UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

Sicurezza Informatica

Progetto in Crittografia

ATTRIBUTE BASED ENCRYPTION (ABE)

SARA LONGO
Matricola 933528
BIAGIO DIPALMA
Matricola 938473

Indice dei contenuti

Indice delle figure

CAPITOLO 1: INTRODUZIONE

Il veloce progresso delle tecnologie informatiche e delle reti di telecomunicazioni hanno portato ad uno spostamento progressivo delle capacità di calcolo e di archiviazione dati nel *cloud*, offrendo benefici sia a singole persone sia ad organizzazioni, quali ad esempio, l'accesso ubiquo ai propri dati e la possibilità di salvare grandi quantità di informazioni a condizioni economicamente vantaggiose.

La possibilità di memorizzare dati su server remoti pone enormi problemi legati alla sicurezza, in quanto essi sono (per lo più) dati sensibili (finanziari, cartelle mediche, dati relativi a pubbliche amministrazioni, oltre a dati personali scambiati, ad esempio, attraverso social network) e non sono sotto il controllo dei legittimi possessori, poiché entità potenzialmente malevole potrebbero accedervi.

1.1 I limiti della crittografia tradizionale

Usualmente, le tecniche ed i metodi utilizzati per proteggere i dati sono di natura crittografica. La crittografia tradizionale (ad esempio la crittografia a chiave pubblica) non fornisce gli strumenti necessari per garantire la sicurezza di grandi quantità di dati complessi. Infatti, gli strumenti crittografici tradizionali consentono un accesso '*coarse-grained*' ai

dati. Più precisamente, solamente un'unica chiave può decifrare i dati criptati. D'altro canto, un individuo potrebbe voler condividere in modo selettivo i propri dati personali e documenti. Analogamente, un'azienda potrebbe voler condividere dati con diversi insiemi di utenti, in funzione delle loro credenziali.

In applicazioni quali il Data Mining su dati medici cifrati o su dati provenienti da reti sociali, è utile fornire accesso parziale ai dati e permettere di eseguire computazioni sui dati criptati, ad esempio, permettere di calcolare statistiche o eseguire interrogazioni.

Idealmente, si vorrebbe poter criptare i dati in modo tale da garantire un controllo degli accessi '*fine-grained*' e la capacità di eseguire calcoli in modo selettivo direttamente sui dati criptati e cioè si vorrebbe avere il controllo su chi ha accesso ai dati criptati e su cosa si può calcolare, avendo a disposizione tali dati. In questo modo, verrebbe riconciliato il conflitto tra la possibilità di dare in outsourcing la gestione e le computazioni sui dati e la necessità di proteggerli.

1.2 Crittografia funzionale o basata su attributi

Nel corso degli ultimi dieci anni, è emerso un nuovo paradigma collegato alla crittografia a chiave pubblica che ha come obiettivi la crittografia basata su attributi (*attribute-based encryption*, ABE), che permette il controllo degli accessi fine-grained e una sua generalizzazione, la crittografia funzionale, che permette la computazione selettiva sui dati cifrati.

Più precisamente:

– *nella crittografia basata su attributi*, i dati criptati sono associati ad attributi ed a chiavi di decifratura segrete, insieme a regole che stabiliscono quali dati cifrati possono essere decriptati da quali chiavi;

– *nella crittografia funzionale*, un utente in possesso di una chiave segreta può apprendere una funzione specifica dei dati cifrati, e solo quella funzione. Ad esempio, la decriptazione di una colonna di valori numerici con una chiave corrispondente alla media aritmetica rivela il valore di tale statistica e niente altro.

Una caratteristica peculiare sia della crittografia basata su attributi sia di quella funzionale consiste nel fatto che possano esserci diverse chiavi segrete, ognuna con differenti capacità di decriptazione. Il paradigma di questa crittografia può essere inteso come una generalizzazione di casi più specifici di estensioni della crittografia a chiave pubblica apparse nel corso degli anni, quali la crittografia *broadcast*, la crittografia basata su identità, o la crittografia che permette l'esecuzione di interrogazioni direttamente sui dati cifrati (*searchable encryption*, SE).

Gli obiettivi che lo studio della crittografia sia basata su attributi sia funzionale sono quindi:

- la definizione di schemi di cifratura che possono esprimere un ampio insieme di politiche di accesso e funzioni;
- la realizzazione di implementazioni (il più possibile) efficienti di tali schemi, basandosi su problemi computazionali che sono (quasi) ritenuti molto difficili da risolvere in modo efficiente.

I più semplici esempi di schemi di cifratura basati su attributi sono i cosiddetti schemi basati sull'identità (*identity-based encryption*, IBE), dove sia il messaggio criptato sia la chiave segreta di decifratura sono associati a identità e la decifratura è possibile solo quando le due identità coincidono.

CAPITOLO 2: ABE

La crittografia basata su attributi, proposta da Amit Sahai e Brent Waters, è un tipo di crittografia a chiave pubblica in cui la chiave privata di un utente e il testo cifrato dipendono dai suoi attributi, come ad esempio il luogo di residenza, la posizione, il tipo di account o di abbonamento che possiede.

La decrittazione del testo cifrato è possibile solo se l'insieme di attributi della chiave utente corrisponde agli attributi del testo cifrato.

In un sistema ABE qualsiasi stringa può potenzialmente fungere da attributo. Inoltre, gli attributi possono essere valori numerici e le politiche possono contenere intervalli su questi valori. L'insieme di attributi utilizzati dipenderà dall'applicazione designata.

