, \sigma^*) = 1$**and** $((id_i \neq id^*\ \forall i$**and** $y^* \neq 0)$**or** $(id_i = id^*$**and** $y^* \notin V_i))$ **then**
    return 1;
else
    return 0;
end if

Definiamo il vantaggio come $Adv(A) = Pr[Esp_S(A) = 1]$*. Diciamo che lo schema* S *è* sicuro *se ogni avversario polinomialmente limitato ha vantaggio prossimo a zero.*

Definizione 3. *Gli schemi di firma omomorfici per Network Coding sono costituiti da una quadrupla di algoritmi randomizzati, eseguiti in tempo polinomiale,* (Setup,Sign,Combine,Vrfy) *definiti come:*

- $Setup(1^k, N)$ *: Dati i parametri* 1^k *e* N*, rispettivamente un parametro di sicurezza e la lunghezza dei vettori da firmare, l'algoritmo restituisce in output una chiave pubblica* PK *e una chiave privata* SK*. Inoltre definisce un campo* $\mathbb{F}$*.*

- $Sign(SK, fid, v)$*: L'algoritmo riceve in input la chiave privata* SK*, l'identificativo del file* $fid \in \{0,1\}^*$ *e un vettore* $v \in \mathbb{F}^N$ *e ritorna la firma* σ*.*

- $Combine(PK, fid, \{(\alpha_i, \sigma_i)\}_{i=1}^l)$ *: Data una chiave pubblica* PK*, l'identificativo del file* fid *e* l *coppie formate da uno scalare* $\alpha_i \in \mathbb{F}$ *e una firma* σ_i*, l'algoritmo ritorna una firma* σ*.*

- $Vrfy(PK, fid, y, \sigma)$ *: Data la chiave pubblica* PK*, l'identificativo del file* fid*, un vettore* $y \in \mathbb{F}^N$ *e la firma* σ*, l'algoritmo ritorna 0 (rifiuta) o 1 (accetta).*

Lemma 1. [3] *Sia* $(Setup_2, Sign_2, Combine_2, Vrfy_2)$ *uno schema di firma omomorfico per network coding. Allora* $(Setup_1, Sign_1, Vrfy_1)$*, definito come segue, è uno schema di firma per network coding.*

- $Setup_1(1^k, N) = Setup_2(1^k, N)$.
- $Sign_1(SK, fid, V) = (Sign_2(SK, fid, v^{(1)}), ..., Sign_2(SK, fid, v^{(m)}))$, *dove* $v^{(1)}, ..., v^{(m)}$ *sono augmented vectors. Definiamo la firma come* $\sigma = (\sigma_1, ..., \sigma_m)$.
- $Vrfy_1(PK, fid, y, \sigma) = Vrfy_2(PK, fid, \sigma, Combine_2(PK, fid, \{(y_{N-m+i}, \sigma_i)\}_{i=1}^m))$.

Definizione 4. *Uno schema di firma omomorfico per Network Coding si definisce sicuro se lo schema di firma per network coding definito secondo il lemma 1 è sicuro.*

1.4 Network Coding sugli interi

Utilizzando strumenti crittografici, spesso le operazioni vengono effettuate modulo p, dove p è un numero primo di k bit. Questo comporta che ogni vettore iniziale viene incrementato con m coordinate, ciascuna grande almeno k bit, aumentando l'overhead.
Il Network Coding basato sugli interi si basa sulla scelta di coefficenti di piccole dimensioni, per esempio 8 bit, al fine di ridurre il tempo di calcolo e la dimensione dei vettori inviati. Gennaro, Katz, Krawczyk e Rabin dimostrano in [1] che questa scelta migliora le performance senza intaccare la probabilità di decodifica e la sicurezza degli strumenti crittografici.

Gli schemi tradizionali di network coding possono essere modificati per operare sugli interi nel seguente modo. Il file originale $\bar{F}$ viene codificato come un insieme di vettori $\vec{v}^{(1)}, ..., \vec{v}^{(m)}$ i cui elementi sono degli interi. Questi vettori poi vengono incrementati con i vettori unitari $\vec{u}^{(1)}, ..., \vec{u}^{(m)}$ come nello schema base. Le combinazioni lineari svolte sui vettori dai nodi intermedi verranno calcolate utilizzano scalari casuali α_i scelti da un insieme $Q = \{0, ..., q-1\}$ per un primo q, come 257. Queste operazioni vengono

[3] Per la dimostrazione del lemma si rimanda a [2]

svolte sugli interi e non *modulo q*.

La definizione di probabilità di decodifica nel caso di network coding sugli interi è uguale a quella del caso base, come dimostra il seguente lemma che riporto per completezza da [1]:

Lemma 2. *Per ogni rete, la probabilità di decodifica quando i coefficenti α_i sono scelti in $Q = \{0, ..., q-1\}$, con q primo, e tutte le combinazioni lineari sono effettuate sugli interi, non è peggiore della probabilità di decodifica della stessa rete usando uno schema di network coding tradizionale su un campo di dimensione q.*

Dimostrazione. Fissiamo una sequenza di coefficenti α_i scelti dai nodi durante il protocollo di network coding. Dobbiamo dimostrare che se i valori α_i formano una matrice U invertibile modulo q alla destinazione, allora lo stesso insieme di α_i, utilizzato in combinazioni lineari sugli interi, darà una matrice U invertibile in $\mathbb{Z}$.
Questa affermazione segue direttamente dal fatto che se $det(U) \neq 0$ modulo q, allora $det(U) \neq 0$ anche sugli interi, quindi U è invertibile. Basta scegliere q in modo che la probabilità di decodifica, operando modulo q, sia sufficientemente alta per avere una buona probabilità di decodifica quando si usano combinazioni lineari sugli interi. In molte applizioni pratiche un campo a 8 bit è sufficente a dare una buona probabilità di decodifica. □

La dimensione delle coordinate di ogni vettore trasmesso nella rete aumentano ad ogni *hop* (passaggio da un nodo ad un altro) se non si effettua nessun calcolo modulare. Ogni salto aumenta il maggior elemento del vettore di un fattore al massimo $min\{mq, lq\}$, dove l è il numero di collegamenti in entrata di un nodo. Dopo L passaggi, le prime m coordinate avranno dimensione al massimo $(mq)^L$ mentre le rimanenti coordinate avranno dimensione $M(mq)^L$, dove con M si indica la massima dimensione delle coordinate nei vettori iniziali. L'incremento in bit si ottiene prendendo il logaritmo dei numeri sopracitati, quindi minore è la dimensione dei coefficenti, minore è il loro impatto sulla dimensione dei dati trasmessi e sugli schemi di firma.
Un modo per raggiungere l'efficenza in termini di bandwidth è rappresentare gli elementi dei vettori con lo stesso numero di bit, utilizzando il numero di bit necessario per rappresentare l'elemento maggiore, ed accodare un valore indicante la lunghezza dell'elemento al vettore. Combinando i vettori, anche l'indicatore della lunghezza crescerà.

1.5 Network Coding ibrido

Una variante ibrida di Network Coding utilizza piccoli coefficenti interi e svolge le combinazioni lineari modulo p, dove p è un numero primo di k bit. Anche questa variante dello schema base è stata discussa da Gennaro, Katz, Krawczyk e Rabin in [1].
In questa variante i vettori originali $\vec{v}^{(1)}, ..., \vec{v}^{(m)}$ trasmessi dalla sorgente appartengono a $\mathbb{Z}_p^n$. I coefficenti utilizzati nelle combinazioni lineari sono scelti in $Q = \{0, ..., q-1\}$, per un numero primo q di piccole dimensioni ($q \approx 256$) e le combinazioni lineari sono eseguite modulo p.
Per gli effetti che hanno queste scelte sulla probabilità di decodifica si rimanda al lemma seguente.
Si considera il caso in cui i vettori da inviare siano m e che la lunghezza del cammino massimo tra la sorgente e la destinazione sia L, con $m \ll 2^k$ e $L \ll 2^k$ dove k è la lunghezza del primo p.

