UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

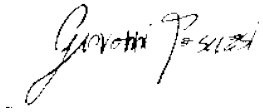
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Secure Multi-party Computation

Relatore
Prof. Cristian Molinaro

Candidato
Giovanni Pascuzzi
Matr. 220183



Anno Accademico 2022/2023

Sommario

Introduzione

Nel panorama sempre più interconnesso e digitalizzato in cui viviamo, l'elaborazione dei dati gioca un ruolo centrale in ogni aspetto della società moderna. Dalle transazioni finanziarie alle analisi mediche, dalla gestione aziendale alle decisioni governative, la disponibilità e la condivisione dei dati sono diventate fondamentali per la società moderna. Tuttavia, la crescente rilevanza dei dati solleva questioni cruciali legate alla privacy e alla sicurezza delle informazioni sensibili. Come si può bilanciare la necessità di attingere e analizzare questi dati con la responsabilità di proteggere e garantire la privacy degli stessi? In questo contesto, emerge una soluzione moderna che permette di affrontare questo problema in modo inaspettato: il *Secure Multi-party Computation* (SMPC). Il SMPC rappresenta un'area all'incrocio tra crittografia, teoria dei protocolli e calcolo distribuito, con l'obiettivo di consentire a diverse entità di elaborare dati condivisi senza mai rivelare direttamente i dati stessi. In altre parole, il SMPC mira ad un equilibrio inaspettato tra condivisione e privacy, permettendo a parti separate di collaborare senza rivelare dettagli sensibili. Questa tesi si propone di svelare la natura del *Secure Multi-party Computation*, scoprire come esso viene utilizzato nelle sue forme più note e analizzarne alcuni casi d'uso. Attraverso un'analisi delle metodologie, dei protocolli e dei casi d'uso, si intende comprendere questa disciplina. Si esaminerà come il SMPC affronta sfide cruciali come la sicurezza computazionale, le prestazioni e l'efficienza, diventando una pietra angolare nella costruzione di soluzioni di analisi e collaborazione basate sui dati. Attraverso situazioni specifiche e reali, questa tesi mira a convincere come il *Secure Multi-party Computation* stia ridefinendo il concetto di sicurezza e privacy nei contesti di condivisione dei dati.

Capitolo 1. Crittografia

La pratica del cifrare i messaggi è l'applicazione principale della crittografia. Essa permette di rendere le informazioni incomprensibili così da perseguirne la riservatezza; per fare ciò è richiesto un algoritmo chiamato *cifrario* ed un valore segreto che prende il nome di *chiave*. La cifratura sicura è tale da garantire l'impossibilità (in pratica) di decifrare le informazioni, o di estrarne una sotto parte, senza la conoscenza della chiave. La cifratura può avvenire mediante cifrari a chiave simmetrica, ovvero in cui la chiave usata per cifrare è la stessa usata per decifrare, o a chiave asimmetrica, in cui la chiave usata per cifrare differisce da quella usata per decifrare. Nella seguente trattazione si farà riferimento a cifrari a chiave simmetrica, tranne nei casi in cui è specificato altrimenti.

1.1 Nozioni di base

Con il termine *testo in chiaro* si farà riferimento al testo non ancora cifrato, mentre il *testo cifrato* sarà ciò che si otterrà come risultato della procedura di cifratura. Un cifrario può essere visto come l'insieme di due funzioni, una prima funzione **E** che dato il testo in chiaro e la chiave ci restituisce il testo cifrato, ed un'altra funzione **D** che dato il testo cifrato **C** e la chiave **K** ci restituisce il testo in chiaro **P**. Se si immaginano queste due funzioni come delle scatole nere risulterà semplice rappresentarle, graficamente, nel seguente modo:

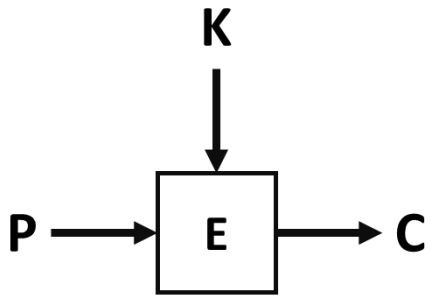


Figura 1-1

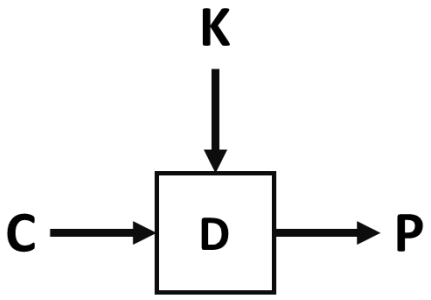


Figura 1-2

Risulta immediato associare alla Figura 1-1 l'operazione di cifratura ed alla Figura 1-2 l'operazione di decifratura. Si indicherà la prima relazione come *C=**E**(K, P)* e la seconda relazione come *P=**D**(K, C)*.

È importante notare che per alcuni cifrari il testo cifrato avrà la stessa lunghezza del testo in chiaro, mentre per altri avrà una lunghezza maggiore; ciò che si può affermare con certezza è che il testo in chiaro non sarà mai più lungo del testo cifrato altrimenti la funzione di cifratura non potrebbe essere invertibile.

1.2 Cifrari Classici

Quando si parla di cifrari classici ci si riferisce a tutti quei cifrari sviluppati prima dell'avvento dei computer e che, dunque, si basano sul manipolare i caratteri invece che i bit. Questi cifrari risultano essere molto semplici proprio perché venivano utilizzati senza il supporto di calcolatori elettronici.

1.2.1 Il Cifrario di Cesare

Il cifrario di Cesare è un primo esempio di cifrario classico; esso si applica su un alfabeto considerato circolare, cioè dopo la lettera Z vi è la A, poi la B e così via. Il suo funzionamento consiste nel fare corrispondere ad ogni lettera del testo in chiaro la lettera che la precede (in alcuni casi si fa riferimento alla lettera che la segue) di tre posizioni nel nostro alfabeto circolare. Per esempio, al testo in chiaro ZOO si fa corrispondere il testo cifrato CRR. Non c'è nulla di speciale nel valore *tre*, è solo molto semplice da calcolare a mente; si può

usare un cifrario di Cesare anche facendo corrispondere ad ogni lettera del testo in chiaro quella che la precede di tredici posizioni. Il cifrario di Cesare, oggi, risulta estremamente insicuro perché vulnerabile ad attacchi basati sulla frequenza delle lettere di un alfabeto. Si potrebbe pensare di rendere il cifrario più sicuro facendo corrispondere ad ogni lettera una lettera che la precede di "X" posizioni, con "X" scelto in modo casuale; nonostante ciò, il cifrario rimarrebbe comunque troppo vulnerabile.

1.2.2 Il Cifrario di Vigenère

A 1500 anni di distanza dal cifrario di Cesare fu finalmente sviluppata una versione considerevolmente migliorata dello stesso che va sotto il nome di *Cifrario di Vigenère*. Questo cifrario è molto simile a quello di Cesare, con l'unica differenza che ad ogni lettera se ne fa corrispondere una che la precede di un numero di posizioni specificate da una chiave. La chiave consiste in una sequenza di caratteri, ognuno rappresentante un valore in base alla sua posizione nell'alfabeto rispetto alla lettera A. Per esempio, la chiave DUH corrisponde ai tre valori "3, 20, 7", perché la D succede di tre posizioni la lettera A, la U di 20 e la H di 7. In fine, il pattern "3, 20, 7" si fa corrispondere ad ogni tre lettere del testo in chiaro, ripetendosi finché non si associa un valore ad ogni lettera del testo in chiaro, così da poter applicare il cifrario di Cesare ad ogni lettera sulla base del valore ad essa associato. Per capire meglio, si consideri il testo in chiaro CRYPTO, a cui viene associato il testo cifrato FLFSNV utilizzando la chiave DUH. Nonostante il cifrario di Vigenère risulti molto più sicuro di quello di Cesare, esso risulta, comunque, essere estremamente vulnerabile ad un serie di attacchi. Un esempio di attacco consiste nell'osservare dei pattern di insiemi di lettere ripetuti nel testo cifrato per poter dedurre la lunghezza della chiave, per poi dedurre la chiave stessa tramite un attacco sulle frequenze; quest'ultimo consiste nell'osservare la frequenza di occorrenza delle varie lettere nel testo cifrato e sostituirle con le lettere con maggiore frequenza della lingua considerata (ad esempio, la lettera E risulta essere la lettera più frequente nella lingua inglese). È immediato capire che un

tale tipo di attacco funziona meglio su testi cifrati di grandi dimensioni. Nonostante le sue grandi debolezze, questo cifrario, a suo tempo, è stato molto usato con successo in quanto veniva utilizzato per cifrare messaggi relativamente corti ed anche perché a volte l'importante è che il messaggio cifrato rimanga tale, agli occhi di un passibile attaccante, solo per un paio di ore; dunque anche se un ipotetico attaccante fosse in grado di infrangere tale cifrario ricavandone la chiave, noi potremmo comunque considerare tale cifrario affidabile nel caso in cui l'attaccante impieghi almeno un paio di ore per infrangerlo.

1.3 Come funzionano i cifrari

Basandosi sui cifrari classici si può astrarre il funzionamento di un cifrario comprendendone le sue due principali componenti, la permutazione e la modalità di funzionamento. Una *permutazione* è una funzione invertibile che si applica su un carattere o un insieme di bit, una *modalità di funzionamento* è un algoritmo che usa una permutazione per processare messaggi di lunghezza arbitraria. Nel cifrario di Cesare la modalità di funzionamento risulta essere banale, in quanto la stessa permutazione viene ripetuta per ogni lettera della frase. Analizzando queste due componenti, possiamo intuire perché i cifrari classici sono destinati ad essere vulnerabili, a differenza dei cifrari moderni che sfruttano la potenza computazionale dei calcolatori.

1.3.1 Permutazione

La maggior parte dei cifrari classici funziona effettuando una sostituzione di una lettera (la rappresentazione delle informazioni non conta, si fa riferimento alle lettere dell'alfabeto latino per convenzione) con un'altra lettera. Questa *sostituzione* non può essere una sostituzione qualsiasi, ma deve essere una *permutazione* così che ogni lettera ne abbia una ed una sola inversa

corrispondente. Per far si che una permutazione sia sicura bisogna che rispetti tre criteri:

- La permutazione dovrebbe essere caratterizzata dalla chiave, così che la permutazione risulti essere segreta fintanto che la chiave è segreta.
- Chiavi diverse dovrebbero caratterizzare permutazioni diverse. Se avessimo più chiavi associate alla stessa permutazione sarebbe molto più semplice decifrare il messaggio senza la conoscenza della chiave.
- Le permutazioni dovrebbero sembrare *casuali*, cioè non dovrebbe essere possibile individuare dei pattern all'interno del testo cifrato, perché i pattern rendono le permutazioni prevedibili e dunque più semplici da attaccare.