2.1 Tipi di controllo degli accessi e ABE

Esistono tre tipi di controllo degli accessi: Controllo degli accessi in base all'Utente (UBAC), il Controllo degli accessi in base al Ruolo (RBAC) e il Controllo degli accessi in base agli Attributi (ABAC).

In UBAC, l'Access Control List (ACL) contiene l'elenco di utenti autorizzati ad accedere ai dati. Questo non è fattibile nel Cloud, data la presenza di molti utenti.

In RBAC, gli utenti sono classificati in base ai loro ruoli individuali, definiti dal sistema. I dati sono accessibili solo da quegli utenti che ne hanno le competenze.

In ABAC, agli utenti sono assegnati degli attributi e i dati sono legati ad una politica di accesso. Solo gli utenti con un set di attributi valido, che soddisfano i criteri di accesso, possono accedere ai dati. I lavori basati sull'ABAC usano come primitiva crittografica l'ABE.

2.1.1 Relazione tra Attribute Based Encryption e Access Control

L'ABE fornisce un modo sicuro e automatico per applicare il controllo degli accessi, ma il controllo degli accessi non necessariamente comporta la crittografia.

In ABE, le risorse a cui si desidera accedere vengono cifrate da un "controller"; i dati vengono resi pubblici e ad ogni utente viene concessa una chiave corrispondente al suo diritto di accesso specifico. Quindi, il controller non ha bisogno di interagire con gli utenti per gestire il loro accesso, poiché tale verifica del controllo degli accessi può essere garantito dalla sicurezza dello schema di crittografia sottostante.

Tuttavia, in un'implementazione generale di controllo dell'accesso, un "controller" (ad esempio un sistema operativo o un'applicazione) potrebbe dover interagire con l'utente (ad esempio tramite un meccanismo di controllo dell'accesso al sistema operativo) e concedergli il diritto di accesso corretto

ogni volta che l'utente lo desidera. Le risorse qui potrebbero non essere cifrate, ma sono disponibili solo per il "controller".

2.1.2 Schemi di crittografia basata sugli attributi

Controllo dell'accesso basato sul contenuto

In un sistema il cui controllo degli accessi è basato sul contenuto, gli attributi vengono associati ad un testo cifrato quando si criptano dati sensibili, mentre la chiave privata viene associata a una politica su questi attributi; in genere la politica verrà espressa come formula booleana. Nella letteratura accademica questa variante viene definita *Key-Policy Attibute Based Encryption (KP-ABE).*

La chiave privata può decifrare un testo cifrato se e solo se la sua politica (formula booleana) è soddisfatta dagli attributi del testo.

Un utente, i cui attributi e chiavi sono stati revocati, non può riscrivere informazioni non aggiornate.

Le chiavi segrete degli utenti vengono generate in base a un albero di accesso che definisce l'ambito dei privilegi dell'utente interessato.

Ad esempio, da e-mail criptate si potrebbero considerare informazioni quali indirizzi mittente e destinatario, data e ora di invio e oggetto come attributi e si potrebbe cifrare il corpo della e-mail come dato segreto. La chiave privata generata da un'autorità potrebbe consentire la decifratura di quelle e-mail che soddisfano la politica:
TO: student@unimi.it OR (subject: examination result AND date > Jan, 1 2019.

Lo schema KP-ABE è costituito dai seguenti quattro algoritmi:

- *Setup*: questo algoritmo accetta come input un parametro di sicurezza κ e restituisce la chiave pubblica PK e una chiave segreta del sistema master MK. PK viene utilizzato dai mittenti dei messaggi per la crittografia. MK viene utilizzato per generare chiavi segrete dell'utente ed è noto solo all'Autorità.
- *Crittografia*: questo algoritmo accetta come input un messaggio M, la chiave pubblica PK e un set di attributi. Emette il testo cifrato E.
- *Generazione delle chiavi*: questo algoritmo accetta come input una struttura di accesso T e la chiave segreta principale MK. Emette una chiave segreta SK che consente all'utente di decriptare un messaggio crittografato sotto un insieme di attributi se e solo se è uguale a T.
- *Decrittazione*: prende come input la chiave segreta SK dell'utente per la struttura di accesso T e il testo cifrato E, che è stato crittografato sotto il set di attributi. Questo algoritmo genera il messaggio M se e solo se l'attributo set soddisfa la struttura di accesso dell'utente T.

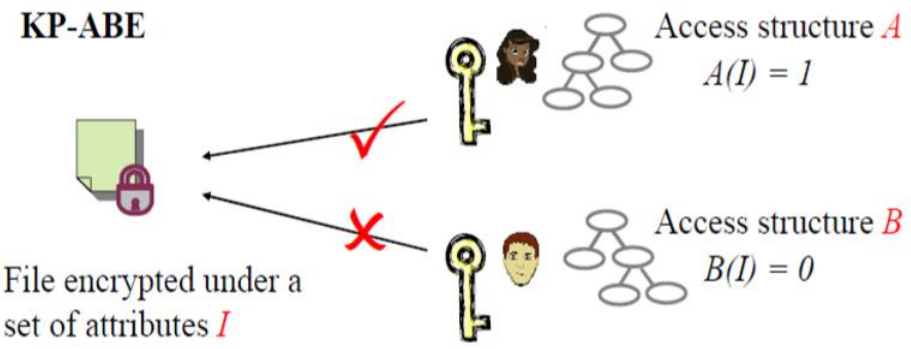


Figura 2-1: Key-Policy Attribute Based Encryption.