Lemma 3. *Per ogni rete, la probabilità di decodifica quando i coefficenti α_i sono scelti in $Q = \{0, ..., q-1\}$, con q primo, e tutte le combinazioni lineari sono effettuate modulo p, con p primo casuale di k-bit, è, a differenza di un fattore $O(Lm \log q/2^k)$, uguale o migliore della probabilità di decodifica di uno schema di network coding standard usato su un campo di dimensione q nella stessa rete.*

Dimostrazione. Sia U una matrice ricostruita dalla destinazione. Dato che le operazioni sono svolte modulo p, gli elementi della matrice appartengono a $\mathbb{Z}_p$. Denotiamo con U^* la matrice ricostruita dalla destinazione nel caso invece di operazioni sugli interi senza effettuare operazioni modulo p, quindi avremo la corrispondenza $U = U^* mod\ p$. Si ha che $det(U) = 0\ mod\ p$ se e solo se $det(U^*) = 0$ o $det(U^*) \neq 0$ ma p è un divisore di $det(U^*)$. Il caso in cui $det(U^*) = 0$ è stato trattato nel Lemma 1, dobbiamo calcolare la probabilità del secondo caso.
Indichiamo con d la lunghezza in bit di $det(U^*)$. I numeri primi di lunghezza k che dividono $det(U^*)$ sono massimo $\frac{d}{k}$ e il numero totale di primi di lunghezza k sono $\frac{2^k}{k}$. Quindi la probabilità di scegliere p in modo che divida $det(U^*)$ è al massimo $\frac{d}{2^k}$.
Ora bisogna dimostrare che d sia trascurabile relativamente a 2^k. La matrice U^* è formata dai vettore unitari ricevuti dalla destinazione. Gli elementi del vettore, attraversando L salti, saranno interi $\leq (mq)^L$. Quindi U^* è una matrice $m \times m$ con ogni elemento $\leq (mq)^L$ e $det(U^*) \leq m!(mq)^{Lm} \leq$

$(mq)^{m(L+1)}$ e $d = |det(U^*)| \leq m(L+1)(logm + logq)$ che è trascurabile rispetto a 2^k. □

Il risparmio si nota subito in un minore carico della rete in reti relativamente piccole e in un minore peso degli strumenti crittografici sulla comunicazione. Se ogni nodo aggiunge l bit ai vettori e abbiamo un cammino di k/l salti o più, il risparmio di rete sui primi k/l salti sarà di $k/2 \cdot k/l \cdot m$. Per piccole reti il risparmio in numero di bit sarà minore, ma maggiore in rapporto alla bandwidth totale.
Inoltre u_i più brevi comporteranno esponenti più brevi nelle formule di verifica nei protocolli basati su hashing omomorfico, mentre negli schemi basati su firma digitale omomorfica il risparmio si trasmette anche a tutte le operazioni in cui sono coinvolti i coefficenti α_i e, considerando che ogni nodo deve calcolare le firme (sia come verifica che come combinazione), questo porta notevoli vantaggi computazionali.

1.6 Linear Network Coding

Questo schema di codifica è stato definito da Li, Yeung e Cai in [11] e ripreso da Boneh, Freeman, Katz e Waters in [2].
In esso il file trasmesso è visto come una sequenza ordinata di m vettori n-dimensionali $\vec{v}^{(1)}, ..., \vec{v}^{(m)} \in \mathbb{F}_p^n$, con p primo. Nei documenti appena citati si usa creare i vettori $w^{(i)}$ attraverso la contatenazione $(\vec{v}^{(i)}||\vec{u}^{(i)})$, qui si manterrà la notazione utilizzata nello schema base, ovvero prependendo il vettore unitario al vettore originale. Le due notazioni sono equivalenti ed è semplice trasformare le operazioni per operare secondo l'una o l'altra definizione.

Prima dell'inizio della trasmissione, la sorgente crea gli augmented vectors dati da: $\vec{w}^{(i)} = (\vec{u}^{(i)}||\vec{v}^{(i)})$. L'overhead per file equivale a $\theta(m^2)$, ovvero m bit per m vettori anche in questo caso. Spesso si sceglie $m \ll n$.

I nodi intermedi ricevono i vettori $\vec{w}^{(1)}, ..., \vec{w}^{(l)} \in \mathbb{F}_p^{n+m}$, dove l è il numero di connessioni in entrata del nodo. Il nodo i-esimo calcola il vettore

$$\vec{w} = \sum_{j=1}^{l} \alpha_{ij} w^{(j)} \text{ con } \alpha_{ij} \in \mathbb{F}_p.$$

I valori α_{ij} sono scelti da un'autorità centrale. Il nodo trasmette poi il vettore così calcolato.

Quando il nodo destinatario riceve gli m vettori $\vec{w}^{(i)}$, li scompone in $\vec{w}^{(i)(L)}$, contenenti gli m elementi sinistri di $\vec{w}^{(i)}$, e in $\vec{w}^{(i)(R)}$, contenenti gli n elementi destri. Poi il nodo calcola la matrice G come l'inversa della matrice dove l'i-esima riga è data dal vettore $\vec{w}^{(i)(L)}$.

$$G = \begin{pmatrix} w^{(1)(L)} \\ ... \\ w^{(m)(L)} \end{pmatrix}^{-1}$$

Successivamente il nodo moltiplica la matrice così ottenuta per la matrice che ha per righe i vettori $\vec{w}^{(i)(R)}$ e ottiene il file originale.

$$\begin{pmatrix} v^{(1)} \\ ... \\ v^{(m)} \end{pmatrix} = G \cdot \begin{pmatrix} w^{(1)(R)} \\ ... \\ w^{(m)(R)} \end{pmatrix}$$

Da notare il fatto che il nodo destinazione non ha bisogno di conoscere i valori degli α_{ij} usati dai nodi intermedi per ricostruire il file. D'altronde, se si conoscono i valori α_{ij} e la topologia della rete, la destinazione può calcolare in anticipo la matrice G.

1.6.1 Random Linear Network Coding

Il funzionamento è analogo a quello della Linear Network Coding, solo che i pesi α_{ij} sono scelti casualmente nel campo $\mathbb{F}_p^{n+m}$ e indipendentemente dai singoli nodi. Questa soluzione è applicata nelle reti completamente decentralizzate, dove non esiste un nodo riconosciuto come autorità di controllo che possa stabilire dei pesi univoci.
Per ottenere il messaggio originale è sufficente premettere ad ogni messaggio m coordinate di codifica che compattano la storia delle combinazioni lineari attuate dai vari nodi intermedi sul vettore preso in esame. La destinazione usa le coordinate per creare una matrice $m \times m$ e, come nel Linear Network Coding, calcola i vettori originali.

In [3] è stato dimostrato, nel caso di rectangular grid network, che la probabilità che tutte le destinazioni riescano a decodificare il messaggio è almeno $(1 - d/q)^v$ per $q > d$, dove d è il numero di nodi destinatari, v è il numero massimo di collegamenti che ricevono il messaggio e i messaggi sono scelti da un campo finito $\mathbb{F}_q$. Il limite è inversamente proporzionale alla

dimensione del campo, quindi la probabilità di successo è elevata in campi sufficentemente grandi.

Capitolo 2

Nozioni e definizioni

Questa sezione contiene tutte le nozioni matematiche che aiuteranno nella comprensione del testo.

2.1 Matrici

Dati $m \cdot n$ numeri a_{ij}, se si dispongono in una tabella di m righe ed n colonne nel modo seguente:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mn} \end{pmatrix}$$

si dice che essi formano una *matrice* A $m \times n$.
Si dice *complemento algebrico* e lo si indica con A_{ij}, il determinante della matrice che ha come righe, le righe di A esclusa la riga i, e come colonne, le colonne di A esclusa la colonna j, moltiplicato per $(-1)^{i+j}$.
Una matrice si dice *quadrata di ordine* m se $m = n$. Di una matrice quadrata si calcola il determinante.
Il *determinante* della matrice A è il numero $\det(A) = a_{i1}A_{i1} + a_{i2}A_{i2} + ... + a_{im}A_{im}$ o $\det(A) = a_{1i}A_{1i} + a_{2i}A_{2i} + ... + a_{mi}A_{mi}$ con $i = 1, ..., m$.
Una matrice A è *invertibile* se $\det(A) \neq 0$, la sua inversa A^{-1} è data dagli elementi $a_{ij} = \frac{A_{ji}}{\det(A)}$.

2.2 Numeri primi e primi safe

Un numero p si dice *primo* quando non esiste un intero che divide p in $\mathbb{Z}$ diverso da 1 e p stesso.

Un numero primo γ si dice *primo safe* o *safe-prime* se è della forma $\gamma = (2p+1)$, dove p è un numero primo.