Una permutazione che soddisfa questi criteri sarà chiamata *permutazione sicura*, una tale permutazione, però, non è sufficiente a garantire un cifrario sicuro. Un cifrario necessita anche di una modalità di funzionamento per poter gestire messaggi di ogni lunghezza.

1.3.2 Modalità di Funzionamento

Se si usasse la stessa permutazione per tutte le lettere di un testo in chiaro, il testo cifrato risultante conterrebbe dei pattern che renderebbero il messaggio estremamente vulnerabile. La modalità di funzionamento permette di utilizzare permutazioni diverse su lettere diverse; un esempio è il cifrario di Vigenère, anche se quest'ultimo, utilizzando una chiave di lunghezza minore della lunghezza del testo in chiaro, potrebbe comunque far comparire alcuni pattern all'interno del testo cifrato. Ciò si potrebbe risolvere assicurandosi di utilizzare chiavi della stessa lunghezza del testo in chiaro, ma anche in quest'ultimo caso si sarebbe esposti ad attacchi basati su analisi di frequenza; per prevenire questi ultimi bisognerebbe assicurarsi che la chiave, di pari lunghezza del testo in chiaro, venga usata una ed una sola volta. In conclusione, per realizzare un cifrario sicuro sarà necessario combinare una permutazione sicura con una modalità di funzionamento sicura.

1.4 Cifratura Perfetta

I cifrari classici sono destinati ad essere vulnerabili perché non sfruttano in alcun modo la potenza computazionale di un calcolatore e perché per performare al meglio avrebbero bisogno di chiavi lunghissime, impraticabili. L'unico cifrario che garantisce la segretezza perfetta (nozione introdotta da Shannon e che vuole formalizzare il requisito di riservatezza) è il *One Time Pad*, che fa uso proprio di chiavi lunghissime. Anche nel caso in cui un attaccante abbia infinite risorse di calcolo, il One Time Pad garantisce l'impossibilità di scoprire qualsiasi cosa riguardante il testo in chiaro, ad eccezione della sua lunghezza. Il One Time Pad riceve un testo in chiaro **P**, una chiave **K** della stessa lunghezza di P, e produce un testo cifrato **C** secondo la seguente relazione:

$$C = P \oplus K$$

In fase di decifratura si ha la seguente relazione:

$$P = C \oplus K$$

che risulta immediato ricavare secondo la seguente:

$$C \oplus K = P \oplus K \oplus K = P$$

in quanto l'operazione di OR esclusivo fra due elementi identici (K) restituisce l'identità di tale operazione, ovvero una sequenza di bit pari a '0'. Il cifrario One Time Pad deve il suo nome alla sua più grande limitazione, ovvero la necessità di utilizzare una chiave una sola volta; dunque, le chiavi di questo cifrario risultano essere usa e getta in quanto, se avessimo due testi in chiaro cifrati con la stessa chiave, lasceremmo trapelare importanti informazioni. Per capire ciò si considerino due testi in chiaro P_1 e P_2 cifrati utilizzando la stessa chiave *K* e i cui testi cifrati sono indicati con C_1 e C_2. Dallo XOR dei due testi cifrati si ha:

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

Dalla conoscenza di P_1 XOR P_2, è possibile risalire ai singoli testi in chiaro. In conclusione, il One Time Pad è estremamente affidabile a patto che alcune limitazioni non vengano violate, come ad esempio la presenza di una chiave usa e getta che sia lunga tanto quanto il testo in chiaro che dovrà cifrare.

1.4.1 Sicurezza del One Time Pad

Per provare che il cifrario One Time Pad offre una cifratura perfetta, a patto che vengano rispettate le sue limitazioni, si procede secondo la seguente idea: nell'ipotesi che un'eventuale attaccante abbia una potenza di calcolo pressoché infinita e che quindi possa generare tutte le possibili chiavi in tempo ragionevole, si vuole dimostrare che l'attaccante non possa escludere alcun testo in chiaro ricavato dal testo cifrato iniziale, ovvero non possa affermare con certezza che almeno un testo decifrato tramite una delle tanti chiavi da egli generata non sia, con certezza, il testo in chiaro originario. Per intuire meglio tale prova si pensi al fatto che la chiave **K** sia, per definizione, totalmente casuale, dunque un'operazione di **XOR** bit a bit che coinvolga **K** non potrà che risultare in una stringa anch'essa casuale. Quindi, in conclusione, dato un testo cifrato **C** è impossibile, in qualsiasi situazione, per un attaccante estrapolare anche solo una minima informazione, ad eccezione della lunghezza, del testo in chiaro **P**. Per convincersi del fatto che l'operazione di XOR bit a bit con la chiave K dia una stringa di bit totalmente casuali, si pensi al fatto che tale operazione restituisca il primo bit con valore pari ad **1** nel 50% dei casi ed un valore pari a **0** nei restanti casi. Inoltre, da tale idea, segue che per assicurare la cifratura perfetta sia necessario avere una chiave lunga tanto quanto il testo in chiaro, altrimenti il testo cifrato avrebbe un serie di bit finali sempre uguali, che permetterebbero all'attaccante di escludere, tramite ragionamenti logici sulla semantica della frase decifrata, il testo decifrato, restringendo l'insieme dei testi decifrati possibilmente candidati ad essere il testo in chiaro originale. Da tale trattazione è emerso un uso essenziale della teoria delle probabilità, in particolare del calcolo combinatorio; l'uso di tale teoria è molto diffuso nello studio dei cifrari perché ci può dare una misura dell'efficacia di possibili attacchi basandosi sui casi favorevoli a che un attacco abbia successo, considerati tutti i casi esistenti. Con il One Time Pad il numero di possibili chiavi è pari a 2^x, come già visto in precedenza, che risulta essere un valore estremamente grande al crescere di x; dunque, la probabilità di "indovinare" la chiave corretta

generandola in modo del tutto casuale è pari a $\frac{1}{2^x}$, ovvero una probabilità *quasi* nulla.

1.5 Cifratura Sicura

Si è visto come il One Time Pad offra la segretezza perfetta a discapito della sua usabilità. Per ricercare un cifrario utilizzabile si deve rinunciare ad un po' di sicurezza, ma per fare ciò dobbiamo prima capire cosa si intende formalmente per *sicurezza*. Si introduce, dunque, il concetto di *modello di attacco* (ipotesi su cosa l'attaccante possa o non possa fare) e di *obiettivo di sicurezza* (descrizione di come un attacco possa essere considerato di successo). Si definisce infine una *nozione di sicurezza* la combinazione tra un modello di attacco ed un obiettivo di sicurezza. Si dirà che un qualsiasi cifrario acquisisce una nozione di sicurezza se un qualsiasi attaccante "efficiente" lavorando secondo un modello di attacco non riesce a conseguire un obiettivo di sicurezza.

1.5.1 Modelli di Attacco

Un modello di attacco è un insieme di ipotesi su come l'attaccante potrebbe interagire con il cifrario e su cosa egli possa o meno fare con lo stesso. I principali scopi dei modelli di attacco sono i seguenti:

- Individuare dei requisiti minimi, che il cifrario deve rispettare, per chi progetta il cifrario stesso, così che i progettisti sappiano da che tipo di attacchi proteggersi.
- Individuare gli scenari in cui il cifrario può essere utilizzato senza esporre vulnerabilità significative.
- Capire se determinati attacchi possano essere portati a termine con successo nel modello di attacco considerato.

I modelli di attacco non devono, per forza, rispettare gli scenari reali ma, anzi, è meglio se sopravvalutano le capacità di un ipotetico attaccante così da

anticipare eventuali tecniche di attacco future. Un cattivo modello di attacco sottovaluta le capacità dell'attaccante; ciò causa una confidenza teorica nella sicurezza del cifrario, che non rispecchia la realtà. Un'ipotesi, sottointesa, in tutti i modelli di attacco è il *Principio di Kerckhoffs*, che afferma che la sicurezza di un cifrario deve basarsi sulla segretezza della chiave e non sulla segretezza del cifrario. Questo principio è supportato dal fatto che avere un cifrario pubblico aiuta a mantenerlo performante e privo di errori di sicurezza, grazie al contributo di tutti coloro che studieranno tale cifrario e ne segnaleranno eventuali errori di sicurezza.

1.5.2 Modelli a Scatola Nera

Si considerino, ora, dei modelli di attacco basandosi solo su ciò che l'attaccante può vedere e sul tipo di query che egli può fare. Con *query* si intende l'invio di un testo in chiaro ad un sistema, che implementa un cifrario, e la conseguente ricezione del testo cifrato corrispondente, senza che tale operazione esponga dettagli implementativi del cifrario. Questi modelli sono chiamati *a scatola nera* in quanto l'attaccante vede solamente l'input e l'output del sistema, senza conoscerne il funzionamento interno. Di seguito è riportata una lista dei modelli di attacco a scatola nera, elencati dal più debole al più forte:

- *Chipertext-only Attackers (COA)*: si ipotizza che l'attaccate possa solo osservare alcuni testi cifrati senza conoscerne il corrispondente testo in chiaro. Gli attaccanti in questo modello di attacco sono osservatori passivi, ovvero non eseguono alcun tipo di query.
- *Known-plaintext Attackers (KPA)*: si ipotizza che l'attaccante abbia la possibilità di osservare una lista di testi cifrati con il corrispondente testo in chiaro, scelto in modo del tutto casuale. Ancora una volta l'attaccante è un osservatore passivo.
- *Chosen-plaintext Attackers (CPA)*: si ipotizza che l'attaccante possa effettuare delle query di cifratura, ovvero che possa sottomettere al sistema dei testi in chiaro ed osservarne il testo cifrato corrispondente. In questo modello l'attaccante ha più libertà e ricopre un ruolo attivo.

- *Chosen-chipertext Attackers (CCA)*: si ipotizza che l'attaccante possa effettuare query di cifratura o decifratura, ovvero si hanno tutte le ipotesi del CPA ed in più l'attaccante può sottomettere un testo cifrato al sistema ed osservarne il testo in chiaro. Apparentemente queste ipotesi potrebbero sembrare ridicole, perché l'attaccante può direttamente decifrare i messaggi, ma così non è, in quanto a volte non è sufficiente il poter decifrare una serie di messaggi per ricavare la chiave.

In tutti i modelli appena descritti bisogna considerare che ogni query ha un costo computazionale, da tenere in considerazione.