Controllo degli accessi in base al ruolo

Un sistema ABE per il controllo degli accessi in base al ruolo "inverte" la semantica del controllo degli accessi in base al contenuto.

In tali sistemi gli attributi sono spesso associati alle credenziali del possessore della chiave privata.

Nella letteratura accademica questa variante viene definita *Ciphertext-Policy Attribute Based Encryption* (CP-ABE).

In questo caso, gli alberi d'accesso sono utilizzati per criptare i dati; le chiavi segrete degli utenti vengono generate su un set di attributi. L'albero è una struttura d'accesso monotona con AND e OR, in cui gli attributi costituiscono le foglie. È un approccio centralizzato, dato che un solo KDC (Key Distribution Center) rappresenta il punto di errore.

Ad esempio, in un sistema ABE per un'azienda, uno sviluppatore di software potrebbe avere una chiave privata associata ad attributi per ogni progetto a cui ha lavorato, come il tipo di progetto, il nome, la data di inizio di lavoro.

Il messaggio è criptato in base a una struttura d'accesso in modo tale che solo coloro i cui attributi soddisfano la struttura possano decriptarlo.

Con la tecnica CP-ABE, i dati criptati possono essere mantenuti riservati e protetti dagli attacchi di collusione.

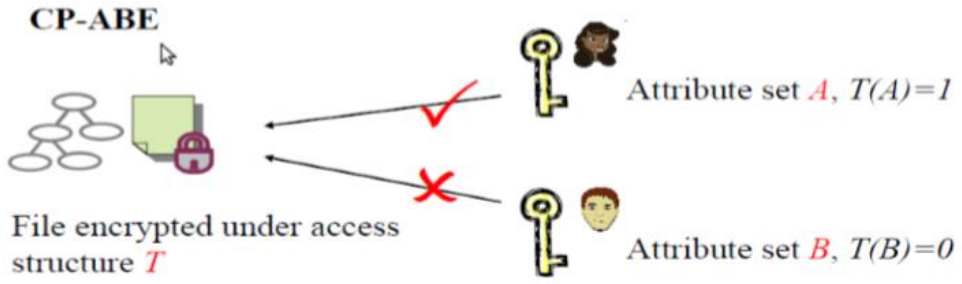


Figura 2-2: Ciphertext-Policy Attribute Based Encryption.

Lo schema CP-ABE è costituito dai seguenti quattro algoritmi:

- *Setup*: questo algoritmo accetta come input un parametro di sicurezza κ e restituisce la chiave pubblica PK e una chiave segreta principale di sistema MK. PK viene utilizzato dai mittenti dei messaggi per la crittografia. MK viene utilizzato per generare chiavi segrete dell'utente ed è noto solo all'autorità.
- *Crittografia*: questo algoritmo accetta come input il parametro pubblico PK, un messaggio M e una struttura di accesso T. Emette il testo cifrato CT.
- *Generazione delle chiavi*: questo algoritmo accetta come input una serie di attributi associati all'utente e alla chiave segreta principale MK. Emette una chiave segreta SK che consente all'utente di decriptare un messaggio crittografato sotto una struttura ad albero di accesso T se e solo se corrisponde a T.
- *Decrittazione*: questo algoritmo accetta come input CT testo cifrato e una chiave segreta SK per un set di attributi. Restituisce il messaggio M se e solo se soddisfa la struttura di accesso associata al CT cifrato.

Controllo di accesso basato sui ruoli multi-autorità

Un problema presente nel controllo di accesso basato sui ruoli è che in molte applicazioni si vorrebbero scrivere politiche di controllo degli accessi che si estendano attraverso diversi confini amministrativi.

Una difficoltà con ABE standard è che richiede una singola autorità per distribuire le chiavi private. Tuttavia, in molte applicazioni è naturale per le diverse autorità gestire diversi attributi. Un sistema ABE multi-autorità consente di associare un testo cifrato a una politica scritta attraverso attributi emessi da diverse autorità: sono presenti quindi numerosi KDC coordinati da un'autorità fidata.

Un sistema Ciphertext-Policy ABE multi-autorità comprende i seguenti cinque algoritmi:

- *Global Setup (λ) → GP*
  L'algoritmo di global setup accetta il parametro di sicurezza λ e genera i parametri globali GP per il sistema;

- *Authority Setup(GP) → SK, PK*
  Ogni autorità esegue l'algoritmo di authority setup con GP come input per produrre la propria coppia di chiavi segreta e pubblica (SK, PK);

- *Encrypt (M, (A, ρ), GP, {PK}) → CT*
  L'algoritmo di crittografia accetta un messaggio M, una matrice di accesso (A, ρ), il set di chiavi pubbliche per le autorità pertinenti e i parametri globali. Emette un CT (ciphertext);

- *KeyGen (GID, GP, i, SK) → Ki, GID*
  L'algoritmo di generazione delle chiavi comprende un GID di identità, i parametri globali, un attributo i appartenente a qualche autorità e la chiave segreta SK per questa autorità. Produce una chiave Ki, GID per queste coppia attributo, identità;

- *Decrypt (CT, GP, {Ki, GID}) → M*
  L'algoritmo di decrittografia include i parametri globali, il testo cifrato e una raccolta di chiavi corrispondenti all'attributo, le coppie di identità tutte con lo stesso GID di identità fisso. Emette il messaggio M quando la raccolta di attributi soddisfa la matrice di accesso corrispondente al testo cifrato. Altrimenti, la decodifica non riesce.

