SAPIENZA
UNIVERSITÀ DI ROMA

Seminario Matematica Discreta

Facoltà di Ingegneria dell'informazione, Informatica e Statistica
Corso di Laurea in Informatica

Candidate
Leonardo Danella
ID number 1885686

Thesis Advisor
Prof. Gullà

Academic Year 2021/2022

Thesis defended on 24 January 2022
in front of a Board of Examiners composed by:

Prof. Gabriele Gullà (chairman)

Seminario Matematica Discreta
Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: danella.1885686@studenti.uniroma1.it

Abstract

One of the most singular characteristics of the art of deciphering is the strong conviction possessed by every person, even moderately acquainted with it, that he is able to construct a cipher which nobody else can decipher. I have also observed that the cleverer the person, the more intimate is his conviction
- Charles Babbage

Una delle caratteristiche più singolari dell'arte della decifrazione è la forte convinzione posseduta da ogni persona, anche moderatamente esperta, di essere in grado di costruire un codice che nessun altro può decifrare. Ho anche osservato che più intelligente è la persona, più intima è la sua convinzione
- Charles Babbage

Contents

Chapter 1

Introduzione

La crittografia è lo studio e la pratica delle tecniche per la comunicazione sicura in presenza di terzi chiamati avversari. Si occupa di sviluppare e analizzare protocolli che impediscono a terzi malintenzionati di recuperare le informazioni condivise tra due entità, seguendo così i vari aspetti della sicurezza delle informazioni.

La comunicazione sicura si riferisce allo scenario in cui il messaggio o i dati condivisi tra due parti non possono essere accessibili da un avversario. In crittografia, un avversario è un'entità maligna che mira a recuperare informazioni o dati preziosi, minando così i principi della sicurezza delle informazioni.

I principi fondamentali della crittografia moderna sono:

- Riservatezza dei dati (Confidentiality),
- Integrità dei dati (Integrity),
- Autenticazione (Authentication)
- Non ripudio (Non-repudiation)

La riservatezza si riferisce a certe regole e linee guida di solito eseguite sotto accordi di riservatezza che assicurano che le informazioni siano limitate a certe persone o luoghi.

L'integrità dei dati si riferisce al mantenimento e all'assicurazione che i dati rimangano accurati e coerenti durante il loro intero ciclo di vita.

L'autenticazione è il processo di assicurarsi che il pezzo di dati rivendicato dall'utente gli appartenga.

Il non ripudio si riferisce alla capacità di assicurarsi che una persona o una parte associata a un contratto o a una comunicazione non possa negare l'autenticità della sua firma su un documento o l'invio di un messaggio.

Quando stiamo criptando un messaggio, il plaintext (testo in chiaro) si riferisce al messaggio non criptato e il ciphertext (testo cifrato) al messaggio cifrato. Un cifrario è quindi composto da due funzioni: la crittazione trasforma un testo in chiaro in un testo cifrato, e la decrittazione trasforma un testo cifrato in un testo in chiaro.
N.B.: Per alcuni cifrari, il testo cifrato ha la stessa dimensione del testo in chiaro; per altri, è leggermente più lungo. Tuttavia, non possono mai essere più corti del testo in chiaro.

Chapter 2

Crittazione a chiave simmetrica

2.1 Introduzione

Gli algoritmi di crittazione a chiave simmetrica possiamo suddividerli principalmente in due gruppi:
Cifrari monoalfabetici, come nel cifrario di Cesare, nel quale lavoriamo con una funzione $f : \mathbb{Z}_{26} \to \mathbb{Z}_{26}$ che mappa $p \to p + 3 \mod 26$
Cifrari polialfabetici, come ad esempio il cifrario di Vigénère, sviluppato nel 15° secolo. Questo cifrario consiste in una generalizzazione del cifrario di Cesare, che invece di spostare sempre dello stesso numero di posti la lettera da cifrare, questa viene spostata di un numero di posti variabile ma ripetuto, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario, e da scrivere ripetutamente sotto il messaggio, carattere per carattere; la chiave era detta anche verme, per il motivo che, essendo in genere molto più corta del messaggio, deve essere ripetuta molte volte sotto questo, come nel seguente esempio:

Testo in chiaro - ESAMEMATEMATICADISCRETA

Verme - VERMEVERMEVERMEVERMEVER

Testo cifrato - ZWRYIHEKQQVXZOEYMJOVZXR

Vantaggi e svantaggi crittografia simmetrica:

Vantaggi:

1. Facile da capire
2. Facile da computare
3. è chiaro capire se l'algoritmo è rotto (broken) o meno

Svantaggi:

1. bisogno di avere una chiave comune da concordare
2. ogni lettera viene mappata in una e una sola lettera, quindi è molto vulnerabile all'analisi di frequenza
3. non hai la prova che il messaggio sia stato effettivamente inviato dalla sorgente, e non modificato da una persona nel mezzo

2.2 Block ciphers

si occupa di cifrari che elaborano messaggi blocco per blocco, ci concentreremo su quello più famoso, l'*Advanced Encryption Standard* (AES).
Un cifratore a blocchi prende un blocco di bit di testo in chiaro e genera un blocco di bit di testo cifrato, generalmente della stessa dimensione. La dimensione del blocco è fissa nello schema dato. La scelta della dimensione del blocco non influisce direttamente sulla forza dello schema di cifratura. La forza del cifrario dipende dalla lunghezza della chiave.
Tra i block ciphers più famosi non possiamo non citare il *DES*, Digital Encryption Standard - un popolare block cipher degli anni '90. Ora è considerato un cifratore a blocchi "rotto", principalmente a causa della piccola dimensione della sua chiave, e della sua variante, il *Triple DES* basato su applicazioni DES ripetute. È ancora un cifratore a blocchi rispettato, ma inefficiente rispetto ai nuovi cifrari a blocchi più veloci disponibili.

2.2.1 AES

Sistema di crittaggio più utilizzato sul web. I blocchi di bit variano tra 128|192|256, e in base alla grandezza varia il numero di round di crittazione da svolgere, in particolare sono rispettivamente 10|12|14 rounds. Un "round" consiste in quattro fasi:

1. **Confusion**: *sostituire i bytes in base alla chiave*:
   Nel passo Confusion, ogni byte $a_{i,j}$ nella matrice di stato viene sostituito con un byte di sostituzione (SubByte) $S(a_{i,j})$ usando una scatola di sostituzione (S-box) a 8 bit. Si noti che prima del round 0, la matrice di stato è semplicemente il testo in chiaro/input. Questa operazione fornisce la non-linearità nel cifrario. La S-box usata è derivata dall'inverso moltiplicativo su $GF(2^8)$[1], nota per avere buone proprietà di non linearità. Per evitare attacchi basati su semplici proprietà algebriche, la S-box è costruita combinando la funzione inversa con una trasformazione affine invertibile. La S-box è anche scelta per evitare qualsiasi punto fisso (e quindi è un derangement), cioè, $S(a_{i,j}) \neq a_{i,j}$ e anche tutti i punti fissi opposti, cioè, $S(a_{i,j}) \oplus a_{i,j} \neq \mathrm{FF}_{16}$. Durante l'esecuzione della decrittazione, viene utilizzato il passo InvSubBytes (l'inverso di SubBytes), che richiede prima di prendere l'inverso della trasformata affine e poi trovare l'inverso moltiplicativo.

2. **Permutation**: *Row Shifting*
   Il passo di shifting delle righe opera sulle righe dello stato; sposta ciclicamente i byte in ogni riga di un certo offset. Nell'AES, la prima riga viene lasciata invariata. Ogni byte della seconda fila è spostato di uno a sinistra. Allo stesso modo, la terza e la quarta riga sono spostate rispettivamente con un offset di due e tre. In questo modo, ogni colonna dello stato di uscita del passo di Row Shifting è composta da byte di ogni colonna dello stato di ingresso. L'importanza di questo passo è di evitare che le colonne siano crittate indipendentemente, nel qual caso AES degenererebbe in quattro cifrari a blocchi indipendenti.