1.5.3 Modelli a Scatola Grigia

Nei modelli di attacco a scatola grigia l'attaccante ha accesso all'implementazione del cifrario. Tali modelli risultano più difficili da definire perché dipendono, appunto, dall'implementazione interna e dunque tendono a non riuscire ad astrarre a pieno la complessità che si avrebbe nella pratica reale. Gli *attacchi a canale secondario* sono una famiglia di attacchi appartenenti ai modelli a scatola grigia. Il termine *canale secondario* vuole indicare il flusso di informazioni che dipende dall'implementazione del cifrario. In questi tipi di attacchi si ipotizza che l'attaccante possa osservare e misurare le caratteristiche analogiche dipendenti dall'implementazione del cifrario senza poterne alterare il suo funzionamento; dunque, questi tipi di attacchi sono detti *non invasivi*. Un esempio di osservazione analogica può essere la misurazione del tempo impiegato dal sistema che implementa il cifrario per fornire una risposta, oppure i vari errori che lo stesso può fornire. Gli *attacchi invasivi* sono un'altra famiglia di attacchi a scatola grigia, più potenti e costosi degli attacchi a canale secondario perché richiedono strumenti sofisticati. Tali strumenti sono utilizzati per effettuare attacchi di ingegneria inversa ed eventualmente modificare, per questo "invasivi", la struttura interna del sistema.

1.5.4 Obiettivi di Sicurezza

Negli anni sono stati formalizzati due principali punti di vista che aiutano a capire cosa vuol dire imparare qualcosa sul funzionamento di un cifrario:

- *Indistinguibilità (IND)*: i testi cifrati dovrebbero essere indistinguibili da delle semplici stringhe casuali. Ovvero se un attaccante ha a disposizione due testi in chiaro e poi riceve il testo cifrato di uno dei due, allora l'attaccante non dovrebbe essere in grado di distinguere a quale testo in chiaro corrisponde il testo cifrato, nemmeno nel modello di attacco CPA o CCA (ovvero nemmeno avendo a disposizione delle query di cifratura o decifratura).
- *Non-Malleabilità (NM)*: Dato un testo cifrato $C_1=E(K, P_1)$, non deve essere possibile creare un testo cifrato C_2, tale per cui il suo corrispondente testo in chiaro P_2 sia, in qualche modo, correlato a P_1. Per esempio, P_2 non deve essere uguale a $P_1 \oplus 1$ oppure a $P_1 \oplus X$ per un dato X. Si noti che il One Time Pad è malleabile.

1.5.5 Nozione di Sicurezza

Gli obiettivi di sicurezza diventano utili quando usati in coppia ad un modello di attacco. Per convenzione si indica una nozione di sicurezza riportando il nome dell'obiettivo di sicurezza e del modello di attacco, separati da un trattino. Per esempio, se indichiamo una nozione di sicurezza con *NM-CCA* vorrà dire che essa è formata dall'obiettivo di sicurezza di "non-malleabilità" e dal modello di attacco "chosen-chipertext attackers".

1.6 Sicurezza Semantica

La più importante nozione di sicurezza in termini di riservatezza è la *IND-CPA*, anche chiamata *sicurezza semantica*. Questa nozione cattura l'idea che i testi cifrati non dovrebbero rivelare alcuna informazione sui testi in chiaro, a patto

che la chiave rimanga segreta. Per rispettare questa nozione è fondamentale che il cifrario restituisca, in corrispondenza di un dato testo in chiaro, un testo cifrato non deterministico, ovvero i testi cifrati dovrebbero sembrare del tutto casuali; ciò vuol dire che se applichiamo due volte lo stesso cifrario allo stesso testo in chiaro con la stessa chiave, otteniamo due testi cifrati completamente differenti. Un modo per assicurare tale nozione è quello di utilizzare la *cifratura randomizzata*, che possiamo esprimere come *C=E(K, R, P)*, dove *R* è una stringa di bit casuali. Nonostante il fattore di casualità coinvolto, la funzione di decifratura dovrà sempre rimanere deterministica, ovvero dovrà sempre essere possibile risalire al testo in chiaro *P* partendo da uno dei tanti testi cifrati C_i ad esso corrispondente. È importante sottolineare come il fattore di casualità sia necessario per perseguire la sicurezza semantica. Infine, notiamo come per poter applicare una cifratura non deterministica sia necessario che i testi cifrati siano leggermente più lunghi dei testi in chiaro.

1.6.1 Ottenere la Sicurezza Semantica

Uno dei modi più semplici per conseguire la sicurezza semantica è quello di utilizzare un *generatore di bit casuali deterministico* (*DRGB*), ovvero un oggetto che restituisce stringhe di bit pseudo casuali a partire da una sorgente di entropia, ovvero un valore segreto:

$$E(K, R, P) = (DRBG(K \,||\, R) \oplus P, R)$$

in cui *K* è la chiave segreta, *R* è una stringa scelta casualmente per ogni nuova cifratura e fornita a un DRBG insieme alla chiave (*K* || *R* indica la stringa composta da *K* seguito da *R*). Possiamo notare, dalla formula sopra riportata, come il cifrario restituisca la stringa *R*, così che in fase di decifratura si possa utilizzare nuovamente il *DRGB* con la stessa sorgente di entropia. Per dimostrare che questa costruzione è IND-CPA bisogna procedere per assurdo, tenendo a mente che per ipotesi il generatore di bit casuali deterministico genera, per l'appunto, stringhe di bit pseudo casuali, dunque non distinguibili da stringhe casuali. Dunque, per prima cosa si nega la tesi, ovvero si afferma che tale costrutto non favorisca una nozione di sicurezza che sia *IND*, quindi si

afferma che è possibile distinguere il testo cifrato $DRBG(K \,||\, R) \oplus P$ da una stringa casuale; dunque, visto che la nozione di sicurezza è anche *CPA* allora l'ipotetico attaccante, conoscendo il testo in chiaro *P*, può ricavare, tramite operazione di **XOR** $DRBG(K \,||\, R)$, che risulterà anch'esso distinguibile da una stringa casuale (perché l'operazione di XOR non altera tale proprietà). In fine, si ha una contraddizione, perché per ipotesi il *DRGB* genera stringhe di bit pseudo casuali, dunque non distinguibili da stringhe casuali; in conclusione, abbiamo provato la qualità di sicurezza semantica per tale cifrario.

1.7 Relazioni tra Nozioni di Sicurezza

Si considerino, ora, le nozioni di sicurezza *NM-CPA*, *NM-CCA*, *IND-CPA*, e *IND-CCA*. Ci si chiede, quali relazioni legalo le stesse? Il soddisfacimento di una implica l'altra? Alcune relazioni sono banali, infatti *NM-CCA* implica *NM-CPA*, oppure *IND-CCA* implica *IND-CPA*, proprio perché tutto ciò che un attaccante può fare in un modello di attacco *CCA* lo può fare anche in un modello di attacco *CPA*. Una relazione meno ovvia risulta essere quella in cui *IND-CPA* non implica *NM-CPA*; per convincersene si pensi al cifrario da noi analizzato per ottenere la sicurezza semantica, in esso il testo cifrato, essendo ottenuto tramite una operazione di **XOR,** risulta essere malleabile (più o meno come accade nel One Time Pad). Ciò perché, se abbiamo un testo cifrato *(X, R)* possiamo ottenere *(X⊕1, R)*, che risulta essere un testo cifrato valido per il testo in chiaro *P⊕1*, dunque si contraddice la nozione di non malleabilità. Nonostante ciò, la relazione in cui *NM-CPA* implica *IND-CPA* sussiste. Per convincersi di questo si pensi che la modalità di attacco *NM* è più restrittiva rispetto alla modalità di attacco *IND*, ovvero la *NM* lascia meno libertà dall'attaccante. In conclusione, si ha anche che *IND-CCA* implica e contro implica *NM-CCA*.

1.8 Tipi di Cifratura

1.8.1 Cifratura Asimmetrica

Fino ad ora si è trattata la cifratura simmetrica, in cui la chiave di cifratura era condivisa sia dal mittente che dal destinatario, ma esiste anche la c*ifratura asimmetrica* in cui il mittente ed il destinatario posseggono chiavi diverse. In particolare, la chiave posseduta dal mittente (o eventualmente dai mittenti), ovvero la chiave di cifratura, viene detta *chiave pubblica* in quando può essere tranquillamente resa pubblica. La chiave posseduta dal ricevente, ovvero la chiave di decifratura, viene detta *chiave privata* perché permette al suo possessore di decifrare tutti i messaggi cifrati con la corrispondente chiave pubblica; dunque, tale chiave deve essere mantenuta segreta. Lo scenario appena discusso si riferisce ad una comunicazione riservata; l'uso delle chiavi cambia in scenari diversi, come ad esempio nel caso della firma digitale. Nella crittografia asimmetrica non deve essere possibile ricavare la chiave privata da quella pubblica. Le nozioni di sicurezza per tale tipo di crittografia sono identiche a quelle per la crittografia simmetrica, con l'unica differenza che nella crittografia asimmetrica il modello di attacco, se non specificato, è il *CPA*. Ciò perché la chiave pubblica è appunto pubblica, dunque chiunque può cifrare un messaggio arbitrario. Da un punto di vista funzionale la cifratura asimmetrica, rispetto alla simmetrica, offre alcuni vantaggi; un primo vantaggio è quello di non necessitare di n^2 chiavi per far comunicare, in modo sicuro, n utenti tra di loro, inoltre nella crittografia simmetrica una chiave risulta essere più vulnerabile perché vi sono due interlocutori che la conoscono e dunque vi sono maggiori casi in cui la chiave venga resa, per sbaglio, pubblica. Un ultimo vantaggio della crittografia asimmetrica è quello di non avere problemi nello scambiare una chiave di cifratura, in modo sicuro, tra due interlocutori. Nonostante tutte queste differenze, la cifratura simmetrica viene utilizzata assieme alla cifratura asimmetrica per offrire un gran numero di servizi perché risulta essere molto più veloce da eseguire.

1.8.2 Cifratura di Autenticazione

Nel tempo sono state sviluppate varianti della cifratura simmetrica ed asimmetrica, così da offrire funzionalità che non si limitassero alla semplice cifratura e decifratura. La *cifratura di autenticazione (AE)* è una variante della cifratura a chiave simmetrica che, oltre al testo cifrato, prevede la generazione di un *tag*. Questo tipo di cifratura può essere indicata come *AE(K, P)=(C, T)* ed illustrata nella Figura 1-3:

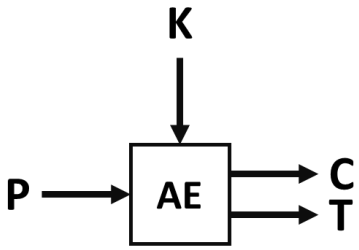


Figura 1-3

in cui *T* indica una stringa, di modeste dimensioni, impossibile da determinare senza la chiave *K*. Lo scopo del tag *T* è quello di confermare la procedura di decifratura (in quanto in fase di decifratura è necessario lavorare su *K*, *P* e *T*) nel senso che se la procedura di decifratura "fallisce" vuol dire che il testo cifrato ricevuto non è valido e potrebbe esser stato alterato. Dunque, il ruolo del tag è quello di garantire integrità. Un'estensione di questo tipo di cifratura è la *cifratura di autenticazione con informazioni associate (AEAD)*, in cui il cifrario lavora su un testo in chiaro, una chiave ed un testo che non deve essere cifrato. Ciò risulta essere molto utile nelle reti di telecomunicazione, in cui l'header del datagramma deve rimanere comprensibile dai nodi intermedi, mentre il payload dello stesso deve essere cifrato; in tale scenario il tag *T* ci aiuta a capire se qualche informazione, che sia del datagramma o del payload, sia stata modificata.