2.2 Limitazioni delle soluzioni alternative all'ABE

2.2.1 Crittografia basata sull'identità

Nella crittografia a chiave pubblica standard, si cifra la mail di un utente recuperando la chiave da un repository di chiavi pubbliche.

In un sistema IBE (Identity Based Encryption) questo passaggio può essere ignorato e si può applicare la cifratura con la sola conoscenza della stringa (ad esempio l'indirizzo e-mail) del destinatario target.

Sebbene la rimozione della fase di recupero delle chiavi sia utile in alcune circostanze, IBE segue ancora il modello tradizionale di crittografia punto a punto in cui un testo cifrato è indirizzato a un singolo utente identificato e non supporta un controllo degli accessi più flessibile o espressivo.

2.2.2 Server enforced access control

Un'alternativa consiste nell'utilizzare un server di archiviazione affidabile per imporre il controllo flessibile dell'accesso. In tale configurazione, un server attendibile memorizzerà le informazioni in forma non cifrata. Il client invierà le sue credenziali di accesso (firmate) e il server determinerà se è autorizzato ad accedere a un determinato elemento di dati e, in tal caso, rispedirli.

Questa soluzione consentirà un controllo flessibile dell'accesso, ma presenta numerosi inconvenienti:

- Se il server di archiviazione è compromesso, la riservatezza dei dati viene persa;

- Il client deve disporre dell'accesso di rete continuo a un server per accedere ai dati sensibili;
- Il client deve contattare il server per ogni elemento di dati a cui desidera accedere;
- Questa soluzione non si traduce nell'impostazione multi-autorità in quanto una parte deve essere responsabile della gestione del server;
- Per motivi di robustezza, è consigliabile archiviare dati criptati in siti non attendibili.

In conclusione, la crittografia basata sugli attributi consente un controllo degli accessi molto espressivo e una condivisione flessibile dell'accesso senza l'affidamento su una terza parte attendibile.

2.3 Esempio di uso di ABE

Alice vuole condurre delle ricerche su un modo efficiente per ricattare le persone. Chiede a Bob, il CTO, di preparare una proposta e un budget. Bob crea il bilancio e condivide il documento confidenziale con più gruppi di persone: i vicepresidenti della ricerca e tutti i membri del dipartimento finanziario.

Utilizzando la crittografia basata sull'identità, avrebbe dovuto creare più versioni criptate di questo documento. Mentre con la crittografia basata sugli attributi, Bob può cifrare il documento con gli attributi *vp-research* and *finance* e specificare una politica in modo tale che almeno uno di questi attributi sia necessario per la decryption. Chiunque abbia uno di questi attributi come parte della propria identità può decifrare il documento.

Il sistema cifrerà il documento costruendo la chiave pubblica appropriata utilizzando gli attributi e la politica. Quando Alice (o un altro utente) tenterà di accedere al documento, richiederà la creazione di una chiave di decrittazione privata da parte del PKG. Il PKG verificherà che Alice abbia l'attributo *finance* come parte della sua identità e creerà la chiave. Alice quindi utilizzerà la chiave per visualizzare il documento.

2.3.1 Possibili algoritmi di cifratura e decifratura

RSA

RSA è basato sull'elevata complessità computazionale della fattorizzazione in numeri primi. Il suo funzionamento base è il seguente:

- si scelgono a caso due numeri primi, p e q abbastanza grandi da garantire la sicurezza dell'algoritmo (ad esempio, il più grande numero RSA, RSA-2048, utilizza due numeri primi lunghi più di 300 cifre)
- si calcolano il loro prodotto $n = p \times q$, chiamato *modulo* e il prodotto $f(n) = (p-1)(q-1)$
- si considera che la fattorizzazione di n è segreta e solo chi sceglie i due numeri primi, p e q, la conosce
- si sceglie poi un numero e (chiamato *esponente pubblico*), coprimo con $f(n)$ e più piccolo di f(n)
- si calcola il numero d (chiamato *esponente privato*) tale che il suo prodotto con e sia congruo a 1 modulo f(n) ovvero che $ed \equiv 1 \pmod{\varphi(n)}$

La chiave pubblica è (n,e), mentre la chiave privata è (n,d).

La forza dell'algoritmo sta nel fatto che per calcolare d da e (o viceversa) non basta la conoscenza di n ma serve il numero $f(n)$, e che il suo

calcolo richiede tempi molto elevati; infatti fattorizzare in numeri primi (cioè scomporre un numero nei suoi divisori primi) è un'operazione computazionalmente costosa.

Un messaggio *m* viene cifrato attraverso l'operazione $m^e \pmod{n}$ trasformandolo nel messaggio *cifrato c*. Una volta trasmesso, viene decifrato con $c^d \pmod{n} = m$. Il procedimento funziona solo se m < n e la chiave e utilizzata per cifrare e la chiave d utilizzata per decifrare sono legate tra loro dalla relazione $ed \equiv 1 \pmod{\varphi(n)}$, quindi quando un messaggio viene cifrato con una delle due chiavi può essere decifrato solo utilizzando l'altra.

El Gamal

ElGamal è un sistema di cifratura a chiave pubblica, proposto dal ricercatore egiziano-americano Taher Elgamal nel 1985. Lo schema è basato sulla difficoltà del calcolo del logaritmo discreto, l'operazione inversa della potenza discreta.