3. **Diffusion**: *Column mixing*
   Nel passo di Column mixing, i quattro byte di ogni colonna dello stato sono combinati usando una trasformazione lineare invertibile. La funzione MixColumns prende quattro byte come input e produce quattro byte, ed ogni byte

[1] a *Galois field* (or *finite field*) is a field that contains a finite number of elements.

di input influenza tutti e quattro i byte di output. Insieme a Row shifting, fornisce la diffusione nel cifrario.
Durante questa operazione, ogni colonna è trasformata usando una matrice fissa (la matrice moltiplicata a sinistra per la colonna dà il nuovo valore della colonna nello stato):

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \qquad 0 \leq j \leq 3$$

La moltiplicazione della matrice è composta dalla moltiplicazione e dall'addizione delle voci. Le voci sono byte trattati come coefficienti del polinomio di ordine x^7. L'addizione è semplicemente uno XOR. La moltiplicazione è modulo del polinomio irriducibile $x^8 + x^4 + x^3 + x + 1$. Se elaborato bit per bit, allora, dopo lo spostamento, deve essere eseguito uno XOR condizionato con $1B_{16}$ se il valore spostato è maggiore di FF_{16} (l'overflow deve essere corretto dalla sottrazione del polinomio generatore). Questi sono casi speciali della solita moltiplicazione in GF(2^8).

In senso più generale, ogni colonna è trattata come un polinomio su GF(2^8) e viene poi moltiplicato modulo $01_{16} \cdot z^4 + 01_{16}$ con un polinomio fisso $c(z) = 03_{16} \cdot z^3 + 01_{16} \cdot z^2 + 01_{16} \cdot z + 02_{16}$. I coefficienti sono visualizzati nel loro equivalente esadecimale della rappresentazione binaria dei polinomi bit da GF(2)[x].

4. **Key Addition**: *aggiungi una subkey e rinizi da 1* Nel passo di Key Addition, la sottochiave è combinata con lo stato. Per ogni round, una sottochiave è derivata dalla chiave principale usando il programma di chiavi di Rijndael[2]; ogni sottochiave è della stessa dimensione dello stato. La sottochiave viene aggiunta combinando ogni byte dello stato con il byte corrispondente della sottochiave usando il bitwise XOR.

Tentare di avere un approccio di brute-force anche solo su una crittazione a 128 bit richiede circa 10^{18} anni, per quella a 192 invece circa 10^{37} anni, mentre per l'ultima 10^{56} anni

2.3 Stream ciphers

presenta cifrari che producono un flusso di bit dall'aspetto casuale al quale si applica lo XOR con i messaggi da crittografare.

[2]Prende il nome dai due crittografi Vincent Rijmen and Joan Daemen, che l'hanno proposta durante l'AES Selection Process

Chapter 3

Crittazione a chiave asimmetrica

3.1 Problemi difficili

Un problema computazionale è un qualcosa a cui si può rispondere facendo sufficienti calcoli, per esempio, "Il 20220119 è un numero primo?" o "Quante lettere m ci sono matematicamente?". La durezza (hardness) computazionale è la proprietà dei problemi computazionali per i quali non esiste un algoritmo che possa essere eseguito in un tempo ragionevole. Tali problemi sono anche chiamati problemi intrattabili e sono spesso praticamente impossibili da risolvere. Ovviamente, la durezza computazionale deve essere indipendente dal tipo di dispositivo di calcolo utilizzato, sia esso una CPU per uso generale, un circuito integrato o una macchina di Turing meccanica. Infatti, una delle prime scoperte della teoria della complessità computazionale è che tutti i modelli di calcolo sono equivalenti. Se un problema può essere risolto in modo efficiente con un dato dispositivo, può essere risolto in modo efficiente su qualsiasi altro dispositivo portando l'algoritmo al linguaggio dell'altro dispositivo - eccezion fatta per i computer quantistici. Il risultato è che non avremo bisogno di specificare il dispositivo di calcolo o l'hardware sottostante quando si discute di durezza computazionale; invece, ci limiteremo a discutere gli algoritmi.
Per valutare la durezza, dobbiamo prima trovare un modo per misurare la complessità di un algoritmo, o il suo tempo di esecuzione. Poi classificheremo i tempi di esecuzione in difficili o facili.
La domanda più importante ancora aperta nell'ambito della complessità computazionale è sicuramente quella sorta dal noto problema "$P = NP$?", formulato negli anni '70 da Stephen Cook dell'Università di Toronto. Informalmente tale problema chiede di dimostrare o refutare se computazioni deterministiche efficienti siano sufficienti per catturare computazioni non-deterministiche efficienti. In tale contesto con computazioni efficienti si intendono algoritmi il cui tempo di esecuzione sia limitato polinomialmente. Ad esempio, supponiamo che si stiano organizzando gli alloggi per un gruppo di quattrocento studenti universitari. Lo spazio è limitato e solo cento degli studenti riceveranno posti nel dormitorio. Per complicare le cose, il rettore ha fornito una lista di coppie di studenti incompatibili e ha richiesto che nessuna coppia di quest'ultima appaia nella scelta finale. Questo è un esempio di ciò che viene chiamato un problema NP, poiché è facile controllare se una data scelta di cento studenti proposta da un collega sia soddisfacente (cioè che nessuna coppia presa dalla lista del tuo collega appaia anche nella lista dell'ufficio del Rettore), tuttavia il compito di generare una tale lista da zero sembra essere così difficile da essere completamente impraticabile. Infatti, il numero totale di modi di scegliere cento studenti tra i quattrocento candidati $\binom{400}{100}$ è maggiore del numero di atomi