2.3 Gruppi e spazi vettoriali

Un *gruppo* è un insieme G munito di una operazione binaria $*$, che ad ogni coppia di elementi a, b di G associa un elemento, che indichiamo con $a * b$, appartenente a G, rispettando i seguenti assiomi:

- 1. proprietà associativa: dati a, b, c appartenenti a G, vale $(a * b) * c = a * (b * c)$.
- 2. esistenza dell'elemento neutro: esiste in G un elemento neutro e rispetto all'operazione $*$, cioè tale che $a * e = e * a = a$ per ogni a appartenente a G.
- 3. esistenza dell'inverso: ad ogni elemento a di G è associato un elemento b, detto *inverso* di a , tale che $a * b = b * a = e$.

La cardinalità dell'insieme G viene indicata con $|G|$ ed è chiamata *ordine* del gruppo: se questa è finita allora G è un gruppo finito.
Indichiamo con $\mathbb{Z}_N = \{0, 1, ..., N-1\}$ il gruppo additivo commutativo.
Si indica con $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N : gcd(x, N) = 1\}$ un gruppo moltiplicativo.
Si definisce $QR_N \subseteq \mathbb{Z}_N^*$ l'insieme dei *quadrati residui* modulo N come $QR_N = \{x \in \mathbb{Z}_N^* : x = y^2 \ mod \ N, y \in \mathbb{Z}_N^*\}$.

La coppia (V, K) è uno *spazio vettoriale* se sono verificate le seguenti proprietà:

- $(V, +)$ è un gruppo commutativo, ovvero valgono le proprietà associativa e commutativa, esiste un elemento neutro ed ogni elemento ha un simmetrico;
- – $(\alpha + \beta) \cdot \vec{v} = \alpha\vec{v} + \beta\vec{v}, \ \forall \alpha, \beta \in K, \forall \vec{v} \in V$;
  – $\alpha \cdot (\vec{v} + \vec{w}) = \alpha\vec{v} + \alpha\vec{w}, \ \forall \alpha \in K, \forall \vec{v}, \vec{w} \in V$;
  – $\alpha(\beta\vec{v}) = (\alpha\beta)\vec{v} \ \forall \alpha, \beta \in K, \forall \vec{v} \in V$;
  – $1_k\vec{v} = \vec{v}, \ \forall \vec{v} \in V$;

Gli elementi di V si diranno *vettori* mentre gli elementi di K si dicono *scalari.*

$G = \{\vec{g_1}, \vec{g_2}, ..., \vec{g_n}\}$ è un *insieme di generatori* di V, se ogni $\vec{v} \in V$ è combinazione lineare dei vettori $\vec{g_i} \in G$.

2.4 Omomorfismo

Una funzione $f : A \to B$ è omomorfica se vale $f(a * b) = f(a) \sharp f(b)$ dove $*$ e $\sharp$ sono le operazioni binarie di A e B rispettivamente.

2.5 Stringhe

Con il simbolo $\{0,1\}^k$, con $k \in \mathbb{N}$, indichiamo l'insieme di tutte le stringhe composte dai simboli 0 e 1 di lunghezza k.
Con $\{0,1\}^*$ si indica l'insieme di tutte le stringhe binarie di lunghezza arbitraria.
Con il simbolo ε si indica la stringa di lunghezza nulla.
Con il simbolo $||$ si indica l'operatore di *concatenazione* tra due stringhe.

2.6 Cenni crittografici

Di seguito si riportano nozioni per comprendere meglio gli strumenti crittografici.
Nello scenario crittografico il *mittente* S deve inviare un messaggio M ad un *destinatario* R attraverso un canale senza che terzi possano interferire.
Se S e R avessero a disposizione un canale ideale, dedicato, impenetrabile e privo di errori, allora la comunicazione potrebbe avvenire in totale sicurezza. Purtroppo nella realtà questo canale non esiste e le nostre informazioni viaggiano in canali ai quali chiunque ha accesso.

Lo scopo della crittografia è di fornire gli strumenti che possano rendere i canali non affidabili quanto più simili al canale ideale. Per raggiungere questo scopo, la crittografia si pone due obbiettivi principali:

- **privacy**, ovvero mantenere segrete a terzi le informazioni che due parti si trasmettono;
- **integrità ed autentica**, ovvero poter certificare che le informazioni trasmesse siano arrivate integre e senza modifiche e certificarne la provenienza.

Un'*avversario* A è chiunque voglia intercettare o modificare un messaggio scambiato tra S e R o voglia spacciarsi per uno dei due attori.
Gli strumenti crittografici si dividono in due categorie: simmetrici e asimmetrici.

Negli schemi di *cifratura simmetrica*, le parti condividono la medesima chiave di cifratura che utilizzano per la cifratura dei messaggi scambiati. Uno schema di cifratura simmetrico è una tripla di algoritmi $SE = (KeyGen, Enc, Dec)$. Sia M il messaggio da trasmettere, sia K la chiave condivisa e sia $Enc()$ lo schema di cifratura e $Dec()$ lo schema di decifratura, il mittente calcola il crittotesto $C = Enc(M, K)$ e lo invia al destinatario. Il destinatario ottiene il messaggio calcolando $M = Dec(C, K)$.

L'autentica e l'integrità con strumenti simmetrici si ottiene tramite i *Message Authentication Code*. Il mittente e il destinatario condividono la stessa chiave K e vogliono trasmettersi il messaggio M. Indichiamo con $MAC()$ l'algoritmo di autentica. Il mittente calcola il tag di autentica $tag = MAC(M, K)$ e invia la coppia (M, tag) al destinatario. Il destinatario riceve (M', tag), calcola $tag' = MAC(M', K)$ e verifica se $tag = tag'$. I due codici risulteranno uguali solo se $M' = M$, quindi il messaggio è integro. Dato che i codici generati sono uguali solo se le chiavi sono uguali, si ha anche l'autentica, infatti solo chi è in possesso di K può calcolare il corretto tag.

Gli schemi di *cifratura asimmetrica* si dicono tali in quanto il destinatario possiede una coppia di chiavi (Sk, Pk), ovvero una chiave segreta ed una pubblica. Lo schema è dato dalla tripla di algoritmi $AE = (KeyGen, Enc, Dec)$. Il destinatario calcola $(Sk, Pk) = Keygen()$ attraverso l'algoritmo di generazione della chiave. I mittenti in possesso della chiave pubblica Pk del destinatario calcolano $C = Enc(M, Pk)$. Il destinatario riceve il crittotesto e calcola il messaggio originale come $M = Dec(C, Sk)$. Questi schemi si basano sulla difficoltà dell'invertibilità di una funzione, per cui è facile calcolare la funzione Enc su M conoscendo Pk ,ma è difficile calcolare l'inversa Dec non conoscendo Sk.

Con le *firme digitali* si verifica l'autenticità e l'integrità di un messaggio nel contesto asimmetrico. In questo caso è il mittente a possedere la coppia di chiavi (Sk, Pk). Questi schemi sono formati dall'algoritmo di generazione della chiave $KeyGen()$, dall'algoritmo di firma $Sign()$ e dall'algoritmo di verifica $Vrfy()$. Il mittente calcola la coppia di chiavi $(Sk, Pk) = KeyGen()$, pubblica in qualche modo la chiave Pk e mantiene segreta la chiave Sk. Il mittente calcola la firma come $\sigma = Sign(M, Sk)$ e invia la coppia (M, σ) al destinatario. Il destinatario riceve la coppia (M', σ), calcola $Vrfy(Sk, M', \sigma)$ che ritorna vero se e solo se $M = M'$.
Le firme digitali, oltre a garantire integrità ed autenticità, garantiscono anche la non-ripudiabilità. Mentre in MAC, i possessori della chiave segreta

sono multipli e quindi non si può garantire chi delle parti abbia firmato il messaggio, nelle firme digitali solo il possessore di Sk sarà in grado di calcolare σ, quindi il mittente, attraverso la firma, è legato al possesso del messaggio e quindi non può ripudiarne la creazione.