Sia A un insieme contenente i numeri interi compresi tra 0 e p-1, dove p è un numero primo: $A = \{0, 1, 2, \ldots, p-1\}$.

Si definisca un'operazione su due numeri a e b appartenenti all'insieme A: $a \otimes b = a^b \bmod p$,dove mod è l'operazione di modulo e l'operazione è la potenza discreta.

Si definisce "logaritmo discreto di un numero x in base a" quel numero b tale che $a \otimes b = x$, cioè $a^b \bmod p = x$.

Generazione delle chiavi

L'utente A genera e rende nota una chiave pubblica: $Y_A = (\alpha^{X_A} \bmod q)$

Analogamente l'utente B genera la sua chiave pubblica: $Y_B = (\alpha^{X_B} \bmod q)$

Dove:

- q numero grande primo (es. dell'ordine di 10^{100}), parametro globale.
- α radice primitiva di q, parametro globale.
- X_A scelto a caso in maniera uniforme tra [*1,(q-1)*], che costituisce la chiave privata di A e deve essere mantenuto segreto.
- X_B scelto a caso in maniera uniforme tra [*1,(q-1)*], che costituisce la chiave privata di B e deve essere mantenuto segreto.

Cifratura

L'utente A che vuole inviare un messaggio M a B, con $M < q$, sceglie a caso un numero k nell'intervallo [*1,(q-1)*] e calcola:

$$K_A = (Y_B)^k \bmod q$$

Dopodiché genera il messaggio da inviare come una coppia (C_1,C_2) formata da:

$$C_1 = \alpha^k \bmod q$$
$$C_2 = K_A M \bmod q$$

Decifratura

Il testo cifrato *(C_1,C_2)* viene inviato a B il quale recupera M nel seguente modo:

$$K_B = (C_1^{X_B}) \bmod q$$
$$M = (C_2 \cdot K_B^{-1}) \bmod q$$

Conclusioni

Tutte le operazioni coinvolte sono algoritmicamente fattibili, in maniera efficiente. I costi computazionali di cifratura e decifratura sono paragonabili all'RSA, ma con una espansione del testo cifrato di un fattore 2 rispetto al testo in chiaro.

Questo algoritmo è resistente ad attacchi di crittoanalisi: l'unico modo per ricavare informazioni segrete dai dati pubblici è effettuare il logaritmo discreto. Ancora oggi non è conosciuto un algoritmo efficiente per calcolare tali valori.

2.4 Meccanismo di revoca degli attributi

Sebbene il concetto ABE sia molto potente, i sistemi presentano principalmente due inconvenienti: non efficienza e inesistenza del meccanismo di revoca degli attributi.

Mentre nei sistemi PKI tradizionali le coppie di chiavi pubbliche/ private sono associate in modo univoco a un singolo utente, nei sistemi basati su attributi, ogni attributo può appartenere a utenti diversi. In un sistema ABE, ciò che viene revocato è l'attributo, non l'utente o la chiave.

Una soluzione semplice ma vincolata è quella di includere un attributo time. Questa soluzione richiederebbe che ogni messaggio sia cifrato con un albero di accesso modificato T0, che viene costruito aumentando l'albero di accesso originale T con un attributo temporale aggiuntivo.

L'attributo time, ζ rappresenta l'attuale "periodo". Ad esempio, ζ può essere l'attributo "data" il cui valore cambia una volta al giorno.

Si presume che ogni utente non revocato riceva le sue nuove chiavi private corrispondenti all'attributo "data" una volta al giorno direttamente

dal server di chiavi mobile MKS (che è l'autorità centrale) o tramite i delegati regionali. Con una struttura di accesso gerarchica, la proprietà di delega chiave di CP-ABE può essere sfruttata per ridurre la dipendenza dall'autorità centrale per l'emissione delle nuove chiavi private a tutti gli utenti ogni intervallo di tempo.

Esistono importanti compromessi tra il carico aggiuntivo sostenuto dall'autorità per la generazione e la comunicazione delle nuove chiavi agli utenti e la quantità di tempo che può trascorrere prima che un utente revocato possa essere effettivamente eliminato.

Questa soluzione ha i seguenti problemi:

- Ogni utente deve ricevere periodicamente dall'autorità centrale la nuova chiave privata corrispondente all'attributo time, altrimenti non sarà in grado di decifrare alcun messaggio.
- È una tecnica di revoca lenta, dato che l'utente revocato non viene eliminato dal sistema fino alla scadenza del periodo di tempo corrente.
- Questo schema richiede una sincronizzazione temporale implicita tra l'autorità e gli utenti.

2.5 Altri problemi dell'ABE

Central Trust

La crittografia basata su attributi richiede la fiducia in un'autorità centrale: il Private Key Generator (simile alla crittografia basata su identità). Questo lo rende più appropriato nelle implementazioni aziendali.

Velocità

La crittografia basata sugli attributi (soprattutto la CP-ABE) è lenta, poiché comporta la creazione di un "albero delle politiche" e molto costosa dal punto di vista della decryption.

Entrambe le costruzioni ABE sono lente rispetto alla crittografia classica e diventano man mano sempre più costose all'aumentare del numero di attributi su un determinato dato. Ciò è dovuto allo specifico costrutto matematico utilizzato (il Weil Pairing), menzionato sopra.