nell'universo conosciuto! Quindi nessuna civiltà futura potrebbe mai sperare di costruire un supercomputer capace di risolvere il problema utilizzando la forza bruta, cioè controllando ogni possibile combinazione. Tuttavia, questa apparente difficoltà potrebbe solo riflettere la mancanza di ingegno del programmatore. Infatti, uno dei problemi più importanti dell'informatica è determinare se esistono domande la cui risposta può essere controllata rapidamente, ma che richiedono un tempo impossibilmente lungo per essere risolte con qualsiasi procedura diretta. Problemi come quello elencato sopra sembrano certamente essere di questo tipo, ma finora nessuno è riuscito a dimostrare che qualcuno di essi sia davvero così difficile come sembra, cioè che non esista davvero un modo fattibile per generare una risposta con l'aiuto di un computer. Stephen Cook e Leonid Levin hanno formulato il problema P (cioè facile da trovare) contro NP (cioè facile da controllare) indipendentemente nel 1971. Più formalmente possiamo definire le due classi P ed NP come segue:
P è l'insieme dei linguaggi che sono decidibili in tempo polinomiale su una macchina di Turing deterministica a nastro singolo. Cioè:

$$P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$$

NP è l'insieme dei linguaggi che sono decidibili in tempo polinomiale su una macchina di Turing non deterministica. Cioè:

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

Se si potessero risolvere i problemi NP più difficili in tempo polinomiale, allora si potrebbero risolvere tutti i problemi NP in tempo polinomiale, e quindi NP sarebbe uguale a P. Suona assurdo; non è ovvio che ci sono problemi per i quali la soluzione è facile da verificare ma difficile da trovare? Per esempio, non è ovvio che la forza bruta in tempo esponenziale è il modo più veloce per recuperare la chiave di un cifrario simmetrico, e quindi che il problema non può essere in P? Si scopre che, per quanto possa sembrare folle, nessuno ha dimostrato che P è diverso da NP, nonostante una ricompensa letteralmente di un milione di dollari.
Il Clay Mathematics Institute, infatti, assegnerà questa taglia a chiunque dimostri che $P \neq NP$ o che $P = NP$. Questo problema, noto come P vs NP, è stato definito "una delle domande più profonde che gli esseri umani abbiano mai chiesto" dal famoso teorico della complessità Scott Aaronson. Pensateci: se P fosse uguale a NP, allora ogni soluzione facilmente verificabile sarebbe anche facile da trovare. Tutta la crittografia usata in pratica sarebbe insicura, perché si potrebbero recuperare le chiavi simmetriche e invertire le funzioni hash in modo efficiente.