2.6.1 Funzioni hash

Una funzione *hash* è una funzione che comprime una stringa di taglia arbitraria in una di lunghezza fissata. La possiamo indicare come $H : K \times \{0,1\}^* \rightarrow \{0,1\}^k$ dove K è l'insieme degli stati della funzione. Le funzioni hash sono spesso usate per garantire l'integrità delle informazioni. Insieme al messaggio M, il mittente invia anche il suo valore hash $h(M)$. Il destinatario calcolerà il valore hash del messaggio che ha ricevuto e se questo valore è uguale ad $h(M)$ allora il messaggio è integro.
Avendo infiniti input e limitati output, per il principio della piccionaia, stringhe diverse saranno mappate nelle medesime stringhe di output.
Definiamo *collisione* una coppia di stringhe $x_1, x_2 \in \{0,1\}^* : H(x_1) = H(x_2)$ con $x_1 \neq x_2$.
Diciamo che una funzione è di *hash universale* quando per ogni avversario polinomialmente limitato A, la probabilità che scelga x_1 e x_2, senza conoscere k, tali che $H_k(x_1) = H_k(x_2)$, con $x_1 \neq x_2$, è prossima a zero.
Definiamo una funzione hash *resistente alle collisioni* quando non esiste alcun avversario A polinomialmente limitato che, scelto k in modo casuale su K, sia in grado di scegliere x_1 e x_2 tali che $H_k(x_1) = H_k(x_2)$ con $x_1 \neq x_2$.
Una funzione hash si dice *division intractable* quando non esiste un avversario A polinomialmente limitato in grado di scegliere $x_1, x_2, ..., x_n, y$, con $x_i \neq y \ \forall i \in [1, n]$, tali che $H(y)$ sia divisore di $H(x_1)H(x_2)...H(x_n)$.

Una funzione hash H resistente alle collisioni è *omomorfica* se per ogni vettore $\vec{a}, \vec{b}$ e scalari α e β, si ha

$$H(\alpha\vec{a} + \beta\vec{b}) = H(\vec{a})^\alpha H(\vec{b})^\beta$$

Quindi dati i vettori $\vec{a}, \vec{b}$ e $\vec{c}$ con $H(\vec{c}) = H(\vec{a})^\alpha H(\vec{b})^\beta$, si ha $\vec{c} = \alpha\vec{a} + \beta\vec{b}$.

2.6.2 RSA

RSA è uno schema di cifratura asimmetrico che prende il nome dai suoi creatori, Rivest, Shamir e Adelman.

Chiamiamo *modulo RSA* un numero $N = pq$, con p e q primi, mentre indichiamo con e un esponente. (N, e) sono informazioni pubbliche, mentre d è la chiave privata.
Lo schema di codifica RSA è definito come:

$$RSA[N, e](x) = x^e mod\ N$$

RSA è una funzione 'trapdoor', la funzione è facile da calcolare ma difficile da invertire. Se si conosce la chiave segreta però, siamo in grado di effettuare l'operazione di inversione.
Dati p e q, si calcola $\phi(N) = (p-1)(q-1)$.
Si scelgono d e t tali che $ed + t\varphi(N) = 1$, ovvero $ed = 1 - t\phi(N)$.
Per decifrare un messaggio y, basta calcolare:

$$y^d = (x^e)^d = x^{ed} = x^{1-t\phi(N)} = x(x^{-t})^{\phi(N)} = x$$

Avendo a disposizione N e $\phi(N)$ si possono ottenere p e q. Si potrebbe calcolare anche la chiave privata d, ma il problema ha la stessa difficoltà di quello della scomposizione in fattori primi di un numero.

Lo schema di cifratura RSA è utilizzato anche per produrre firme digitali. Siano $N = pq$, $\phi(N) = (p-1)(q-1)$ e sia e un esponente pubblico con $\gcd(\phi(N), e) = 1$. Si definisce la funzione RSA come la funzione biunivoca $RSA[N, e] : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ tale che

$$RSA[N, e](x) \ = \ x^e \ mod\ N \ = \ y.$$

Avendo un messaggio $m \in \mathbb{Z}_N$ e conoscendo d, si ottiene la firma come $RSA^{(-1)}[N, e](m) = \sigma$. Per effettuare la verifica, basta conoscere l'esponente pubblico e calcolare $\sigma^e \ mod\ N \ = \ y$. Se $y = m$, allora il messaggio è autentico.

Algorithm 2 $Sign_{N,\alpha}(m)$

if $m \notin \mathbb{Z}_N$ **then**
  return $\perp$
end if
$\sigma \leftarrow m^d \ mod\ N$
return σ

Algorithm 3 $Ver_{N,e}(m, \sigma)$

if $m \notin \mathbb{Z}_N \vee \sigma \notin \mathbb{Z}_N$ **then**
  return $\perp$
end if
if $\sigma^e = m \ mod \ N$ **then**
  return 1
else
  return 0
end if

2.6.3 Random Oracle Model

E' un modello usato per la dimostrazione della sicurezza di strumenti crittografici.
Un *Random Oracle* O è una funzione a scatola chiusa che risponde a delle richieste ritornando risposte scelte casualmente dall'insieme degli output validi.
Nel modello Random Oracle, formalizzato da M.Bellare e P.Rogaway in [8], si suppone che una funzione hash sia sostituita da una funzione casuale, chiamata appunto Random Oracle. L'avversario A non può calcolare la funzione hash, ma si rivolge all'oracolo sottoponendo ad esso il messaggio.
Una funzione hash si comporta come un Random Oracle quando il suo output non è distinguibile da quello di una funzione casuale.

2.6.4 Schemi di firma omomorfici

Questi schemi di firma hanno la proprietà che per ogni vettore $\vec{a}$ e $\vec{b}$ e scalari α e β, si ha

$$Sign(\alpha a + \beta b) = Sign(a)^{\alpha} Sign(b)^{\beta}$$

Capitolo 3

Soluzioni basate su RSA

Questo capitolo definisce ed approfondisce alcuni schemi di firma, tra i quali quelli che sono stati implementati tramite programmi scritti in C++. Le scelte implementative e i problemi di programmazione riscontrati nella scrittura del codice verranno descritti nel capitolo successivo.

3.1 Schema di firma per Network Coding basato su RSA

Lo schema presentato da Gennaro, Katz, Krawczyk e Rabin in [1] si avvale di una prova di sicurezza nel Random Oracle Model, sotto le assunzioni di RSA standard e raggiunge le migliori prestazioni in reti dove la distanza tra la sorgente e la destinazione non è elevata (20 - 30 hop).

L'idea che sta alla base dello schema è quella di comporre una firma omomorfica su input di lunghezza fissata con una funzione hash applicata ai vettori dello spazio lineare sottostante.
La firma omomorfica è ottenuta calcolando la trasformazione RSA su vettori di lunghezza fissata.
La funzione hash omomorfica è calcolata effettuando le operazioni modulo N, dove N è un numero composto secondo la definizione di RSA, ed usa generatori del gruppo ciclico QR_N.

Per poter utilizzare le funzioni appena citate, lo spazio lineare e le operazioni lineari applicate ai pacchetti devono necessariamente essere calcolate in aritmetica modulo $\phi(N)/4$, ciò comporta che i nodi intermedi debbano conoscere $\phi(N)$, e quindi potrebbero fattorizzare N. Per evitare questo

problema tutte le operazioni vengono svolte sugli interi senza applicare la riduzione modulo $\phi(N)/4$.

Il file è rappresentato da m vettori $\vec{v}^{(i)} \in \mathbb{Z}^n$ per un qualche valore n. Data la dimensione del file da trasmettere e il numero di vettori m, si può determinare la dimensione dei vettori $\vec{v}^{(i)}$ che indichiamo con $|v|$.
Il valore di n può essere scelto tra 1 e $|v|$ ed è lasciato arbitrario per permettere di raggiungere due obbiettivi differenti: n piccoli risparmiano comunicazione mentre n grandi ci danno vantaggi computazionali.

Anche in questo schema, il mittente antepone il vettore unitario $\vec{u}^{(i)}$ al vettore $\vec{v}^{(i)}$ producendo i vettori $\vec{w}^{(i)} = \vec{u}^{(i)}||\vec{v}^{(i)} \in \mathbb{Z}^{n+m}$.
I nodi intermedi calcolano le combinazioni lineari scegliendo i coefficenti α_i dall'insieme $Q = \{0, ..., q-1\}$ dove q è un numero primo piccolo.
Grazie al Lemma2 si dimostra che lo schema ha una buona probabilità di decodifica.

3.1.1 Firma omomorfica basata su RSA

Definiamo uno schema di firma omomorfica basilare basato su RSA, chiamato *Bsig*[1], che opera su vettori di interi di lunghezza prefissata n. Gli elementi utilizzati sono:

- Gruppo RSA: è l'insieme $\mathbb{Z}_N^*$, dove N è il prodotto tra due numeri primi. Si richiede che QR_N, il sottogruppo dei quadrati residui, sia ciclico e che scelto casualmente un elemento di QR_N, questo sia un generatore del gruppo con alta probabilità (un modo per ottenere queste proprietà è quello di scegliere i fattori di N come *safe-prime*).