ABE deve generare una nuova chiave privata ogni volta che si vuole decifrare un documento. Negli altri sistemi, invece, si utilizza sempre la stessa chiave.

CAPITOLO 3: IL PROGETTO

3.1 Introduzione

Nel progetto sviluppato si è deciso di analizzare il funzionamento della crittografia asimmetrica, prendendo in considerazione un cifrario dall'implementazione semplice ma estremamente potente come ElGamal.

Successivamente, nel modulo ElGamalABE.py, è stato introdotto il concetto dell'ABE con l'obiettivo di creare un cifrario dove tutti gli utenti potessero condividere chiavi pubbliche e private e nel quale l'ABE sarebbe stato l'elemento fondamentale per discriminare le diverse cifrature.
Tutto ciò ha come obiettivo quello di godere della sicurezza di un cifrario a chiave pubblica e della versatilità della cifratura basata su attributi.

Inoltre, sempre nel modulo ElGamalABE.py, si mette a confronto la cifratura fatta da un utente A con i propri attributi ed un eventuale tentativo di un utente B di decifrare Il ciphertext di A.

3.2 ElGamalClassico.py

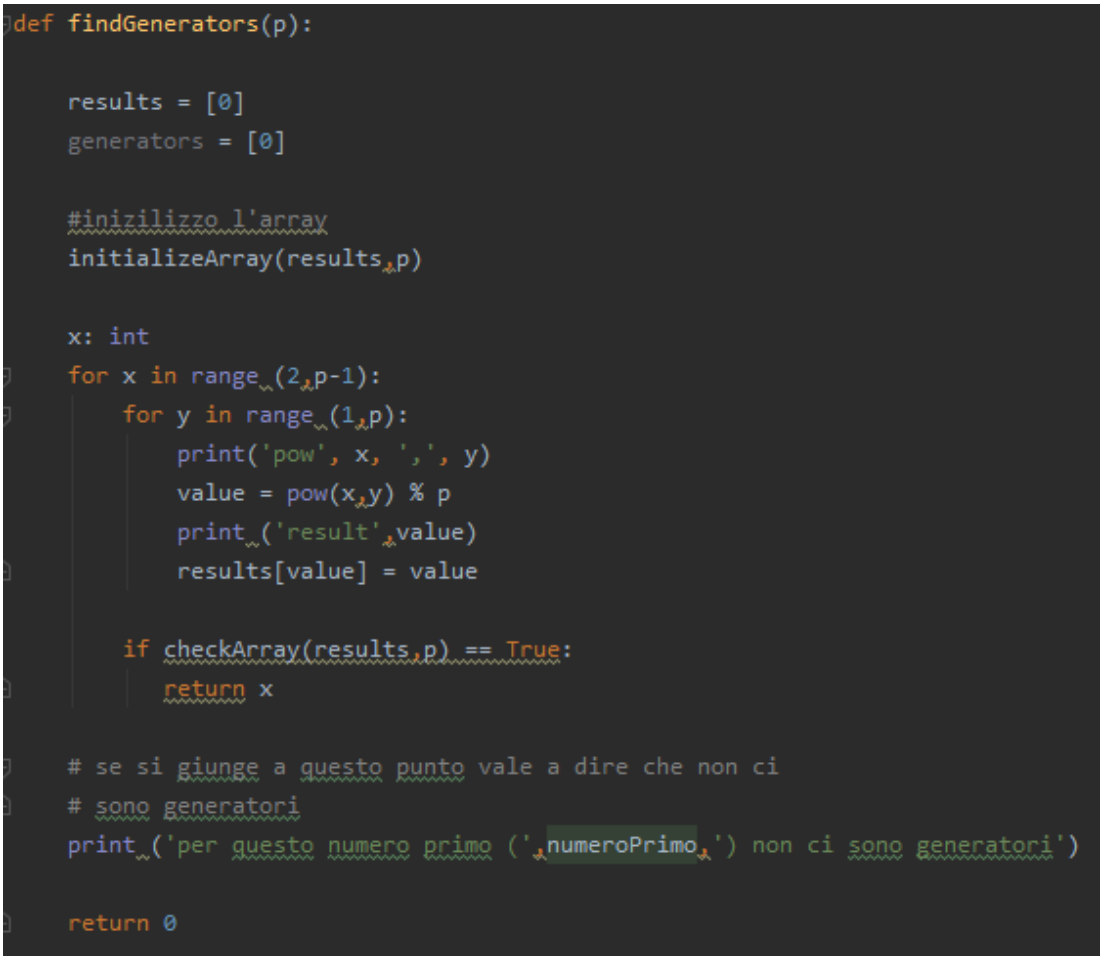
Gli elementi iniziali che l'algoritmo ElGamal prende in considerazione sono:

- il numero primo q;
- il generatore g;
- la chiave key.

I numeri primi devono essere necessariamente composti da almeno 8 bit, al fine di poter cifrare anche le stringhe.

Il generatore è stato trovato tramite il modulo "**FindGenerators.py**". Questa classe è costituita da:

- una funzione che inizializza un array con tutti gli elementi a zero;
- una funzione che controlla se all'interno dell'array è presente un elemento inizializzato a zero e, in quel caso, significa che il numero scelto non è un generatore;
- il metodo per la ricerca dei generatori:



```
def findGenerators(p):

    results = [0]
    generators = [0]

    #inizilizzo l'array
    initializeArray(results,p)

    x: int
    for x in range (2,p-1):
        for y in range (1,p):
            print('pow', x, ',', y)
            value = pow(x,y) % p
            print ('result',value)
            results[value] = value

        if checkArray(results,p) == True:
            return x

    # se si giunge a questo punto vale a dire che non ci
    # sono generatori
    print ('per questo numero primo (',numeroPrimo,') non ci sono generatori')

    return 0
```

Figura 3-1 : Metodo findGenerators per la ricerca dei generatori.