3.2 RSA

Il sistema di crittografia Rivest-Shamir-Adleman (RSA) ha rivoluzionato crittografia quando è emerso nel 1977 come il primo schema di crittografia a chiave pubblica. Mentre i classici schemi di crittografia a chiave simmetrica usano la stessa stessa chiave segreta per criptare e decriptare i messaggi, la crittografia a chiave pubblica (chiamata anche crittografia asimmetrica) utilizza due chiavi: una è la chiave pubblica, che può essere usata da chiunque voglia criptare messaggi per te, e l'altra è la tua chiave privata, che è necessaria per decifrare i messaggi crittografati con la chiave pubblica. Questa magia è la ragione per cui RSA è stato un vero passo avanti, e 40 anni dopo, è ancora il modello della crittografia a chiave pubblica e un cavallo di battaglia della sicurezza su internet. (Un anno prima di RSA, Diffie e Hellman avevano introdotto il concetto di crittografia a chiave pubblica, ma il loro schema non era in grado di eseguire la crittografia a chiave pubblica). RSA è soprattutto un trucco aritmetico. Funziona creando un oggetto matematico chiamato permutazione trapdoor, una funzione che trasforma un numero x in un numero y nello stesso intervallo, in modo tale che calcolare y da x è facile usando la chiave pubblica, ma calcolare x da y è praticamente impossibile a meno che non si conosca la chiave privata, la trapdoor. (Si può pensare a x come un testo in chiaro e y come un testo cifrato).
Oltre alla crittografia, l'RSA è anche usato per costruire firme digitali, dove il proprietario della chiave privata è l'unico in grado di firmare un messaggio, e la chiave pubblica permette a chiunque di verificare la validità della firma.

3.2.1 Matematica dietro l'RSA

Quando si cripta un messaggio, RSA vede il messaggio come un grande numero, e la crittografia consiste essenzialmente in moltiplicazioni di grandi numeri. Quindi, per capire come funziona RSA, dobbiamo sapere che tipo di grandi numeri manipola e come funziona la moltiplicazione su quei numeri. RSA vede il testo in chiaro che sta criptando come un numero intero positivo tra 1 e n-1, dove n è un valore appositamente enorme che, in $\mathbb{Z}_n$ (il gruppo dei co-primi con n), chiamiamo modulo. Per calcolare la cardinalità di $\mathbb{Z}_n$, come ben sappiamo, basta usare la funzione di Eulero ($\phi(n)$), che dato un $n = p_1 \times p_2 \times ... \times p_m$ calcola:

$$\phi(n) = (p_1 - 1) \times (p_2 - 1) \times ... \times (p_m - 1)$$

RSA si occupa solo di numeri n che sono il prodotto di due grandi primi, di solito indicato come $n = p * q$. Il gruppo associato $\mathbb{Z}_n$ conterrà quindi $\phi(n) = (p-1)(q-1)$ elementi. Espandendo questa espressione, otteniamo la definizione equivalente $\phi(n) = n - p - q + 1$, o $\phi(n) = (n + 1) - (p + q)$, che esprime più intuitivamente il valore di $\phi(n)$ rispetto a n. In altre parole, tutti i numeri tra 1 e n - 1, tranne (p + q), appartengono a $\mathbb{Z}_N$ e sono "numeri validi" nelle operazioni RSA.
La permutazione trapdoor dell'RSA è l'algoritmo centrale dietro la crittografia e le firme digitali basate su RSA.

3.2.2 La permutazione trapdoor dell'RSA

Dato un modulo n ed un numero e, chiamato l'esponente pubblico, la permutazione trapdoor trasforma un numero x dall'insieme $\mathbb{Z}_n$ in un numero $y = x^e \mod n$. In altre parole, calcola il valore che è uguale a x moltiplicato per se stesso e volte modulo n e poi restituisce il risultato. Quando usiamo la permutazione trapdoor per criptare, il modulo n e l'esponente e costituiscono la chiave pubblica