- Chiave pubblica Pk: è data da $(N, e, g_1, ..., g_n)$ dove N è il prodotto tra due numeri primi, e è un esponente pubblico e $g_1, ..., g_n$ sono generatori di QR_N.

- Chiave di firma segreta Sk: è data da d tale che $ed = 1 \ mod \ \phi(N)$.

- Firma: Sia $v = (v_1, ..., v_n) \in \mathbb{Z}^n$, definiamo la funzione

$$Bsig(v) = (\prod_{i=1}^{n} {g_i}^{v_i})^d \ mod \ N$$

Si può provare che questo schema di firma è omomorfica, ovvero $\forall v, v' \in \mathbb{Z}^n$ e $\alpha, \beta \in \mathbb{Z}$, $Bsig(\alpha v + \beta v') = (Bsig(v))^{\alpha}(Bsig(v'))^{\beta}$.

3.1.2 Firma omomorfica basata su RSA per network coding

Con qualche modifica, lo schema di firma riportato nel paragrafo precedente può essere trasferito nel contesto del Network Coding.
Questo schema, denominato *Nsig* dai suoi creatori ([1]), pone un limite superiore L di hop che un pacchetto può effettuare prima che esso venga rifiutato, per cui la distanza tra la sorgente e le destinazioni deve essere minore di L, altrimenti la destinazione non riceverà mai alcun pacchetto.
Si definisce anche un limite $B = (mq)^L$ che rappresenta la massima dimensione possibile di ogni elemento dei vettori u trasmessi nella rete.
Con M si denota il limite superiore al valore degli elementi dei vettori iniziali $\vec{v}^{(i)}$, ovvero definiamo i vettori $\vec{v}^{(i)}$ come vettori i cui elementi sono scelti dall'insieme $\{0, ..., M\}$.
Ciò premesso, il valore massimo di un elemento trasmesso nella rete è dato da BM che indichiamo con B^*.
Questo schema è composto da:

- Parametri: m, n, M, B e B^*.
- Chiave pubblica: una chiave del tipo $(N, e, g_1, ..., g_n)$ come nello schema Bsig
- Esponente pubblico e : è l'esponente pubblico di RSA, scelto come un primo di dimensioni maggiori di mB^*. Per ottimizzare le prestazioni, e può essere scelto come un numero la cui rappresentazione binaria possiede pochi bit impostati ad 1 (come per esempio $2^{\lceil \lg mB^* \rceil} + a$, dove a è un piccolo intero).
- Chiave privata d: come in RSA, d è scelto tale che $ed = 1 \ mod \ \phi(N)$.
- Funzione di firma: dato un vettore $w \in \mathbb{Z}^{m+n}$ della forma $w = (u_1, ..., u_m, v_1, ..., v_n)$, definiamo la funzione di firma come:

$$Nsig(w) = \Big(\prod_{i=1}^{m} {h_i}^{u_i} \prod_{j=1}^{n} {g_j}^{v_j}\Big)^d \ mod \ N$$

  dove i valori h_i sono definiti come il valore hash della funzione hash calcolata sul fid del file $h_i = H(i, fid)$. La sorgente calcola la firma

[1]Per una trattazione più approfondita si veda [1].

per ogni vettore $\vec{w}_i$ e la trasmette insieme a questi vettori e al file identificato da fid.

- Funzione di verifica $Vrfy(w, \sigma, S, fid)$: un nodo riceve un vettore $w \in \mathbb{Z}^{m+n}$ della forma $w = (u_1, ..., u_m, v_1, ..., v_n)$, una firma σ, un identificativo del file fid e l'identificativo della sorgete S. La funzione verifica che gli elementi di u siano positivi e minori di B e che ogni elemento di v sia positivo e minore di B^*, altrimenti rifiuta il vettore come invalido. Superato questo controllo, la funzione recupera la chiave pubblica $(N, e, g_1, ..., g_n)$ di S, calcola i valori $h_i = H(i, fid)$ e accetta il vettore w se e solo se

$$\sigma^e = \prod_{i=1}^{m} {h_i}^{u_i} \prod_{j=1}^{n} {g_j}^{v_j} \ mod\ N.$$

- Combinazione di firme da parte di un nodo intermedio: un nodo intermedio riceve l vettori $\vec{w}^{(1)}, ..., \vec{w}^{(l)}$ e le relative firme $\sigma_1, ..., \sigma_l$. Dopo aver verificato che $Vrfy(\vec{w}^{(i)}, \sigma_i, S, fid) = 1,\ \forall i \in [1, l]$ e scartato i vettori che non superano questo controllo, controlla che gli elementi dei vettori u siano minori di B/mq e che gli elementi di v siano minori di B^*/mq (questi limiti sono inferiori a quelli usati della funzione di verifica per produrre vettori che vengano accettati dalla funzione *Vrfy*). Ai vettori che hanno superato i controlli, il nodo intermedio applica la funzione $Combine(w^{(1)}, ..., w^{(l)}, \sigma_1, ..., \sigma_l)$ che, scelti casualmente dei coefficenti $\alpha_1, ..., \alpha_l \in Q = \{0, ..., q-1\}$ calcola

$$w = \sum_{i=1}^{l} \alpha_i w^{(i)}$$

e la firma

$$\sigma = \prod_{i=1}^{l} \sigma_i^{\alpha_i} \ mod\ N.$$

Per la dimostrazione di sicurezza di questo schema, si rimanda a [1].

3.2 Network Pseudo-Free Signature

Lo schema presentato da Catalano, Fiore e Warinschi in [4] è il primo schema di firma omomorfica basato su RSA la cui sicurezza è dimostrabile senza il presupposto del Random Oracle.

Anche in questo schema, il file $\bar{F}$ viene considerato come una sequenza di vettori $(\vec{v}^{(1)}, ..., \vec{v}^{(m)})$ n-dimensionali. Per firmare $\bar{F}$, si firma ogni suo singolo vettore $\vec{v}^{(i)}$.
Prima di iniziare la trasmissione, il nodo sorgente crea gli augmented vector $\vec{w}^{(i)}$ anteponendo i vettori $\vec{u}^{(i)}$ ai vettori $\vec{v}^{(i)}$.
Viene indicato con $Q = \{0, ..., q-1\}$, con q primo, l'insieme dal quale i coefficenti sono scelti in modo casuale e con L il limite superiore in salti della distanza tra la sorgente e ogni destinazione.
Come in Nsig, si definisce un limite $B = (mq)^L$ che rappresenta la massima dimensione possibile di ogni elemento dei vettori u trasmessi nella rete, si definisce M come limite superiore alla dimensione degli elementi dei vettori iniziali $\vec{v}^{(i)}$. Si imposta $B^* = MB$.
Sia φ_N una distribuzione che riceve come input un identificativo del file fid, uno spazio vettoriale V e un limite B^*. Sia l_s un parametro di sicurezza e sia l un intero tale che $2^l > B^*$.
La funzione calcola $e = H(fid)$, dove H è una funzione hash del tipo $H : \{0,1\}^* \to \{0,1\}^l$ con la proprietà di essere *division intractable*.
Successivamente, per ogni vettore $\vec{v}^{(i)}$, la funzione sceglie casualmente un intero s_i di $l_s + l$ bit e ritorna $(s_i, \vec{u}^{(i)}, \vec{v}^{(i)})$. Riassumendo, la distribuzione φ_N ritorna i valori $(e, \{(s_i, \vec{u}^{(i)}, \vec{v}^{(i)})\}_{i=1}^m)$.

3.2.1 Schema NetPFSig

Sia G una famiglia di gruppi che è adaptive pseudo-free w.r.t. φ_N.
Lo schema *NetPFSig* è composto da 4 algoritmi $(NetKG, NetSign, NetVer, Combine)$:

- $NetKG(1^k, n)$: Siano $A = \{g, g_1, ..., g_n, h_1, ..., h_m\}$ e $X = \{x\}$ gli insiemi di simboli variabili. L'algoritmo seleziona casualmente un gruppo $\mathbb{G}$ da G e fissa una funzione di assegnazione $\alpha : A \to \mathbb{G}$ per i simboli di A ed imposta la chiave pubblica di verifica $vk = (X, A, \alpha, \mathbb{G}, \varphi_N)$ e la chiave segreta di firma $sk = ord(\mathbb{G})$. Lo spazio di input di φ_N, indicato con $\mathcal{M}$, è l'insieme dei vettori di m-dimensionali i cui elementi sono interi positivi di grandezza massima M.