Key rappresenta la chiave privata, unica per tutti gli utenti. Per poter trovare il valore di questa chiave è stato utilizzato un metodo contenuto nella classe "**FindKValue.py**".

Dapprima è stato creato un metodo per valutare se due numeri sono relativamente primi fra loro, poi è stato sviluppato un metodo per la generazione di un k tale che gcd (k, numero primo scelto) sia primo e che quindi k e il numero primo scelto siano relativamente primi.

```
def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b;
    else:
        return gcd(b, a % b)
```

Figura 3-2: Metodo che calcola il massimo comun divisore tra due numeri.

```
def gen_key(q):
    key = random.randint(1, q)
    while gcd(q, key) != 1:
        key = random.randint(1, q)

    print("la key vale: ",key)
    return key
```

Figura 3-3: Metodo per la generazione della chiave key.

L'obiettivo di questa soluzione è quello di sfruttare la sicurezza e le performance dei cifrari asimmetrici, associandole alla versatilità di quelli asimmetrici. L'ABE viene realizzata attraverso l'uso del gammaValue.

Metodo di esponenziazione modulare

L'algoritmo di esponenziazione modulare permette di calcolare in modo efficiente la riduzione di una potenza modulo n.

```python
def power(a, b, c):
    x = 1
    y = a

    while b > 0:
        if b % 2 == 0:
            x = (x * y) % c
        y = (y * y) % c
        b = int(b / 2)

    return x % c
```

Figura 3-4: L'esponenziazione modulare.

Cifratura

Per quanto riguarda la cifratura, i parametri coinvolti sono:

- msg : plaintext da cifrare;
- q : numero primo scelto;
- h : betaValue, valore di Beta dato dal pow (generatore, alfa);
- g : generatore utilizzato.

```python
def encrypt(msg, h, k, q, g):
    en_msg = []

    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])

    print("g^k used : ", p)
    print("g^ak used : ", s)
    for i in range(0, len(en_msg)):
        en_msg[i] = (s * ord(en_msg[i]))

    print('messaggio cifrato: ', en_msg)

    return en_msg, p
```

Figura 3-5 : Il metodo per la cifratura.

La funzione di encryption ritorna en_msg, cioè l'array del messaggio criptato e p, il generatore elevato alla k mod q.

Decifratura

Nella decifratura i parametri sono:

- en_msg : ciphertext da decifrare;
- q : numero primo scelto;
- p : generatore elevato alla k mod q;
- key : k scelto.

```
def decrypt(en_msg, p, key, q):
    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int((en_msg[i]) / h)))

    return dr_msg
```

Figura 3-6: Il metodo per la decifratura.

La funzione di decryption ritorna dr_msg, l'array del messaggio decifrato.

Il main

La classe main permette all'utente di scegliere un numero primo e un messaggio da cifrare; va poi a ricercare il generatore e a generare la chiave privata tramite il metodo gen_key.

Applica poi l'encryption del messaggio in chiaro, prendendo come parametri in input:

- il betaValue: ottenuto come g^a, cioè il generatore elevato alla chiave privata;
- il numero primo q;
- il generatore g.

Infine, restituisce il messaggio decriptato attraverso la funzione di decryption.

```
def main():
    #scelta del numero primo da usare
    q = int(raw_input('inserisci il numero primo che vuoi usare (suggerimento: un numero primo grande è 3613): '))

    print ('il numero primo che hai scelto è {0}'.format(q))

    #trovo il generatore
    g = findGenerators(q)
     print ('il generatore trovato è: {0}'.format(g))

    msg = str(raw_input('inserisci il messaggio da cifrare: '))
    print("Messaggio originale :", msg)

    #genero la chiave privata tramite il metodo gen_key
    key = gen_key(q)

    betaValue = power(g, key, q)
    print('numero primo usato: ',q)
    print("generatore usato : ", g)
    print("g^a vale : ", betaValue)

    en_msg, p = encrypt(msg, betaValue,key, q, g)
    dr_msg = decrypt(en_msg, p, key, q)
    dmsg = ''.join(dr_msg)
    print("Messaggio decifrato :", dmsg);
```

Figura 3-7: La classe main.py

3.3 ElGamalABE.py

StringToNumbers() e generateGammaValue()

Il primo metodo è utile per trasformare le lettere in numeri: entra in gioco quando si produce il gammaValue.

```
def stringToNumbers(string):
    outputValue = []
    finalValue = 1
    for character in string:
        number = ord(character)
        outputValue.append(number)

    for index in range (1,len(outputValue)-1):
        finalValue = outputValue[index] * finalValue

    return finalValue
```

Figura 3-8: Metodo che trasforma le stringhe in numeri.

Il secondo, genera un valore gamma, utile a cifrare lo stesso messaggio in modo diverso in base agli attributi inseriti.