RSA. Per recuperare x da y, usiamo un altro numero, indicato con d, per calcolare quanto segue:

$$y^d \mod n = (x^e)^d \mod n = x^{ed} \mod n = x$$

Poiché d è la trapdoor che ci permette di decifrare, fa parte della chiave privata in una coppia di chiavi RSA e, a differenza della chiave pubblica, dovrebbe essere sempre tenuta segreta. Il numero d è anche chiamato, infatti, esponente segreto.
Ovviamente, d non è un numero qualsiasi; è il numero tale che e moltiplicato per d è equivalente a 1, e quindi tale che $x^{ed} \mod n = x$ per qualsiasi x. Più precisamente, dobbiamo avere $ed = 1 \mod \phi(n)$ per ottenere $x^{ed} = x^1 = x$ e per decifrare correttamente il messaggio. Si noti che calcoliamo modulo $\phi(n)$ e non modulo n qui perché gli esponenti si comportano come gli indici di elementi di $\mathbb{Z}_n$ piuttosto che come gli elementi stessi. Poiché $\mathbb{Z}_n$ ha $\phi(n)$ elementi, l'indice deve essere inferiore a $\phi(n)$.
Il numero $\phi(n)$ è cruciale per la sicurezza di RSA. Infatti, trovare $\phi(n)$ per un modulo n di RSA è equivalente a rompere RSA, perché l'esponente segreto d può essere facilmente derivato da $\phi(n)$ ed e, calcolando l'inverso di e. Quindi anche p e q dovrebbero essere segreti, poiché conoscere p o q dà $\phi(n)$ calcolando $(p-1)(q-1) = \phi(n)$.

3.2.3 Generazione e sicurezza delle chiavi nell'RSA

La generazione della chiave è il processo con cui viene creata una coppia di chiavi RSA, cioè una chiave pubblica (modulo n ed esponente pubblico e) e la sua chiave privata (esponente segreto d). Anche i numeri p e q (tali che n = pq) e l'ordine $\phi(n)$ devono essere segreti, quindi sono spesso visti come parte della chiave privata.
Per generare una coppia di chiavi RSA, prima scegliamo due numeri primi a caso, p e q, e poi calcoliamo $\phi(n)$ da questi, e calcoliamo d come l'inverso di e.
Per fare ciò possiamo sfruttare una libreria molto potente in python, chiamata sage e usarla nel seguente modo:

```
sage: p = random_prime(2^32); p
    1103222539
sage: q = random_prime(2^32); q
    17870599
sage:n = p*q; n
    c
sage: phi = (p-1)*(q-1); phi
    36567230045260644
sage: e = random_prime(phi); e
    13771927877214701
sage: d = xgcd(e, phi)[1]; d
    15417970063428857
sage: mod(d*e, phi)
    1
```

Qui ovviamente, per evitare inutili sprechi di spazio ho usato un modulo n a 64 bit, mentre nella realtà ne andrebbe usato uno di almeno 2048.

Usiamo la funzione random_prime() per scegliere due primi casuali p e q, che sono più piccoli del dato argomento. Poi, moltiplichiamo p e q per ottenere il modulo n e $\phi(n)$. Generiamo poi un esponente pubblico casuale, e, scegliendo un primo

casuale inferiore a ϕ per assicurarci che e abbia un modulo inverso a ϕ. Generiamo poi l'esponente privato associato d usando la funzione xgcd(). Questa funzione calcola i numeri s e t dati due numeri, a e b, con l'algoritmo euclideo esteso tale che $as + bt = GCD(a, b)$. Infine, controlliamo che $ed \mod \phi(n) = 1$, per assicurarci che d funzionerà correttamente per invertire la permutazione RSA.
Ora possiamo applicare la permutazione trapdoor:

```
sage: x = 1234567
sage: y = power_mod(x, e, n); y
    19048323055755904
sage: power_mod(y, d, n)
    1234567
```

Assegniamo l'intero 1234567 a x e poi usiamo la funzione power_mod(x, e, n) , l'esponenziazione modulo n, o $x^e \mod n$ in forma di equazione per calcolare y. Avendo calcolato $y = x^e \mod n$, calcoliamo $y^d \mod n$ con la trapdoor d per restituire la x originale. Ma quanto è difficile trovare x senza la trapdoor d? Un attaccante che può fattorizzare grandi numeri può rompere l'RSA recuperando p e q e poi $\phi(n)$ per calcolare d da e. Ma questo non è l'unico rischio. Un altro rischio per l'RSA sta nella capacità di un attaccante di calcolare x da $x^e \mod n$, o l'e-esima radice modulo n, senza necessariamente fattorizzare n. Entrambi i rischi sembrano strettamente connessi, anche se non sappiamo con certezza se sono equivalenti.
Assumendo che la fattorizzazione sia effettivamente difficile e che trovare l'e-esima radice sia altrettanto difficile, il livello di sicurezza di RSA dipende principalmente da due fattori:

- la dimensione di n,
- la scelta di p e q

Se n è troppo piccolo, potrebbe essere fattorizzato in un tempo realistico, rivelando la chiave privata. Per essere sicuri, n dovrebbe essere lungo almeno 2048 bit (un livello di sicurezza di circa 90 bit, che richiede uno sforzo computazionale di circa 2^{90} operazioni), ma preferibilmente 4096 bit (un livello di sicurezza di circa 128 bit). I valori p e q dovrebbero essere numeri primi casuali non correlati numeri primi casuali non correlati di dimensioni simili. Se sono troppo piccoli, o troppo vicini, diventa più facile determinare il loro valore da n.

3.2.4 Crittare con l'RSA

Tipicamente, RSA è usato in combinazione con uno schema di crittografia simmetrica, con l'RSA che viene usato per criptare una chiave simmetrica che viene poi usata per crittografare un messaggio con un cifrario come l'Advanced Encryption Standard (AES). Ma criptare un messaggio o una chiave simmetrica con RSA è più complicato che convertire semplicemente l'obiettivo in un numero $x^e \mod n$

3.2.5 Malleabilità della crittografia RSA da manuale

La crittografia RSA da manuale è la frase usata per descrivere il semplicistico schema di crittografia RSA in cui il testo in chiaro contiene solo il messaggio che si vuole crittografare.
Per esempio, per criptare la stringa RSA, dovremmo prima convertirla in un numero concatenando le codifiche ASCII di ogni tre lettere come byte: R (byte 52), S (byte 53) e A (byte 41). La stringa di byte risultante 525341 è uguale a 5395265 una

volta convertita in decimale, che potremmo poi criptare calcolando $5395265^e \mod n$. Senza conoscere la chiave segreta, non ci sarebbe modo di decifrare il messaggio. Tuttavia, la crittografia RSA da manuale è deterministica: se si cripta lo stesso testo in chiaro due volte, si otterrà lo stesso testo cifrato due volte. Questo è un problema, ma c'è un problema più grande: dati due testi RSA cifrati $y_1 = x_1^e \mod n$ e $y_2 = x_2^e \mod n$, si può ricavare il testo cifrato di $x_1 \times x_2$ moltiplicando questi due testi cifrati insieme, in questo modo:

$$y_1 \times y_2 \mod n = x_1^e \times x_2^e \mod n = (x_1 \times x_2)^e \mod n$$

Il risultato è $(x_1 \times x_2)^e \mod n$, e il testo cifrato del messaggio è $x_1 \times x_2 \mod n$. Così un attaccante potrebbe creare un nuovo cifrario valido da due cifrari RSA permettendo di compromettere la sicurezza della vostra crittografia permettendo loro di dedurre informazioni sul messaggio originale. Noi diciamo che questa debolezza rende la crittografia RSA da manuale malleabile. (Naturalmente, se si conoscono x_1 *e* x_2 , si può calcolare $(x_1 \times x_2)^e \mod n$, ma se si conoscono solo y_1 e y_2, non si dovrebbe essere in grado di moltiplicare i testi cifrati e ottenere un testo cifrato dei testi in chiaro moltiplicati).

Per ovviare a questo grave problema, il testo cifrato dovrebbe consistere nel messaggio, e in più vengono aggiunti dei dati con un'operazione chiamata padding. Questo metodo viene applicato nella crittazione dell'RSA nel cosiddetto RSA-OAEP (Optimal Asymmetric Encryption Padding): questo schema comporta la creazione di una stringa di bit grande quanto il modulo, riempendo il messaggio con dati extra e randomness prima di applicare la funzione RSA.