- $NetSign(sk, V)$: L'algoritmo di firma sceglie casualmente un identificativo fid per il file che viene visto come lo spazio vettoriale V. Successivamente esegue $\varphi_N(V, B^*, fid)$ e ottiene $(e, \{(s_i, \vec{u}^{(i)}, \vec{v}^{(i)})\}_{i=1}^m)$. Successivamente, per $i \in \{0, ..., m\}$ calcola

$$x_i^e = g^{s_i} \prod_{j=1}^{m} h_j^{u_j^{(i)}} \prod_{j=1}^{n} g_j^{v_j^{(i)}}.$$

Sia $\psi : X \to \mathbb{G}$ la funzione di assegnazione di x_i e sia $\sigma_i = (e, s_i, \vec{u}^{(i)}, \vec{v}^{(i)}, fid, \psi)$ la firma per $\vec{w}^{(i)}$. L'algoritmo ritorna $\sigma = (\sigma_1, ..., \sigma_m)$ come firma per V.

- $NetVer(vk, V, \sigma)$: per verificare la correttezza della firma σ, l'algoritmo verifica la dimensione degli elementi dei vettori e se le equazioni $x_i^e = g^{s_i} g_1^{v_1^{(i)}} ... g_n^{v_n^{(i)}} h_1^{u_1^{(i)}} ... h_m^{u_m^{(i)}}$ sono soddisfatte per ogni i in $\mathbb{G}$ da $\psi(x_i)$.
- $Combine(vk, fid, w_1, ..., w_l, \sigma_1, ..., \sigma_l)$: la procedura combina le l firme dei vettori dello stesso file scartando prima tutti i vettori w_i che abbiano elementi di u negativi o maggiori di B/mq o elementi di v negativi o maggiori di B^*/mq. Successivamente sceglie casualmente $\alpha_1, ..., \alpha_l \in Q$, calcola $w = \sum_{i=1}^{l} \alpha_i w_i$ e restituisce in output la firma $\sigma = (e, s, w, fid, \psi)$ di w . La firma è ottenuta calcolando

$$\psi = \prod_{i=1}^{l} \psi_i^{\alpha_i}, \quad s = \sum_{i=1}^{l} \alpha_i s_i.$$

3.2.2 NetPFSig su un gruppo RSA

Consideriamo G come una famiglia di gruppi RSA, scegliamo un gruppo al suo interno e lo fissiamo scegliendo due primi safe p e q e generando $N = pq$. L'utilizzo di primi safe come fattori di N ci assicura che il gruppo scelto è pseudo-free come dimostrato in [4].
Gli elementi di A saranno scelti in QR_N e la chiave di verifica sarà data da (A, N).
Dalla distribuzione φ_N si ottiene il valore e come $e = H(fid)$. Per risolvere l'equazione

$$x_i^e = g^{s_i} \prod_{j=1}^{m} h_j^{u_j^{(i)}} \prod_{j=1}^{n} g_j^{v_j^{(i)}}$$

basta calcolare un valore d tale che $ed = 1 \ mod \ \phi(N)$, dove $\phi(N) = (p-1)(q-1)$ e ottenere x_i come

$$x_i = \big(g^{s_i} \prod_{j=1}^{m} h_j^{u_j^{(i)}} \prod_{j=1}^{n} g_j^{v_j^{(i)}}\big)^d.$$

Tale operazione sarà eseguibile solo dal mittente in quanto è l'unico a conoscere la fattorizzazione di N e quindi il valore $\phi(N)$. Possiamo considerare la chiave di firma come la coppia (p, q).

Capitolo 4

Implementazioni

In questa sezione verranno descritti i programmi sviluppati, i problemi di sviluppo, le scelte effettuate nell'implementazione, i relativi pregi e difetti. Il codice è stato scritto prelevamente in C++ con qualche riferimento a funzioni C.

Il primo problema che si deve affrontare nello sviluppo di un software crittografico è il trattamento di grandi numeri. I tipi standard definiti dai compilatori hanno la dimensione di una-due word e quindi, a seconda delle caratteristiche della macchina, il maggior numero intero senza segno che può essere memorizzato è 2^{32} o 2^{64}. Quindi una limitazione come quella descritta sopra rende impossibile sviluppare strumenti crittografici sicuri, i quali hanno bisogno di numeri molto grandi per essere definiti tali.
Per ovviare a questo problema, nell'implementazione del codice si è fatto uso della *GNU Multiple Precision Arithmetic Library*[1]versione 4.3.2 (indicata brevemente come GNU MP o GMP). La libreria definisce i tipi di dati per l'utilizzo di grandi numeri, le operazioni su questi e un'interfaccia per le classi C++.

4.1 La classe Ncvector

Gli schemi per il Network Coding utilizzano dei vettori e questa classe definisce proprio il tipo di dato che li rappresenta, ovvero *Ncvector* (Network coding vector). La classe opera su grandi numeri interi sfruttando il tipo di dato *mpz_class* definito in GNU MP e mette a disposizione i metodi base di

[1]La libreria, rilasciata sotto General Public License, è scaricabile dal sito http://gmplib.org/insieme a tutta la documentazione necessaria per lo sviluppo del codice.

definizione, lettura e scrittura dei dati.
Inoltre definisce il metodo *makeUnitation(int i)* che trasforma l'Ncvector nel vettore unitario con l'elemento posto ad 1 nella i-esima posizione.
Viene anche definita la funzione *concatena(Ncvector* prefisso, Ncvector* suffisso)* che ritorna un Ncvector dato dalla concatenazione del vettore *prefisso* con il vettore *suffisso*.
La classe inoltre esegue controlli sugli indici, infatti qualora si cercasse di accedere ad un elemento il cui indice supera la dimensione massima, viene visualizzata una condizione di errore.

4.2 La classe MatrixF

Il nodo destinatario ricostruisce il file operando su delle matrici. Per agevolare queste operazioni, è stata creata la classe *MatrixF* che rappresenta matrici di grandi numeri in virgola mobile. Si è scelto di utilizzare numeri in floating point (da qui la F nel nome della classe) per ovviare ai problemi derivanti dal calcolo della matrice inversa.
La classe mantiene una variabile di stato che è settata a 0 quando il determinante non è stato calcolato e ad 1 quando il determinante è noto. Questo stato è utilizzato nei controlli delle operazioni che richiedono il determinante per essere eseguite.

4.3 Le classi fondamentali per il Network Coding

Sono state realizzare le classi *SourceNode*, *IntermediateNode*, *TargetNode* come la rappresentazione, rispettivamente, del nodo sorgente, del nodo intermedio e del nodo destinazione.
Le classi sono in grado di sfruttare sia lo schema Nsig che lo schema NetPFSig tramite l'overloading delle funzioni.
La classe del nodo sorgente si occupa di scomporre il file in vettori, creare gli augmeted vectors, firmarli e inviare le informazioni.
Nella versione realizzata, i vettori w sono ottenuti come $w = u||v$, ovvero anteponendo i vettori unitari ai vettori originali del file.
Il nodo intermedio si occupa di ricevere l vettori e le relative firme, ne verifica la correttezza, combina i vettori e crea la firma del nuovo vettore sfruttando la proprietà omomorfica. Infine invia il vettore w e la sua firma.
La classe del nodo destinazione raccoglie tutti gli m vettori e ne verifica

l'integrità tramite la loro firma, quindi crea le matrici per la ricostruzione del file $\bar{F}$.
Senza l'utilizzo delle funzioni di firma e di verifica delle tre classi, si può realizzare lo schema base del Network Coding. Per poter operare con gli schemi di firma per Network Coding, basta richiamare le relative funzioni che aggiungono gli strumenti crittografici allo schema base e istanziare l'oggetto relativo allo schema di firma che si intende utilizzare.