1.8.3 Cifratura a Preservazione di Formato

I normali cifrari lavorano su sequenze di bit, senza considerare se quelle sequenze rappresentino un'immagine, un indirizzo, o comunque senza

considerarne il formato. Ma se si ha la necessità di preservare il formato, ovvero di avere un testo cifrato con una determinata struttura o forma, si ricorre alla *cifratura a preservazione di formato (FPE)*. Essa permette di creare testi cifrati che hanno lo stesso formato dei corrispondenti testi in chiaro. Questo tipo di cifratura potrebbe essere richiesta in sistemi di persistenza, in cui è possibile memorizzare dati che hanno un determinato formato.

1.8.4 Cifratura Completamente Omomorfica

La *cifratura completamente omomorfica (FHE)* permette di eseguire operazioni su testi cifrati allo scopo di effettuare operazioni (eventualmente diverse) sui corrispondenti testi in chiaro. Questo tipo di cifratura risulta essere estremamente potente, ma il suo lato negativo è che i cifrari esistenti che permettono di portare a termine una tale cifratura sono estremamente lenti, anche se applicati su funzioni semplici.

1.8.5 Cifratura di Ricerca

La *cifratura di ricerca* è una tecnica crittografica che consente di cercare informazioni all'interno di un database cifrato senza rivelare i dettagli della ricerca o i termini di ricerca a chiunque, nemmeno al server o a terze parti. Questo è utile quando si desidera mantenere privati i dati sensibili durante le ricerche, ad esempio quando i dati sono memorizzati in un server remoto (come nel cloud) e si vuole evitare che il fornitore di servizi cloud possa vedere i dettagli delle ricerche effettuate.

1.8.6 Cifratura Modificabile

La *cifratura modificabile (TE)* è un tipo di cifratura simile alla cifratura di base, ma con un parametro aggiuntivo chiamato "tweak". Questo "tweak" è una sorta di "aggiustamento" o "modifica" che viene utilizzato per simulare diverse versioni di un cifrario. Questo è utile per aumentare la flessibilità e la sicurezza

della cifratura in situazioni specifiche. La cifratura modificabile è principalmente usata nella cifratura dei dischi rigidi, inoltre è una cifratura di basso livello su cui si realizzano altri schemi di cifratura, come la AE.

1.9 Cifrari Deboli

Gli algoritmi di cifratura, o eventualmente le loro implementazioni, posso fallire nel mantenere la segretezza dei testi da essi cifrati. Un esempio pratico è quello dell'algoritmo di cifratura *A5/1*, per reti mobili 2G, in cui si è adoperato un attacco di tipo *time-memory trade-off (TMTO)*; questo attacco consiste nell'effettuare pesanti calcoli computazionali per svariate settimane così da creare delle tabelle contenenti le coppie *testo in chiaro-testo cifrato* usate per effettuare, poi, il reale attacco. Nonostante questo attacco abbia richiesto svariati anni per essere portato a termine, il cifrario A5/1 è da considerarsi debole. Un cifrario potrebbe fallire nel garantire la sicurezza e la riservatezza dei messaggi da esso cifrato anche per motivi non direttamente correlati all'algoritmo di cifratura. Per esempio, si consideri un sistema che implementi un cifrario che rispetta il modello di attacco *CCA*; un cifrario così fatto potrebbe comunque essere vulnerabile ad attacchi di tipo *padding oracle*, ovvero attacchi in cui vi è un "oracolo" (un servizio esposto dal sistema) che certifica la correttezza del testo cifrato. Tale interazione, da parte del sistema, potrebbe permettere all'attaccante di ricostruire il padding applicato al testo in chiaro e dunque estrapolare il testo in chiaro stesso senza ricorrere alla chiave segreta di cifratura.

Capitolo 2. Secure Multi-party Computation

Si immagini di avere un sistema di calcolo distribuito, quindi un sistema dove calcolatori distinti devono portare a termine l'esecuzione di almeno una funzione (come, ad esempio, le operazioni di update di un database distribuito), e di volerla rendere sicura. Il *Secure Multi-party Computation* ha proprio l'obiettivo di permettere ai vari calcolatori (*parties*), che hanno un ruolo all'interno del sistema distribuito, di processare le funzioni del sistema in modo sicuro. Quindi, a differenza del *distributed computing*, che si occupa di possibili errori di calcolo dovuti ad accessi concorrenti e/o situazioni di crash del sistema, il Secure Multi-party Computation si occupa di problematiche di sicurezza legate alla presenza di calcolatori, o parties, malevoli che possono, o meno, fare parte del sistema distribuito. Quindi, si parte dal presupposto che un protocollo possa essere soggetto ad attacchi che mirano a estrapolare informazioni o alterare il risultato del calcolo distribuito. Dunque, due fattori fondamentali per un protocollo sicuro di calcolo distribuito sono la *privacy* e la *correttezza*. Con privacy si intende che nulla, che non sia estremamente necessario, deve essere fornito come informazione ad un party; dunque, ogni party deve conoscere solo il minimo indispensabile per poter generare il suo output. La correttezza riguarda il fatto che ogni party deve generare l'output in modo corretto; dunque, il calcolo complessivo non deve differire dal calcolo della funzione inizialmente considerata dal sistema distribuito. Il Secure Multi-party Computation ci permette, dunque, di effettuare dei calcoli accedendo a dei dati senza violarne la privacy e garantendone la correttezza. Un esempio è il problema di confrontare il DNA di una persona con dei DNA campione presenti all'interno di un database per poter scoprire se la persona ha una probabilità, alta o bassa che sia, di sviluppare un tipo di cancro; in un tale scenario è essenziale che il DNA della persona e i DNA campione presenti nel database non vengano resi pubblici, in quanto risultano essere dati fin troppo

sensibili. Inoltre, in un tale scenario, è necessario che il risultato finale, ovvero l'esito del test, sia corretto.

2.1 Terminologia

Nella letteratura scientifica il *Secure Multi-party Computation* è indicato con le sigle *MPC* o *SMPC*. Oltre a queste sigle, si fa riferimento all'argomento in oggetto con la sigla *SFE*, acronimo di *Secure Function Evaluation*. Inoltre, specifiche versioni del *MPC* hanno dei nomi propri, come ad esempio il *private set intersection* (*PSI*).

2.2 Sicurezza del MPC

Come già detto, il contesto in cui ci si muove è quello di un sistema distribuito, in cui vi è un attaccante che controlla un sottoinsieme dei vari parties che formano il sistema e che ha la malevola intenzione di attaccare il protocollo di esecuzione. I vari parties sotto il controllo dell'attaccante sono detti *corrotti*. Dunque, bisogna capire quando e come un protocollo può essere definito "sicuro", perciò una definizione formale di sicurezza è richiesta all'interno del MPC. Nel tempo sono state molte le definizioni proposte, tutte con l'obiettivo di assicurare determinate proprietà; qui, si elencano le principali proprietà:

- *Privacy*: ogni party non deve poter conoscere nulla dell'input degli altri party ad eccezione di ciò che si può ricavare dai loro output.
- *Correttezza*: ogni party produce un output considerato corretto.
- *Indipendenza degli input*: i party corrotti devono scegliere il loro input in modo indipendente da quello degli altri party onesti. È importante osservare come questa proprietà non sia implicata dalla *privacy*, in quanto un party corrotto potrebbe far variare il proprio input in base ad informazioni ricavate dall'output di un party onesto e quindi in qualche modo correlate all'input di quest'ultimo.

- *Ricezione dell'output garantita*: i party corrotti non devono essere in grado di impedire la ricezione di un output, generato da un party onesto, da parte di un altro party onesto. In altre parole, il protocollo deve prevenire attacchi di tipo denial of service (DOS) da parte dell'attaccante.
- *Imparzialità*: i parties corrotti devono ricevere i propri output (output generati da altri parties) se e solo se i parties onesti ricevono i propri output. Si noti come la proprietà di *ricezione dell'output garantita* implichi l'*imparzialità*, ma non il viceversa.

Si osservi, nuovamente, come questa serie di proprietà non costituisce una definizione di "sicurezza", ma ci dà una serie di requisiti minimi che ogni protocollo "sicuro" dovrebbe rispettare. Ovviamente un possibile modo di formulare una definizione di "sicurezza" è quello di elencare una serie di requisiti (come si è appena visto) e poi imporre che vengano rispettati affinché il protocollo possa essere considerato sicuro. Nonostante ciò, tale approccio non è conveniente, sia perché nell'elencare tante proprietà se ne potrebbe dimenticare qualcuna correlata a qualche caso specifico, sia perché vi è la necessità di avere una definizione che sia semplice da verificare. La definizione di sicurezza viene, oggi, formalizzata considerando il seguente concetto: si immagini un **mondo ideale** in cui vi è un party fidato e incorruttibile che si offre di eseguire i calcoli computazionali degli altri parties; dunque, ogni party interagisce con questo party fidato inviandogli i propri input e ricevendo un output. In un tale scenario, le uniche libertà che ha un attaccante sono quelle di scegliere i vari input dei vari party corrotti. In questo mondo ideale tutte le proprietà da noi descritte sono rispettate, in particolare la correttezza è rispettata perché, per ipotesi, il party fidato è anche incorruttibile e dunque genererà sempre un output corretto e non deviato. Ovviamente nel mondo reale non vi può essere un party fidato, piuttosto vi sono i vari parties ognuno che esegue il protocollo in modo locale senza alcun aiuto; nonostante ciò, un protocollo sicuro dovrebbe simulare il comportamento che si ha all'interno del mondo ideale. Dunque, *un protocollo eseguito dai vari parties è detto sicuro se i vari attacchi che possono essere portati a termine, da un ipotetico attaccante, nel mondo reale sono gli stessi che possono essere portati a termine anche nel*

mondo ideale. Questa definizione risulta essere molto solida perché nel mondo ideale non può essere portato a termine alcun tipo di attacco. Più formalmente, la sicurezza di un protocollo si stabilisce confrontando i risultati, a parità di input, dell'esecuzione dello stesso nel mondo reale e nel mondo ideale. Quindi, a tutti gli effetti, l'esecuzione di un protocollo nel mondo reale deve *emulare* quella dello stesso nel mondo ideale; tale formulazione di "sicurezza" è chiamata *modello di simulazione ideale/reale*. Per comprendere il perché di questa definizione si osservino i motivi per cui le proprietà da noi inizialmente descritte sono rispettate; ignoriamo le motivazioni riguardanti la *privacy* e la *correttezza,* perché triviali e già in parte osservare precedentemente, per concentrarci sulle restanti:

- *Indipendenza degli input*: nell'esecuzione ideale tutti gli input vengono inviati al party fidato prima che i vari output siano ricevuti dai vari parties. Dunque, i parties corrotti non hanno alcuna informazione riguardante gli input dei parties onesti al tempo di invio degli input, perciò tutti gli input sono forniti in modo indipendente.
- *Ricezione dell'output garantita* e *imparzialità*: nell'esecuzione ideale il party fidato restituisce sempre gli output, ad ogni singolo party. Ricordiamo come la ricezione dell'output garantita implica l'imparzialità.