Questa funzione prende in input lo username dell'utente, la sua matricola, la chiave privata e il numero primo q, poi trasforma le stringhe username e la chiave in numeri, infine produce il valore gammaValue come prodotto tra i valori mod q.

```
def generateGammaValue(username, matricola, secretKey, q):
    intUser = stringToNumbers(username)
    secretKey = stringToNumbers(secretKey)
    gammaValue = (intUser * matricola * secretKey) % q

    print('gammaValue: ', gammaValue)
    return gammaValue
```

Figura 3-9: Metodo per la generazione del gammaValue.

Metodo di esponenziazione modulare e collectAttributes()

L'algoritmo di esponenziazione modulare permette di calcolare in modo efficiente la riduzione di una potenza modulo n.

```
def power(a, b, c):
    x = 1
    y = a

    while b > 0:
        if b % 2 == 0:
            x = (x * y) % c
        y = (y * y) % c
        b = int(b / 2)

    return x % c
```

Figura 3-10: Algoritmo di esponenziazione.

collectAttributes() è un metodo che restituisce lo username, la matricola e la chiave segreta inseirti dall'utente.

```
def collectAttributes():

    print ('inserisci i tuoi attributi per la cifratura.')
    username = str(raw_input('username: '))
    matricola = int(raw_input('matricola: '))
    secretKey = str(raw_input('secret key: '))

    return username, matricola, secretKey
```

Figura 3-11: Il metodo per la collezione degli attributi dell'utente.

Cifratura

```
def encrypt(msg, h, k, q, g, gammaValue):
    en_msg = []

    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])

    print("g^k used : ", p)
    print("g^ak used : ", s)
    for i in range(0, len(en_msg)):
        en_msg[i] = (s * ord(en_msg[i])) + gammaValue

    print('messaggio cifrato:', en_msg)

    return en_msg, p
```

Figura 3-12: La funzione di encryption.

La funzione prende in input il messaggio da criptare, il betaValue h, cioè valore di Beta dato dal pow (generatore, alfa), il numero primo q, il generatore g e il gammaValue precedentemente ottenuto.

Decifratura

```
def decrypt(en_msg, p, key, q, gammaValue):
    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int((en_msg[i] - gammaValue) / h)))

    print(dr_msg)

    return dr_msg
```

Figura 3-13: La funzione di decryption.

La funzione prende in input il messaggio cifrato, p cioè il generatore elevato alla k mod q, il numero primo q, la chiave key e il gammaValue.

Main

La classe main permette all'utente di scegliere un numero primo e un messaggio da cifrare; va poi a ricercare il generatore e a generare la chiave privata tramite il metodo gen_key.

Recupera gli attrubuti dell'utente attraverso il metodo collectAttributes() precedentemente descritto e crea il betaValue.

Applica poi l'encryption del messaggio in chiaro, prendendo come parametri in input:

- il betaValue: ottenuto come g^a, cioè il generatore elevato alla chiave privata
- il numero primo q
- il generatore g

Dopo di che, restituisce il messaggio decriptato attraverso la funzione di decryption.

Infine, si considerano gli attributi di un altro utente, usando lo stesso numero primo, lo stesso generatore, lo stesso messaggio in chiaro, la stessa chiave. Si calcola il gammavalue, poi procede con l'encryption e la decryption.

```
def main():
    global q, g, key

    #scelta del numero primo da usare
    q = int(raw_input('inserisci il numero primo che vuoi usare (suggerimento: un numero primo grande è 3613): '))

    print ('il numero primo che hai scelto è {0}'.format(q))
    #raccolta attributi
    username, matricola, secretKey = collectAttributes()
    gammaValue = generateGammaValue(username, matricola, secretKey, q)

    #trovo il generatore
    g = findGenerators(q)

    print ('il generatore trovato è: {0}'.format(g))

    msg = str(raw_input('inserisci il messaggio da cifrare: '))
    print("Messaggio originale :", msg)

    #genero la chiave privata tramite il metodo gen key
    key = gen_key(q)

    betaValue = power(g, key, q)
```

```
    en_msg, p = encrypt(msg, betaValue, key, q, g, gammaValue)
    dr_msg = decrypt(en_msg, p, key, q, gammaValue)
    dmsg = ''.join(dr_msg)
    print("il messaggio cifrato per questo utente è:", dmsg);

    print ("******************")

    print ("considero gli attributi di un altro utente, usando lo stesso plaintext {0}, lo stesso numero primo {1}, "
           "lo stesso generatore {2},"
           "la stessa chiave {3}".format(msg, q, g, key))
    usr, mat, secKey = collectAttributes()

    gammaV = generateGammaValue(usr, mat, secKey, q)

    en_msg2, p2 = encrypt(msg, betaValue, key, q, g, gammaV)
    dr_msg2 = decrypt(en_msg2, p2, key, q, gammaV)
    dmsg2 = ''.join(dr_msg2)
    print("il messaggio cifrato per questo utente è:", dmsg2);
```

Figura 3-14: La classe main.

BIBLIOGRAFIA

[1] Attribute Based Encryption with Privacy Preserving using Asymmetric Key in Cloud Computing - S Sankareswari, S.Hemanth.

[2] An Introduction to Attribute-Based Encryption – Zeutro LLC.

[3] Attribute-Based Encryption with verifiable outsourced decryption - Abha Pandit, Aishwarya Lamture, Pooja Sankpal, Shubham Dixit, Tabassum Maktum.

[4] https://www.zeutro.com/technology.html

[5] https://medium.com/asecuritysite-when-bob-met-alice/towards-true-security-attribute-based-encryption-20d5799aeda6