I vettori considerati nell'implementazione appartengono a $\mathbb{Z}^{256}$, il che ha reso più semplice lo sviluppo del codice, trattando il file $\bar{F}$ come uno stream di byte.
Il numero n di elementi dei vettori originali è lasciato arbitrario. Qualora sia definito n, m è calcolato dividendo la dimensione del file in byte per il numero di elementi del vettori, cioè $m = \frac{|\bar{F}|}{n}$.
Il numero di vettori m è anch'esso lasciato arbitrario. Se m è definito ed n non lo è, si calcola n come $n = \frac{|\bar{F}|}{m}$.
Qualora sia stato scelto sia n che m, il valore n ha la precedenza. Qualora non sia stato scelto nessuno dei due parametri, il valore m viene impostato pari alla costante NUM_VEC ed n calcolato di conseguenza.
I coefficenti α_i utilizzati dai nodi intermedi sono scelti casualmente in $\{0, q\}$, dove $q = 257$ è definito in fase di compilazione.
Vengono scelti in fase di compilazione anche il limite superiore di hop L e il numero massimo di connessioni in entrata di un nodo l.
Tutte le costanti definite in fase di compilazione per lo schema generale possono essere modificate nel file *config.h* per poter adattare lo schema alle proprie esigenze.

4.4 La classe Nsig

Questa classe offre le funzioni per l'utilizzo dello schema Nsig.
Dopo aver generato le chiavi o averle prese in input, la classe sarà in grado di firmare un vettore, verificare la correttezza di una firma e combinare più firme insieme.
Il modulo RSA N è calcolato come il prodotto di due safe-prime, ciò semplifica la ricerca dei generatori g_i di QR_N. Infatti, scelto $a \in [0, N], g = a^2 \ mod \ N$ sarà con altissima probabilità un generatore di QR_N.
I valori h_i sono ottenuti utilizzando l'algoritmo Skein come funzione Hash. La funzione prende in input l'indice i concatenato al fid e genera un digest

a 256 bit. Per la dimostrazione di sicurezza di Skein, si rimanda a [6].
Il fid è calcolato come l'hash del file, utilizzando sempre Skein. Questa scelta è perseverata anche nello schema NetPFSig.
La dimensione del digest $H(i||fid)$ può essere modificata in fase di compilazione intervenendo sulla costante HASH_DIGEST_SIZE. I valori ammessi sono 256, 512 e 1024. E' stato scelto 256 per ridurre il tempo d'esecuzione e la dimensione dei numeri in fase di calcolo.

4.5 La classe NetPFSig

Questa classe offre le funzioni per lo schema NetPFSig.
Dopo aver generato le chiavi o averle prese in input, la classe sarà in grado di firmare un vettore, verificare la correttezza di una firma e di combinare più firme insieme.
Il valore e è definito come $e = H(fid)$, dove la funzione hash H è del tipo $H : \{0,1\}^* \rightarrow \{0,1\}^l$ e gode della proprietà della division-intractability. Come funzione hash H anche in questo caso è stata scelta Skein, con output a 1024 bit. Il parametro l è stato posto a 2048 in modo da avere digest a 2048 bit (è semplice dimostrare che, in questa implementazione, questa scelta verifica la condizione $2^l > B^*$). Si è scelto questo valore per poter assumere che Skein sia division-intractable come suggerito in [5]. Purtroppo Skein ha un output limitato ad un massimo di 1024 bit quindi, per raggiungere un digest di dimensioni maggiori, l'esponente e è calcolato come $e = H(fid||1)||H(fid||2)$. Essendo Skein un Random Oracle, questa costruzione risulta sicura in quanto apportare modifiche limitate ad un messaggio provoca grandi modifiche nel digest, quindi le due parti del digest possono essere considerate indipendenti tra loro. Nel caso in cui il valore e scelto sia pari, si somma 1 ad esso.
Anche il parametro l_s è stato posto a 2048, questo comporta che i valori s_i siano di 4096 bit.
Il modulo N è generato come prodotto tra due primi safe per operare su un gruppo RSA pseudo-free.
In fase di firma si verifica se sia stata calcolata prima la distribuzione φ_N, in caso contrario viene calcolata, dunque questa operazione avviene una sola volta per file. Inoltre si verifica se sia stato calcolato un valore d tale che $ed = 1 \ mod \ \phi(N)$ per poter generare la firma.
In fase di verifica viene controllato se effettivamente il valore e ricevuto sia ottenuto come $H(fid||1)||H(fid||2)$. Per fare ciò è necessario calcolare un valore e' dal fid ricevuto. Questa operazione viene fatta solo una volta per

fid, memorizzando il valore e' per evitare eccessivi richiami della funzione hash.
Nella definizione dello schema, la firma del vettore $w^{(i)}$ è data da $\sigma_i = (e, s_i, w^{(i)}, fid, x_i)$. Quest'ultima viene ricevuta insieme al vettore $w^{(i)}$ e al fid del file, quindi sono stati rimossi questi valori dalla firma per ridurne la lunghezza.

4.6 Rilevazioni

Per valutare la bontà dei due schemi, sono state effettuate due analisi del loro tempo di esecuzione.
La prima riguarda il calcolo del *costo computazionale*, con la quale viene calcolato matematicamente il numero di operazioni che l'algoritmo esegue. Ogni operazione si presume sia effettuata in tempo costante, quindi questa analisi può differire da quella *empirica* che viene affrontata successivamente. In entrambe le analisi non si prenderà in considerazione il tempo di esecuzione impiegato dalle procedure di generazione delle chiavi. Essa viene eseguita una volta soltanto e senza l'utilizzo della rete, quindi non influisce sulle sue prestazioni.
Non viene preso in esame inoltre il tempo impiegato per la combinazione dei vettori, in quanto la procedura è la medesima in entrambi gli schemi.
Bisogna ricordare che:

- q è un numero primo indicante il limite superiore dei valori α_i scelti dai nodi intermedi;
- L è il numero massimo di hop che può effettuare un vettore;
- m è il numero di vettori e n è la dimensione dei vettori originali;
- M è il limite superiore alla dimensione delle coordinate dei vettori originali, $B = (mq)^L$ è il limite superiore alla dimensione degli elementi di u e $B^* = MB$ è il limite superiore alla dimensione degli elementi di v una volta combinati.

Queste variabili avranno il medesimo significato per tutta l'analisi.

4.6.1 Analisi del costo computazionale

Questo tipo di analisi assegna un valore costante alle operazioni basilari e valuta il costo dell'algoritmo come la somma dei costi medi delle operazioni eseguite. La valutazione avviene come se l'algoritmo fosse eseguito da una macchina ad un processore, senza esecuzioni di istruzioni in parallelo, che chiameremo *RAM* (Random Access Machine).
Per una trattazione più approfondita dell'analisi del costo computazionale, si rimanda ai primi capitoli di [7].

Iniziamo dall'analisi dello schema Nsig.
La sua procedura di firma calcola, per ogni vettore $w = u||v$,

$$\sigma = \Big(\prod_{i=1}^{m} {h_i}^{u_i} \prod_{j=1}^{n} {g_j}^{v_j}\Big)^d \ mod\ N$$

I valori h_i sono dati dalla valutazione di una funzione hash che consideriamo avente tempo costante K_1, mentre i valori g_j sono dei generatori facenti parte della chiave pubblica e quindi non influenzeranno l'analisi. Per ogni nuovo file quindi dovremo calcolare m funzioni hash. Dati m vettori di $m + n$ componenti, per generare la firma verranno calcolate $m + n + 1$ elevazioni a potenza. Considerando che gli elementi dei vettori u hanno molti elementi nulli, possiamo considerare una media di $\frac{m}{2} + n + 1$ elevazioni a potenza per la generazione di una firma.
La procedura di verifica opera verificando l'uguaglianza

$$\sigma^e = \prod_{i=1}^{m} {h_i}^{u_i} \prod_{j=1}^{n} {g_j}^{v_j} \ mod\ N.$$

Anche in questo caso avremo $m/2 + n + 1$ elevazioni a potenza per la verifica di una firma.
Per quanto riguarda la procedura di combinazione delle firme, verranno fatte l elevazioni a potenza dove l'esponente è al più q. Quindi il tempo di esecuzione della procedura di combinazione delle firme, in relazione ad un input di l firme, è di $\theta(lq)$.