Infine, è importante dire che in alcuni casi la definizione viene rilassata, escludendo la necessità di soddisfare le proprietà di ricezione dell'output garantita e dell'imparzialità. In questi casi il livello di sicurezza ottenuto è detto di *sicurezza con interruzione*. Di solito questo livello di sicurezza si usa nei casi in cui risulta praticamente impossibile assicurare l'imparzialità o nei casi in cui vi è la necessità di utilizzare protocolli più efficienti (resi tali proprio dal fatto che non devono garantire l'imparzialità). Per esempio, nei casi in cui vi è un solo party che riceve l'output, questo livello di sicurezza potrebbe essere usato per ottenere vantaggi in termini di efficienza.

2.3 Potere dell'Avversario

nel senso che in alcuni circostanze (avversario che dispone di poche risorse) si possono rilassare le condizioni sotto cui garantire la sicurezza

La definizione di sicurezza che si è vista non considera il *potere dell'avversario*, inteso come il livello di controllo o influenza esercitato dall'attaccante sui parties corrotti. Per poter, in qualche modo, considerare quest'ultimo si introducono due parametri che aiutano a misurarlo: il *comportamento consentito* e la *strategia di corruzione*.

2.3.1 Comportamento Consentito

Questo concetto riguarda le azioni che un attaccante può intraprendere durante l'esecuzione del protocollo. Ci sono tre tipi di possibili attaccanti:

(a). *attaccanti quasi onesti*: in questo modello anche i parties corrotti seguono le specifiche del protocollo in modo onesto, però l'attaccante ottiene lo stato interno di tutti i parties corrotti con l'obiettivo di poter ricavare informazioni che dovrebbero rimanere private. Nonostante la semplicità di questo modello, che non è adatto alle minacce odierne, un protocollo con questo livello di sicurezza impedisce che vi siano fuoriuscite di dati non consentite. Gli attaccanti quasi onesti sono chiamati anche "curiosi ma onesti" o "passivi".

(b). *attaccanti malevoli*: in questo modello i parties corrotti possono **arbitrariamente** eseguire una versione del protocollo che non rispetta la specifica dello stesso. Ci si riferisce a questo tipo di attaccanti con l'aggettivo di "attivi", in contrapposizione agli attaccanti "passivi".

(c). *attaccanti occulti*: in questo modello l'attaccante potrebbe comportarsi come se fosse *malevolo*, con la differenza che al momento dell'attacco egli potrebbe venire scoperto con una determinata probabilità dipendente dall'applicazione. Ovviamente, se l'attaccante non viene scoperto questo modello converge al modello con attaccanti malevoli.

In conclusione, fare riferimento al modello con attaccanti malevoli ci consente di assicurare uno standard di sicurezza maggiore, in quanto assicurare la

sicurezza in un tale modello vuol dire che nessun attacco può essere portato a termine.

2.3.2 Strategia di Corruzione

Questo concetto riguarda il come e il quando un determinato party può essere corrotto. Si possono descrivere tre principali modelli:

(a). *modello di corruzione statica*: in questo modello l'insieme dei parties corrotti è determinato a priori, prima che il protocollo venga eseguito. Dunque, durante l'esecuzione dello stesso i parties onesti rimangono onesti e quelli corrotti rimangono corrotti.

(b). *modello di corruzione adattiva*: in questo modello l'insieme dei parties corrotti non viene determinato a priori ma, i parties si corrompono a tempo di esecuzione del protocollo. La scelta di chi e quando corrompere spetta all'attaccante che può, in maniere arbitraria, effettuarla anche in modo *adattivo*, ovvero secondo dinamiche correlate all'esecuzione del protocollo stesso. È importante osservare come, in tale modello, dopo che un party viene corrotto non potrà più diventare un party onesto.

(c). *modello di corruzione proattivo*: in questo modello si hanno le stesse dinamiche del *modello di corruzione adattiva* con la differenza che i parties sussistono nello stato di corruzione per un intervallo di tempo finito, cioè un party corrotto a tempo di esecuzione del protocollo può ridiventare onesto. Un tale comportamento, nella vita pratica, si può manifestare in seguito alla scoperta di una compromissione in un eventuale sistema e alla sua successiva riparazione, riportando lo stesso ad uno stato in cui non vi sono parties corrotti.

In conclusione, non esistono modelli che risultano essere sempre giusti, anzi bisogna scegliere il modello più adatto che può variare da applicazione ad applicazione.

2.4 Teoremi di Canetti, Composizione Modulare Sequenziale e Parallela

Un protocollo SMPC, nella realtà, non viene eseguito da solo ma viene affiancato all'esecuzione di molti altri protocolli (SMPC o meno). È immediato domandarsi se questo accoppiamento tra le esecuzioni avviene in modo sequenziale o parallelo, ma prima di dare una risposta bisogna presentare un importante risultato. R. Canetti, nell'articolo *Security and Composition of Multi-party Cryptographic Protocols*, dimostra che, se si esegue un protocollo SMPC all'interno di un sistema più grande la sicurezza dello stesso si preserva, considerando il giusto modello come definizione di sicurezza. Tale teorema prende il nome di *composizione modulare*, e rende possibile la costruzione di macro-protocolli tramite composizione di protocolli, visti come dei moduli base, sicuri ma più piccoli. Conoscendo, ora, tale risultato, possiamo ritornare alla domanda che sorgeva spontanea per dire che, nell'eventualità in cui il protocollo di SMPC venga eseguito in modo sequenziale rispetto ad altri protocolli del sistema (cioè, in un determinato istante di tempo il protocollo di SMPC sarà l'unico in esecuzione) allora il teorema di composizione modulare verrà coniugato tramite la sua forma di *teorema di composizione modulare sequenziale* e continuerà a valere. In tale forma, questo teorema, prevede il *modello di simulazione ideale/reale* come definizione di sicurezza. Nella maggior parte dei casi, però, i protocolli di SMPC vengono eseguiti, tra di loro e in relazione ad altri protocolli non sicuri, in parallelo; in questi casi, il *teorema di composizione modulare sequenziale* non sussiste e dunque questi protocolli non possono essere reputati sicuri. Per ovviare a tale problema si è introdotta una nuova definizione di sicurezza che va sotto il nome di *componibilità universale*. Ogni protocollo reputato sicuro rispetto a questa definizione avrà sempre una esecuzione equivalente ad una esecuzione all'interno del mondo ideale, a prescindere da quali siano i protocolli che verranno eseguiti in modo concorrente ad esso (cioè, utilizzando questa nuova definizione di sicurezza il teorema di Canetti potrà essere applicato). È dunque immediato capire perché questa nuova definizione è considerata la definizione standard del concetto di

sicurezza per i protocolli SMPC. Ovviamente tale definizione obbliga a perdere valore in termini di efficienza e di semplicità del sistema.

2.5 Importanti Implicazioni di Definizione

La definizione di sicurezza secondo il modello di simulazione ideale/reale ha delle importanti implicazioni pratiche. Per utilizzare un protocollo SMPC, basta verificare la sicurezza del sistema nel caso in cui il protocollo SMPC sia eseguito nel “mondo ideale”; se tale test dimostra che il sistema è sicuro, allora anche il sistema nel mondo reale lo sarà, a patto che ci si trova negli adeguati casi di composizione. Dunque, il modello di simulazione ideale/reale offre un’astrazione che permette agli utilizzatori di un protocollo SMPC di ignorarne il funzionamento o i dettagli strettamente crittografici. Nonostante ciò, vi è un dettaglio spesso frainteso dei protocolli SMPC, che riguarda la tipologia di input accettati. Un protocollo SMPC si comporta come se venisse eseguito nel “mondo ideale”, ma nel mondo ideale qualsiasi parties può selezionare l’input che vuole, persino i parties corrotti, e non c’è modo di prevenire ciò. Quindi, se la sicurezza di un sistema dipende anche dal tipo di input che un sistema può accettare, andranno costruiti meccanismi esterni al protocollo per assicurarsi che gli input siano accettabili. Un possibile meccanismo potrebbe essere quello di utilizzare input firmati. Un altro dettaglio spesso frainteso riguarda, questa volta, l’output. Un protocollo SMPC assicura la sicurezza del processo di esecuzione, ovvero tutto rimane privato durante il processo di calcolo, però non assicura quello dell’output, ovvero l’output generato potrebbe rivelare informazioni sensibili. Un semplice esempio riguarda l’uso di un protocollo SMPC per calcolare il salario medio tra due persone; l’output finale sarà il salario medio, ma a partire dallo stesso ognuno dei due parties potrebbe ricavare il salario altrui (perché entrambi sono in possesso del valore del proprio salario) nonostante il salario altrui sia un’informazione protetta durante il processo di esecuzione del protocollo. In conclusione, il semplice uso di un solo protocollo SMPC non è sufficiente a far fronte a tutte le minacce.

La definizione di sicurezza per un protocollo SMPC, secondo il modello di simulazione ideale/reale, potrebbe sembrare troppo restrittiva, quasi impossibile da soddisfare. Sorprendentemente, però, è stato dimostrato come qualsiasi funzione distribuita possa essere eseguita in modo del tutto sicuro, in presenza di parties corrotti. Si analizzano, ora, alcuni di questi risultati, con l'assunzione di indicare con la lettera **n** il numero di parties coinvolti nell'esecuzione, e con la lettera **t** la cardinalità massima del sottoinsieme dei parties corrotti di **n** (si presti attenzione al fatto che l'identità dei parti corrotti non è nota):

1. Per $t < \frac{n}{3}$ (cioè, nei casi in cui meno di un terzo di tutti i parties può risultare corrotto) si possono sviluppare dei protocolli SMPC che assicurino la *sicurezza computazionale*, per tutte le possibili funzioni, a patto che i parties del sistema comunichino tramite dei canali autenticati su una rete punto-punto sincrona. Si può fare lo stesso, ma assicurando una *sicurezza information-theoretic*, assumendo che i canali di comunicazione siano anche privati [2,3].
2. Per $t < \frac{n}{2}$ (cioè, nei casi in cui è garantita una maggioranza onesta) si possono sviluppare dei protocolli SMPC che assicurino la *sicurezza computazionale* e la *sicurezza information-theoretic*, per tutte le possibili funzioni, a patto che i parties del sistema rispettino le assunzioni al punto **1** e che abbiano, inoltre, accesso ad un canale broadcast [4,8].
3. Per $t \geq \frac{n}{2}$ (cioè, nei casi in cui il numero di parties corrotti non è limitato) si possono sviluppare dei protocolli SMPC (che non assicurano le proprietà di *imparzialità* e *ricezione dell'output garantita*).