Analizziamo ora lo schema NetPFSig.
Per ogni file, verrà calcolata la distribuzione φ_N che calcolerà e come l'hash del fid e verranno generati m valori casuali s_i. Consideriamo il costo della distribuzione come $m + K_1$. La procedura di firma calcolerà dapprima la

distribuzione φ_N e poi la firma come

$$\sigma_i = \big(g^{s_i} \prod_{j=1}^{m} h_j^{u_j^{(i)}} \prod_{j=1}^{n} g_j^{v_j^{(i)}}\big)^d \ mod\ N.$$

Il costo delle produttorie sarà medesimo a quello dello schema Nsig, a questo dobbiamo aggiungere l'elevazione a potenza di g^{s_i}. Quindi la firma di un vettore avrà un costo di $\frac{m}{2} + n + 1$.
La funzione di verifica calcolerà

$$\sigma_i^e = g^{s_i} g_1^{v_1^{(i)}} ... \ g_n^{v_n^{(i)}} h_1^{u_1^{(i)}} ... \ h_m^{u_m^{(i)}}$$

Anche in questo caso avremo $m/2 + n + 1$ elevazioni a potenza.
La procedura di combinazione di l firme è la medesima per il calcolo della firma combinata, quindi $\theta(lq)$, in più abbiamo il tempo impiegato per il calcolo del valore random combinato s, che implica un tempo di esecuzione di $\theta(l)$.

La dimensione in bit dei numeri utilizzati influisce fortemente sui tempi di calcolo. Gli esponenti hanno le medesime dimensioni in entrambi gli schemi, mentre le basi h_i utilizzate in NetPFSig sono circa 4 volte più grandi.
Nel caso della firma di un file con Nsig, si può considerare $T_1 = m(\frac{m}{2}c_1 + nc_2 + |d| + K_1)$, mentre nel caso di firma con NetPFSig si ha $T_2 = m(\frac{m}{2}c_2 + nc_2 + |d|)$. Si indica con c_1 il costo di elevazione a potenza di una base di 256 bit e con c_2 il costo di elevazione a potenza di una base di 1024 bit, quindi $T_1 = m(\frac{m}{2}\frac{c_2}{4} + nc_2 + |d| + K_1)$. Tralasciando la costante c_2, $T_1 = \theta(m(m + n + K_1))$ e $T_2 = \theta(m(m + n))$, dunque i due schemi hanno gli stessi costi computazionali per le firme, a meno del valore K_1.
Nel caso della verifica, i tempi sono influenzati da σ^e. Indicando con e_1 l'esponente usato con Nsig e con e_2 l'esponente di NetPFSig, abbiamo $1024 \approx |e_1| < |e_2| = 2048$. Per quanto riguarda la seconda parte della verifica, valgono le considerazioni fatte per le firme. Considerando che con K_1 indichiamo il tempo di esecuzione di una funzione hash, da questa fase di analisi lo schema NetPFSig risulta leggermente migliore di Nsig.

4.6.2 Analisi empirica

In questa analisi ho utilizzato le implementazioni da me create per valutare i tempi di esecuzione su una macchina reale. Nello specifico ho utilizzato un sistema composto da due Pentium Core2 da 1,60 GHz, 2 GB di memoria

RAM, con il sistema operativo Linux Ubuntu 10.04, kernel 2.6.32-29 generic. I test sono stati eseguiti cercando di rispettare le medesime condizioni del sistema e con l'ausilio di script per automatizzarne l'esecuzione e la raccolta dei dati.

Tabella 4.1: Tempi impiegati dagli schemi per firmare e verificare un file delle dimensioni indicate, con vettori di 512 elementi

Dimensioni		Nsig		NetPFSig			
m	n	firma	verifica	firma	verifica	Diff.firme	Diff.verifiche
2	1	0.08	0.06	0.15	0,13	-0,07	-0,07
6	3	0,22	0,20	0,34	0,38	-0,12	-0,18
10	5	0,37	0,33	0,56	0,62	-0,19	-0,29
12	6	0,45	0,39	0,67	0,75	-0,22	-0,36
16	8	0,6	0,53	0,89	0,99	-0,29	-0,46
20	10	0,75	0,67	1,12	1,23	-0,37	-0,56
30	15	1,15	1,00	1,7	1,84	-0,55	-0,84
40	20	1,49	1,33	2,20	2,45	-0,71	-1,12
42	21	1,63	1,4	2,33	2,56	-0,7	-1,16
44	22	1,66	1,45	2,47	2,69	-0,81	-1,24
80	40	3,21	2,67	4,39	4,91	-1,18	-2,24
82	41	3,07	2,74	4,5	5,07	-1,43	-2,33
88	44	3,3	2,93	4,84	5,4	-1,54	-2,47
90	45	3,37	3,00	4,95	5,5	-1,58	-2,5
92	46	3,44	3,07	5,05	5,62	-1,61	-2,55
120	60	4,47	3,99	6,6	7,33	-2,13	-3,34
122	61	4,55	4,06	6,79	7,47	-2,24	-3,41
140	70	5,22	4,67	7,69	8,85	-2,47	-4,18
190	95	7,12	6,35	10,44	11,67	-3,32	-5,32
192	96	7,31	6,41	10,53	11,76	-3,22	-5,35
194	97	7,25	6,72	10,67	11,84	-3,42	-5,12
200	100	7,5	6,7	11,02	12,27	-3,52	-5,57

La tabella 4.1 indica il tempo impiegato dal nodo sorgente per firmare un file delle dimensioni indicate e dal nodo destinazione per verificare il medesimo file. Il valore n è stato impostato a 512, mentre il valore m cambia di conseguenza ed è riportato in tabella.
Dalla colonna delle differenze si nota che Nsig è migliore rispetto a NetPFSig. Si nota inoltre che il divario, in rapporto alla dimensione del file,

diminuisce al crescere delle dimensioni del file in file molto piccoli (con dimensione minore a 10 KB) per poi restare stabile.
Il rapporto tra i tempi di verifica di NetPFSig e i tempi di Nsig è minore a 2 mentre il rapporto dei tempi di firma è minore a 1,5, quindi lo schema NetPFSig non è così cattivo come la dimensione dei valori usati potesse far pensare.

Capitolo 5

Conclusioni

Per concludere, possiamo dire che il progetto cerca di contribuire alla diffusione del Network Coding e delle sue problematiche, ancora poco noti al di fuori dell'ambito accademico.
Grazie allo studio e all'implementazione degli schemi di firma omomorfici basati su RSA, Nsig e NetPFSig, trattati in questa Tesi di Laurea, gli utenti potranno scegliere più consapevolmente tra di essi, ricordando che:

- è preferibile l'utilizzo di Nsig per i suoi tempi ridotti;
- in reti di piccole dimensioni si può utilizzare anche NetPFSig, che lavorando con grandi numeri, pur dilatando i tempi di esecuzione, garantisce maggior sicurezza.

Questo elaborato e le librerie prodotte sono pubblicate sul sito http://danielelicitra.altervista.org da dove sono liberamente scaricabili.

Bibliografia

[1] R. Gennaro, J. Katz, H. Krawczyk and T. Rabin. *Secure Network Coding Over the Integers*.

[2] D. Boneh, D. Freeman, J. Katz and B. Waters. *Signing a Linear Subspace: Signature Schemes for Network Coding*. Public Key Cryptography(PKC), 2009.

[3] T. Ho, R. Koetter, M. Medard, D. R. Karger and M. Effros. *The Benefits of Coding over Routing in a Randomized Setting*. 2003 IEEE International Symposium on Information Theory

[4] D. Catalano, D. Fiore, B. Warinschi. *Adaptive Pseudo-Free Groups and Applications*. Eurocrypt 2011. To appear.

[5] J. Coron *Security analysis of the Gennaro-Halevi-Rabin signature scheme*. Eurocrypt 2000.

[6] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas and J. Walker. *The Skein Hash Function Family*. submitted to NIST for their cryptographic hash algorithm competition.

[7] T. Cormen, C. Leiserson, R.Rivest, C. Stein. *Introduzione agli algoritmi e strutture dati*, seconda edizione. McGrawn-Hill, 2005.

[8] M. Bellare, P. Rogaway. *Random oracles are practical: A paradigm for designing efficient protocols*. In Proceedings of the 1st ACM Conference on Computer and Communications Security(1993), 62 -73.

[9] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, J. Crowcroft. *XORs in The Air: Practical Wireless Network Coding*. IEEE/ACM Trans. Netw.16, 2008.

[10] S. Alexander, B. DeCleene, J. Rogers, P. Sholander. *Requirements and Architectures for Intrinsically Assurable Mobile Ad-hoc Networks.* MILCOM 2008.

[11] S. R. Li, R. W. Yeung, N. Cai. *Linear Network Coding.* IEEE Trans.Inform.Theory 49, (2003).