Ricordando i teoremi di Canetti, è immediato capire che i protocolli SMPC possono essere realizzati per ogni sistema distribuito. Dunque, qualsiasi funzione che richiede una esecuzione distribuita può essere eseguita in modo sicuro. Sottolineiamo come tutti questi risultati siano di tipo teorico, ovvero non considerino i limiti pratici dovuti a costi computazionali nell'esecuzione dei protocolli.

Capitolo 3. Tecniche

Nel tempo sono state sviluppate svariate tecniche per costruire protocolli SMPC che assicurassero determinate proprietà per determinate composizioni. Esula da questa trattazione descrivere tutte le tecniche esistenti, però si analizzeranno alcuni semplici esempi su come un protocollo SMPC possa essere costruito.

3.1 Condivisione Segreta di Shamir

Uno strumento molto usato per la costruzione di protocolli SMPC è la *condivisione segreta di Shamir*, ideata da Adi Shamir, co-inventore dell'algoritmo RSA. Uno schema di condivisione segreta pone la soluzione al problema di condividere un segreto **s** tra **n** parties, imponendo che un insieme di **t+1** o più parties possa ricostruire il segreto mentre un insieme di **t** o meno parties non possa ricostruire il segreto e non possa ricavarne alcun tipo di informazione (non è un caso che nel secondo capitolo di tale trattazione si sia usata la lettera **t** per indicare la cardinalità massima dell'insieme dei parties corrotti). Uno schema che riesce a risolvere un problema così formulato prende il nome di *schema a condivisione segreta con soglia a (t+1)-su-n*. Lo schema ideato da Adi Shamir sfrutta il fatto che, dati *t+1* punti del piano cartesiano $(x_1,y_1),\ldots,(x_{t+1},y_{t+1})$ con ascissa x_i unica, esiste un unico polinomio *q(x)* di grado massimo *t* tale per cui $q(x_i)=y_i$ per ogni *i*. Inoltre, è possibile ricostruire il polinomio *q(x)* in modo efficiente; un modo per fare ciò è tramite l'uso della formula di *interpolazione di polinomi di Lagrange*:

sono "t"

$$\mathrm{q(x)} = \sum_{i=1}^{t+1} \ell_i(x) \cdot y_i$$

in cui il valore $\ell_i(x_m)$ può essere visto come il delta di Kronecker, ovvero:

$$\ell_i(x_m) = \delta_{im} = \begin{cases} 0 \; se \; m \neq i \\ 1 \; altrimenti \end{cases}$$

ottenibile tramite la seguente:

$$\ell_i(x) = \frac{(x - x_0)}{(x_i - x_0)} \cdots \frac{(x - x_{i-1})}{(x_i - x_{i-1})} \frac{(x - x_{i+1})}{(x_i - x_{i+1})} \cdots \frac{(x - x_{t+1})}{(x_i - x_{t+1})} = \prod_{\substack{0 \leq m \leq t+1 \\ m \neq i}} \frac{(x - x_m)}{(x_i - x_m)}$$

È immediato capire che il polinomio *q(x)* avrà grado pari a *t+1* o inferiore, e che:

$$\mathrm{q}(\mathrm{x}_m) = \mathrm{y}_m$$

inoltre, è possibile dimostrare che un polinomio così ottenuto è unico.

D'ora in poi si adopererà l'assunzione che tutti i calcoli saranno svolti prendendo come campo di esistenza quello dei numeri interi $\mathbb{Z}_p$ (insieme dei residui in modulo *p*), con *p* numero primo strettamente maggiore di *n*. Considerando tutto ciò, per condividere un segreto **s** si sceglie un polinomio casuale **q(x)** di grado massimo pari a **t** tale per cui **q(0)=s**. (Costruire un polinomio che ha quest'ultima qualità è triviale, basta imporre il termine noto dello stesso pari ad **s** ed i coefficienti di tutti gli altri addendi del polinomio pari ad un valore casuale appartenente a $\mathbb{Z}_p$) Poi, per ogni *i=1,…,n*, si distribuisce all'i-esimo party il valore $y_i=q(i)$ (per questo motivo è richiesto che *p>n*, così che ogni party possa avere un valore diverso). In fine, se un party vuole conoscere il segreto **s** necessiterà di altri **t** parties per poter interpolare il polinomio e calcolare **q(0)=s**. Ricordiamo che è richiesto un sottoinsieme di parties di cardinalità t+1 per ricavare il segreto, un sottoinsieme di cardinalità t o inferiore non potrà ricavare alcuna informazione a proposito del segreto **s**.

3.2 Protocolli SMPC a maggioranza onesta con segreto condiviso

La maggior parte dei protocolli SMPC viene sviluppata in una serie di passaggi, il primo dei quali prevede di rappresentare la funzione da dover eseguire, in modo distribuito, come se fosse un circuito booleano o aritmetico, composto da porte logiche o aritmetiche (si ricorda che i circuiti aritmetici e booleani sono *Turing completi*, dunque qualsiasi funzione può essere espressa tramite essi).

Nel caso in cui si scelga di utilizzare un circuito aritmetico per sviluppare un protocollo SMPC a maggioranza onesta e con segreto condiviso, allora il circuito andrà considerato rispetto al campo $\mathbb{Z}_p$ con *p>n*, come descritto nel paragrafo precedente. Tutti i parties appartenenti al sistema distribuito che eseguiranno tale protocollo avranno una copia del circuito, con l'assunzione che possano tutti comunicare tra di loro in modo sicuro. Consideriamo, per semplicità, un protocollo realizzato con l'intenzione di contrastare *attaccanti quasi onesti* (si veda il paragrafo 2.3.1), esso consisterà nelle seguenti fasi:

1. **Condivisione degli input:** In questa fase ogni party condivide il proprio input con gli altri parties tramite lo schema di condivisione segreta di Shamir. In particolare, lo schema è con soglia a (t+1)-su-n, con:

   $$t = \left\lfloor \frac{n-1}{2} \right\rfloor$$

   Così facendo il grado del polinomio sarà pari a **t** e ciò ci fornirà una sicurezza contro un insieme di parties corrotti che non formano una maggioranza assoluta, perché per ricostruire il segreto saranno necessari almeno **t+1** parties, ma **t+1** rappresenta una maggioranza. Dunque, ogni singolo party sarà in possesso di una parte del polinomio, necessaria a ricostruire il segreto, di ogni altro party.
2. **Valutazione del circuito:** In questa fase ogni party valuta il circuito, una porta per volta. Nell'eseguire una valutazione deve valere l'invariante per cui se il party possiede una parte del segreto (ovvero se possiede $q(x_i)$, con *q(x)* che è l'intero polinomio) secondo lo schema con soglia (t+1)-su-n per ognuno dei due valori di input della porta, allora alla fine dovrà possedere una parte del segreto secondo lo stesso schema con soglia (t+1)-su-n per il singolo valore di output, calcolato, della porta. Questa invariante, come tale, potrà non essere rispettata durante la valutazione, ma dovrà valere prima dell'inizio e subito dopo la conclusione della stessa.
   a) *Calcolo delle porte di addizione:* In accordo con l'invariante ogni party possiede una parte del segreto per ogni input della porta aritmetica del circuito; quindi, l'i-esimo party possiederà i valori *a(i)* e *b(i)*, dove *a(x)* e *b(x)* sono i rispettivi polinomi, necessari a

ricavare il segreto **s**, dei due input della porta aritmetica. L'output di questa porta deve essere una parte del segreto *a(0)+b(0)* secondo uno schema con soglia (t+1)-su-n (per rispettare l'invariante). Ciò è facilmente ottenibile dall'i-esimo party calcolando *a(i)+b(i)*, quindi ottenendo la parte del segreto *a(0)+b(0)*, che grazie alle proprietà additive dei polinomi mantiene un grado massimo pari a **t**, dunque si preserva la soglia dello schema di input. Si osservi come nessun tipo di comunicazione è necessaria al fine di calcolare le porte di addizione.

b) *Calcolo delle porte di moltiplicazione:* Come nel caso delle porte di addizione, ai due input della porta sono associati i polinomi *a(x)* e *b(x)* e l'i-esimo party possiede i valori *a(i)* e *b(i)*. Ogni party procede nel calcolo di $c(i)=a(i)\cdot b(i)$, ottenendo una parte del segreto $c(0)=a(0)\cdot b(0)$; a differenza del caso **a)**, ora *c(x)* sarà di grado **2t**, e non più **t**, dunque lo schema dell'output sarà con soglia (2t+1)-su-n, ma ciò viola l'invariante. Dunque, per poter portare a termine il calcolo delle porte di moltiplicazione sarà necessario che ogni party effettui una riduzione di grado, da **2t** a **t**, del polinomio risultante *c(x)* senza modificarne il suo valore *c(0)*, questo perché alla fine del calcolo l'invariante deve risultare rispettata. Prima di procedere nella descrizione del processo di riduzione di grado, si osservi che, visto che si è in un ambito in cui si ipotizza di avere una maggioranza onesta di partecipanti al sistema distribuito, e dunque che si ha $t < \frac{n}{2}$, le parti del segreto *c(x)* in possesso di tutti gli **n** parties sono sufficienti a poter interpolare il polinomio *c(x)* per poterne ricavare il segreto *c(0)*, perché *c(x)* risulta essere di grado **2t+1**, dunque nella peggiore delle ipotesi il grado sarà pari a:

$$\begin{cases} (2t+1) = n - 1 < n \\ t = \frac{n}{2} - 1 \end{cases}$$

Per capire come effettuare la riduzione di grado, si ipotizzi che tutti i parties posseggano due parti, indipendenti, dello stesso

segreto **r**, una appartenente ad un polinomio $R_{2t}(x)$ di grado *2t* e l'altra appartenente ad un polinomio $R_t(x)$ di grado *t*; quindi, si avrà che $R_{2t}(0)= R_t(0)=r$, e l'i-esimo party sarà in possesso di $R_{2t}(i)$ e $R_t(i)$. Ora, ogni party potrà calcolare, in locale, la quantità $d(i)=c(i)-R_{2t}(i)$ (si osservi come sia $c(x)$ che $R_{2t}(x)$ sono di grado *2t*, dunque anche $d(x)$ lo sarà). Procedendo, ogni party riceve le altre $d(i)$ degli altri parties, ricostruisce $d(x)$ e calcola $d(0)$ che è pari a:

$$d(0) = c(0) - R_{2t}(0) = a(0) \cdot b(0) - r$$

In fine l'i-esimo party calcola la parte del segreto di output della porta come il seguente:

$$c'(i) = R_t(i) + d(0)$$

Si osservi come $c'(i)$ sia di grado pari a **t**, perché $R_t(i)$ è di grado **t** e ad esso aggiungiamo una semplice costante, $d(0)$. Inoltre, visto che $R_t(0)=r$, si ha che:

$$c'(0) = R_t(0) + d(0) = r - a(0) \cdot b(0) - r = a(0) \cdot b(0)$$

dunque, ogni party avrà una parte del segreto $a(0) \cdot b(0)$ secondo uno schema con soglia (t+1)-su-n, soddisfacendo l'invariante. In tutto ciò, il valore $d(0)$ che viene rivelato a tutti i parties non fa trapelare alcuna informazione riguardante i segreti, proprio perché $R_{2t}(x)$, tramite l'operazione di somma aritmetica, maschera il segreto *r*, ma in particolare maschera il valore $a(0) \cdot b(0)$. In conclusione, rimane da capire come generare $R_{2t}(x)$ e $R_t(x)$ per ogni party. Per fare ciò, l'i-esimo party genera un valore segreto casuale r_i e ne condivide una parte con tutti gli altri parties tramite due polinomi, uno di grado **2t** e l'altro di grado **t**; chiameremo questi polinomi, rispettivamente, $R^i_{2t}(x)$ e $R^i_t(x)$. Una volta che ogni party ha ricevuto le due parti del segreto r_i per ogni i=1,…,n l'i-esimo party procederà a calcolare le seguenti:

$$R_{2t}(i) = \sum_{j=1}^{n} R^j_{2t}(i)$$

$$R_t(i) = \sum_{j=1}^{n} R^j_t(i)$$

In fine, visto che i valori r_i sono segreti, ovvero l'i-esimo è conosciuto solo dall'i-esimo party che lo genera (perché $R^i_{2t}(x)$ e $R^i_t(x)$ sono, a loro volta, conosciuti solo dall'i-esimo party che li genera), e visto che si ha la seguente:

$$r = \sum_{j=1}^{n} r_j$$

è immediato capire che *r* rimarrà segreto a tutti i parties, cioè nessun party lo conoscerà.

3. **Ricostruzione dell'output:** In questa fase, dopo che ogni party ha ottenuto la propria parte di output, può ricostruire l'intero output, tramite l'interpolazione di Lagrange, condividendo la propria parte con quelle degli altri parties. È importante sottolineare che, laddove lo si voglia, sarebbe possibile far si che parties diversi ricevano output diversi. Inoltre, nella maggior parte dei casi, gli unici parties a ricevere tutte le parti dell'output sono coloro che dovranno usare quell'output come input in un'altra porta aritmetica del circuito. Il protocollo descritto in questo paragrafo risulta essere sicuro nei casi in cui gli attaccanti sono *attaccanti quasi onesti* e i party onesti risultano essere una maggioranza, questo perché tutti i valori letti dai vari parties sono parti di segreti e dunque, non essendovi una maggioranza disonesta o corrotta, i segreti rimangono tali. Se ci si volesse proteggere da *avversari malevoli*, che possono alterare la normale esecuzione del protocollo, bisognerebbe utilizzare metodologie diverse all'interno del protocollo.

3.3 Private Set Intersection

Nel paragrafo precedente è stato descritto come un protocollo SMPC potesse essere realizzato per assicurare il calcolo sicuro di una qualsiasi funzione, in generale. In molti casi questi approcci "generali" risultano essere i più efficienti ma, nonostante ciò, una struttura più specifica di una funzione potrebbe essere sfruttata per poter sviluppare metodologie protocollari più efficienti. In questo, e

nel prossimo paragrafo, sono descritte delle funzioni con una struttura particolare; inoltre, saranno trattate alcune metodologie che sfruttano tali strutture. In un protocollo di tipo *Private Set Intersection (PSI)* vi sono due parties, ognuno con un insieme (in senso matematico) di valori; l'obiettivo è trovare l'intersezione tra questi due insiemi senza rivelare alcun valore che non appartiene a tale intersezione. Inoltre, in alcuni casi specifici vi è la necessità di mantenere tutti i valori segreti perché l'unico risultato desiderato è quello della cardinalità dell'insieme generato dall'intersezione. Nel tempo questo problema ha attirato molte attenzioni, con lo scopo di trovare una soluzione che potesse essere la più efficiente possibile; in questo paragrafo, per semplicità, si descriverà solo l'idea dietro un protocollo che fornisce una possibile soluzione. Per capire questa idea, si consideri una funzione con chiave, ovvero una funzione pseudo-casuale *F* tale per cui i valori di output da essa generati, in corrispondenza di input conosciuti, risultano essere a prima vista completamente casuali. Quindi, data una serie di valori $x_1,x_2,\ldots,x_n$, la serie di valori $F_k(x_1), F_k(x_2),\ldots, F_k(x_n)$ sembrerà casuale, più in particolare dato il valore $F_k(x_i)$ *sarà impossibile ricavare* x_i. Il protocollo che si basa su ciò utilizza uno strumento chiamato "*oblivious pseudorandom function evaluation"*; quest'ultimo è, a sua volta, un protocollo SMPC estremamente specifico, perché è richiesto che il primo party inserisca come input la chiave **k**, che il secondo party inserisca come input il valore **x** e poi riceva il valore $\boldsymbol{F_k(x)}$. Si osservi come il primo party non viene a conoscenza di **x**, il secondo non viene a conoscenza di **k** e non ottiene nient'altro che non sia $\boldsymbol{F_k(x)}$. Uno strumento del genere può essere costruito in molti modi, le cui descrizioni non trovano spazio nella corrente trattazione. Si considerino, ora, due parties, ognuno con il proprio insieme di elementi che indicheremo rispettivamente con $x_1,x_2,\ldots,x_n$ e $y_1,y_2,\ldots,y_n$ (si applica l'ipotesi semplificativa tale per cui la cardinalità di tali insiemi sia uguale e pari ad **n**; si sottolinea come tale ipotesi è semplificativa e non necessaria in generale), il protocollo procede secondo i seguenti passi:

1. Il primo party sceglie una chiave **k** per una funzione pseudo-casuale.
2. I due parties eseguono **n** *oblivious pseudorandom function evaluation*; alla i-esima esecuzione il primo party inserisce in input **k**, il secondo $\mathbf{y_i}$.

Dunque, il secondo party riceve come output $F_k(y_1), F_k(y_2),\ldots, F_k(y_n)$, mentre il primo non viene a conoscenza di $y_1,y_2,\ldots,y_n$.

3. Visto che il primo party conosce **k**, procede nel calcolo di $F_k(x_1), F_k(x_2),\ldots, F_k(x_n)$ e comunica tali valori al secondo party.
4. Il secondo party calcola l'intersezione tra la serie di valori $F_k(y_1), F_k(y_2),\ldots, F_k(y_n)$ e la serie di valori $F_k(x_1), F_k(x_2),\ldots, F_k(x_n)$, poi comunica i valori y_j tali per cui $F_k(y_j)$ appartiene all'intersezione (il secondo party può fare ciò perché egli conosce i valori $y_1,y_2,\ldots,y_n$).

Il protocollo appena descritto non rivela nulla ad eccezione dei valori presenti nell'intersezione, perché il secondo party non viene a conoscenza dei valori $x_1,x_2,\ldots,x_n$ ed il primo non viene a conoscenza dei valori $y_1,y_2,\ldots,y_n$ (eccezion fatta per quelli appartenenti all'intersezione che, come output del protocollo, sono pubblici). Inoltre, tale protocollo risulta essere sicuro nei casi in cui gli attaccanti sono *attaccanti quasi-onesti*, nel caso di *attaccanti malevoli* la sicurezza è molto più difficile da ottenere.

3.4 Crittografia con Soglia

Lo scopo della *crittografia con soglia* è quello di permettere ad un insieme di parties di portare a termine operazioni crittografiche, o di cifratura, senza che nessuno di loro conosca la chiave di cifratura. Ciò può essere utile quando c'è la necessità di convalidare una transazione mediante più firmatari, oppure quando si vuole prevenire il furto di una chiave di cifratura dividendola in varie parti e diffondendo le parti su dei dispositivi diversi (così che l'attaccante dovrà attaccare ogni singolo dispositivo per poter ricavare la chiave). In questo paragrafo si descriverà un semplice protocollo che permette a due parties di portare a termine una operazione di cifratura mediante cifrario *RSA*. Prima di descrivere tale protocollo è doveroso dire che riadattare lo stesso in un modello con più di due parties o con cifrari differenti è molto più difficile. Il cifrario *RSA* è un cifrario a chiave asimmetrica, dunque per essere usato richiede una chiave pubblica, formata da una coppia di valori e indicata con *(e,N)*, ed una privata, anch'essa formata da una coppia di valori (uno dei quali condiviso con la chiave

pubblica) e indicata con *(d,N)*. Per cifrare un testo in chiaro *x* si esegue la funzione $y = x^e \ mod \ N$, per decifrare un testo cifrato *y* si esegue la funzione inversa $x = y^d \ mod \ N$. La versatilità del cifrario *RSA* ha permesso, nel tempo, il suo utilizzo in svariati ambiti che vanno dalla semplice cifratura alla firma di documenti digitali; in questo paragrafo sarà sufficiente conoscere le funzioni di cifratura e decifrature, poc'anzi descritte, in quanto lo scopo è dimostrare come la funzione di decifratura possa essere calcolata da due parties distinti senza che nessuno dei due possa calcolarla in modo autonomo. Per ottenere ciò il sistema è strutturato in modo che i due parties posseggano, rispettivamente, le chiavi di decifratura (d_1,N) e (d_2,N), in cui d_1 e d_2 sono scelti casualmente rispettando il vincolo che $d_1 + d_2 = d$. Per poter calcolare in modo sicuro la funzione $y^d \ mod \ N$, il primo party calcola $x_1 = y^{d1} \ mod \ N$ ed il secondo calcola $x_2 = y^{d2} \ mod \ N$, poi questi valori vengono scambiati. In seguito, ogni party calcola la funzione:

$$x = x_1 \cdot x_2 \ mod \ N$$

e ne verifica la correttezza tramite la seguente uguaglianza:

$$x^e = y \ mod \ N$$

se la correttezza è verificata, allora *x* sarà considerato un output valido. Si osservi come questo calcolo è giustificato dalla seguente:

$$x = y^{d_1} \cdot y^{d_2} \ mod \ N \ = \ y^{d_1 + d_2 \ mod \ \phi(N)} \ mod \ N \ = \ y^d \ mod \ N$$

Inoltre, si osservi come il primo party (analogamente il secondo) può calcolare, date le informazioni in suo possesso, solo il valore x_2 (analogamente x_1), infatti si ha la seguente:

$$x_2 = \frac{x}{y^{d_1}} \ mod \ N$$

di cui si può verificare la correttezza grazie alla seguente:

$$x_2 = y^{d_2} = y^{d_1+d_2-d_1} = y^d \cdot y^{-d_1} = \frac{x}{y^{d_1}} \; mod \; N$$

questo vuol dire che il primo party (analogamente il secondo) non viene a conoscenza di nulla che non sia l'output del protocollo o i suoi input. Occorre sottolineare che la *crittografia con soglia* completa può coinvolgere più di due parties, può anche porre l'attenzione sul raggiungimento di un quorum di firme, come ad esempio far si che sia richiesto che *t+1* parties su *n* firmino. Ovviamente ciò può essere fatto ma in modo non banale, mantenendo comunque un'ottima efficienza. Recentemente la *crittografia con soglia* è stata oggetto di vari interessi per quanto riguarda il mondo delle criptovalute.

3.5 Protocolli SMPC con Maggioranza Non Onesta

Nel paragrafo **3.2** si è descritto un protocollo SMPC generale che può essere reputato sicuro sotto alcune condizioni; una di queste condizioni è che vi sia una maggioranza onesta. Nel caso in cui vi fosse una maggioranza non onesta, in tal caso ricade anche la configurazione con solo due parties con uno dei due corrotto, sarebbero richiesti approcci totalmente diversi non contenuti in questa trattazione. Nonostante ciò, è bene dire che, nel tempo, sono stati sviluppati svariati protocolli che operano in casi di maggioranza non onesta; alcuni di questi prestano più attenzione sull'efficienza, altri sull'applicabilità ecc.

3.6 Protocolli SMPC Efficienti e Pratici

I primi venti anni di ricerca nell'ambito dei protocolli SMPC hanno avuto lo scopo di studiare la fattibilità di quella, che all'epoca, era ancora un'idea; quindi, si cercava di definire il concetto di sicurezza e analogamente si cercava il modo di provarlo, si studiavano le precondizioni tali per cui un protocollo SMPC

potesse funzionare ecc. Negli anni seguenti, invece, le ricerche si sono concentrate sul cercare di capire come rendere i protocolli SMPC più efficienti; si è pensato, per esempio, di sviluppare algoritmi più efficienti, di diminuire il tempo richiesto per la computazione evitando l'uso di primitive crittografiche, si è arrivati persino ad operare sui supporti hardware. Inoltre, visto che molti protocolli SMPC richiedono che la funzione oggetto della computazione sia rappresentata come un circuito (booleano o aritmetico che sia), si è pensato di sviluppare dei compilatori appositi che generassero i circuiti a partire dal codice. Ovviamente questi compilatori non sono generali, ovvero sono creati su misura per rispettare le proprietà che un protocollo SMPC deve avere. Sapendo, dalla più basilare teoria riguardante i circuiti digitali, che le porte (logiche) di XOR richiedono meno risorse delle porte di AND ed OR, tali compilatori sono progettati in modo che prioritizzino l'uso di porte di XOR per fornire un'efficienza maggiore. Un altro fattore determinante l'efficienza riguarda i singoli protocolli, infatti alcuni di essi hanno una complessità maggiormente influenzata dalla profondità del circuito, mentre altri dall'ampiezza dello stesso. Dunque, alcuni compilatori realizzano i corrispondenti circuiti cercando di minimizzare la profondità o l'ampiezza degli stessi, in base al protocollo. In conclusione, nel tempo si è riuscito a sviluppare protocolli con un'efficienza tale da renderli utilizzabili in una vasta gamma di contesti.

Capitolo 4. Casi d’Uso

I protocolli SMPC sono, ormai, molto diffusi ed utilizzati all’interno di sistemi reali; essi risultano presenti all’interno di sistemi che si occupano di confrontare diversi DNA mantenendo la segretezza delle persone a cui quel DNA appartiene, sono utilizzati per ricavare delle statistiche senza rivelare nulla a proposito dei campioni analizzati, sono utilizzati in svariati ambiti. Si concluderà, dunque, questa trattazione descrivendo alcuni esempi e casi d’uso in cui i protocolli SMPC sono realmente utilizzati.

4.1 Divario Salariale di Boston

Nel 2017, il *Consiglio delle Donne Lavoratrici* di Boston ha utilizzato il *Secure Multi-party Computation* (SMPC) al fine di calcolare statistiche sulla retribuzione di 166.705 dipendenti in 114 aziende, corrispondenti approssimativamente al 16% della forza lavoro dell'area metropolitana di Greater Boston. L'utilizzo dell'MPC è stato fondamentale, poiché le aziende non avrebbero fornito i loro dati grezzi a causa delle preoccupazioni legate alla privacy. I risultati hanno dimostrato che il divario di genere nell'area di Boston è ancora più ampio rispetto a quanto stimato in precedenza dall'Ufficio delle Statistiche del Lavoro degli Stati Uniti. Questo è un potente esempio che dimostra come l'MPC possa essere utilizzato per il bene sociale.

4.2 Conversione Pubblicitaria

Per calcolare in modo accurato gli acquisti correlati o influenzati da una pubblicità (*conversione pubblicitaria*), l’azienda Google calcola la dimensione dell'intersezione (in senso matematico) tra l'elenco delle persone a cui è stato

mostrato un annuncio e l'elenco delle persone che effettivamente hanno acquistato i beni pubblicizzati. Quando i beni non vengono acquistati online e quindi non si può risalire all'annuncio mostrato a partire dall'acquisto effettuato, Google e l'azienda che paga per la pubblicità devono condividere i rispettivi elenchi al fine di calcolare la dimensione dell'intersezione. Per calcolare ciò senza rivelare altro se non la dimensione dell'intersezione, Google utilizza un protocollo di *Private Set Intersection* preservando la privacy. Il protocollo utilizzato da Google è descritto in [5]. Sebbene questo protocollo sia lontano dall'essere il più efficiente conosciuto oggi, è semplice e soddisfa i loro requisiti computazionali.

4.3 Protocolli SMPC per la Protezione Crittografica di Chiavi

Come descritto nel paragrafo **3.4**, la crittografia con soglia offre la possibilità di eseguire operazioni crittografiche (come decifratura e firma) senza che la chiave privata sia conservata in un singolo luogo. Diverse aziende hanno deciso di applicare la crittografia con soglia come alternativa all'hardware tradizionale dedicato alla protezione delle chiavi crittografiche. In questa applicazione, i protocolli SMPC non vengono eseguiti da parties diversi che detengono, ognuno, alcune informazioni private. Piuttosto, un'organizzazione singola utilizza uno o più protocolli SMPC per generare chiavi ed eseguire operazioni crittografiche, senza che la chiave sia mai in un unico luogo dove possa essere rubata. Posizionando le parti delle chiavi in diversi ambienti, è molto difficile per un avversario rubare tutte le parti e ottenere la chiave. In questo contesto, il *modello di corruzione proattivo* descritto nel paragrafo **2.3.2** è il più adatto. Un altro utilizzo dei protocolli SMPC in questo contesto riguarda la protezione delle chiavi di firma utilizzate per proteggere criptovalute e altre risorse digitali. Qui, la capacità di definire quorum generali consente l'applicazione crittografica di politiche rigorose per l'approvazione delle transazioni finanziarie, o per condividere le chiavi tra fornitori di servizi (come, ad esempio, i fornitori di portafogli digitali) e clienti.

4.4 Collaborazione Governativa

Diversi dipartimenti governativi detengono informazioni sui cittadini, e considerevoli benefici possono essere ottenuti correlando tali informazioni. Tuttavia, i rischi per la privacy legati all'aggregazione di informazioni private possono impedire ai governi di procedere in questo senso. Ad esempio, nel 2000, il Canada ha abbandonato un programma per aggregare le informazioni dei cittadini, a seguito delle critiche che lo definivano come la creazione di un "database Grande Fratello". Utilizzando un protocollo SMPC, l'Estonia ha raccolto dati cifrati riguardanti le dichiarazioni dei redditi e dei percorsi di istruzione superiore al fine di analizzare se gli studenti che lavorano durante il loro percorso di studio hanno una probabilità maggiore di conseguire risultati peggiori rispetto a coloro che si concentrano esclusivamente sugli studi. Utilizzando il *Secure Multi-party Computation*, il governo garantiva che tutte le normative sulla protezione dei dati e il segreto fiscale fossero rispettati senza perdere l'utilità dei dati.

4.5 Analisi a Salvaguardia della Privacy

L'uso del machine learning sta aumentando rapidamente in molti settori. Il *Secure Multi-party Computation* può essere utilizzato per eseguire modelli di machine learning sui dati senza rivelare il modello (che contiene una preziosa proprietà intellettuale) al proprietario dei dati e senza rivelare i dati al proprietario del modello. Inoltre, le analisi statistiche possono essere condotte tra organizzazioni per scopi di contrasto al riciclaggio di denaro, calcolo dei punteggi di rischio e altro ancora.

Conclusioni

Il *Secure Multi-party Computation* è un esempio di successo nel lungo lavoro che è quello della ricerca. Nei primi venti anni di ricerca nessuna applicazione pratica dello stesso era visibile, anzi si tendeva a mettere in dubbio una possibile applicazione pratica del *Secure Multi-party Computation* per il futuro. Nell'ultimo decennio, l'usabilità del *Secure Multi-party Computation* ha subito una trasformazione radicale. In questo periodo, il *Secure Multi-party Computation* non solo è diventato sufficientemente veloce da essere utilizzato in pratica, ma ha ricevuto riconoscimenti dai settori in cui esso è applicato ed è passato dall'essere una tecnologia in via di sviluppo a una tecnologia effettivamente implementata. Il *Secure Multi-party Computation* richiede ancora una grande competenza per essere implementata, e ulteriori scoperte nella ricerca sono necessarie per rendere la computazione sicura praticabile su grandi quantità o insiemi di dati, per problemi complessi e per renderla facile da utilizzare per i non esperti. I progressi degli ultimi anni e la grande quantità di ricerca applicata che si sta producendo dipingono un futuro positivo per il *Secure Multi-party Computation* nella pratica. Insieme a ciò, la ricerca teorica sul *Secure Multi-party Computation* continua, garantendo che le applicazioni pratiche dei protocolli SMPC siano basate su solide basi scientifiche.

Bibliografia

[1] Aumasson, J. Serious Cryptography. No Starch Press, San Francisco, 2018.

[2] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic FaultTolerant Distributed Computation. 20th STOC, 1988.

[3] D. Chaum, C. Cr´epeau and I. Damg˚ard. Multi-party Unconditionally Secure Protocols. In the 20th STOC, pages 11–19, 1988.

[4] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In the 19th STOC, pages 218–229, 1987. Details in Foundations of Cryptography: Volume 2 – Basic Applications (Cambridge University Press 2004), by Oded Goldreich.

[5] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan and M. Yung. Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions. IACR Cryptology ePrint Archive, report 2017:738, 2017.

[6] Jacobolus, Willondon. Lagrange polynomial. https://en.wikipedia.org/wiki/Lagrange_polynomial , 2023.

[7] Lindell, Y. Secure Multiparty Computation. Communications of the ACM, Vol. 64, No. 1, pp. 86-96, 2021.

[8] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In the 21st STOC, pages 73–85, 1989.