3.3 Diffie-Hellman

Estende la crittografia asimmetrica alla nozione di accordo di una chiave, in cui due parti stabiliscono un valore segreto usando solo valori non segreti:
Primo algoritmo sviluppato funzionante a chiave asimmetrica.
Questo metodo di crittazione prevede:

- un gruppo ciclico, cioè un gruppo che può essere generato da un unico elemento. Tale gruppo è isomorfo a $\mathbb{Z}/n\mathbb{Z}$ delle classi resto modulo n, o al gruppo $\mathbb{Z}$ degli interi: $G = g^n : n \in \mathbb{Z}$;
- un punto che definiamo generatore, che chiameremo α, che sarà pubblico;
- un numero primo largo, che useremo come modulo e che sarà anch'esso pubblico, che chiameremo p.

Ora Alice e Bob decidono due chiavi private, che saranno gli esponenti nella funzione:
$\alpha^a \equiv A \mod p$ per Alice, dove a è la chiave privata, ed A è quella pubblica
$\alpha^b \equiv B \mod p$ per Bob, dove b è la chiave privata, e B è quella pubblica

3.3.1 Diffie-Hellman Key Exchange

Alice: $B + a = K$, dove K è la chiave di sessione che va mantenuta privata!

Vogliamo dimostrare che Bob può calcolare K nel seguente modo: $A + b = K$

Quindi Alice calcola $B^a = (\alpha^b)^a = \alpha^{b*a}$
Bob calcola invece $A^b = (\alpha^a)^b = \alpha^{a*b}$
I risultati, essendo in un gruppo ciclico, e quindi commutativo, sono esattamente la stessa cosa.
Quindi A+b calcolato da Bob è proprio uguale a K.

esempio di funzionamento a 2 parti:
Alice e Bob concordano pubblicamente di usare un modulo p = 23 e una base g = 5 (che è una radice primitiva modulo 23). Alice sceglie un intero segreto a = 4, poi invia a Bob $A \equiv g^a \mod p$
$A = 5^4 \mod 23 = 4$
Bob sceglie un intero segreto b = 3, poi manda ad Alice $B \equiv g^b \mod p$
$B = 5^3 \mod 23 = 10$
Alice calcola $s \equiv B^a \mod p$
$s \equiv 10^4 \mod 23 = 18$
Bob calcola $s \equiv A^b \mod p$
$s \equiv 4^3 \mod 23 = 18$
Alice e Bob ora condividono un segreto (il numero 18).

Solo a e b sono tenuti segreti. Tutti gli altri valori, cioè p, g, $g^a \mod p$, e $g^b \mod p$, sono inviati in chiaro. La forza dello schema deriva dal fatto che $(g^a)^b \mod p = (g^b)^a \mod p$ richiedono tempi estremamente lunghi per essere calcolati da qualsiasi algoritmo conosciuto sfruttando solo la conoscenza di p, g, $g^a \mod p$, e $g^b \mod p$. Una volta che Alice e Bob calcolano il segreto condiviso possono usarlo come chiave di crittografia, nota solo a loro, per inviare messaggi attraverso lo stesso canale di comunicazione aperto.

Naturalmente, sarebbero necessari valori molto più grandi di a, b e p per rendere sicuro questo esempio, poiché ci sono solo 23 possibili risultati di $n \mod 23$. Tuttavia, se p è un primo di almeno 600 cifre, allora anche i più veloci computer moderni che usano il più veloce algoritmo conosciuto non possono trovare un dato utilizzando solo g, p e $g^a \mod p$. Tale problema è chiamato il problema del logaritmo discreto. Il calcolo di $g^a \mod p$ è noto come esponenziazione modulare e può essere svolto in maniera efficiente anche per grandi numeri. Si noti che g non deve essere necessariamente grande, e in pratica è di solito un piccolo numero intero (come 2, 3, ...).

esempio di funzionamento a 3 parti invece che 2:

Le parti si accordano sui parametri dell'algoritmo p e g.
Le parti generano le loro chiavi private, chiamate a, b e c.
Alice calcola g^a e la invia a Bob.
Bob calcola $(g^a)^b = g^{a*b}$ e lo invia a Carol.
Carol calcola $(g^{a*b})^c = g^{a*b*c}$ e lo usa come suo segreto.
Bob calcola g^b e lo invia a Carol.
Carol calcola $(g^b)^c = g^{b*c}$ e lo invia ad Alice.
Alice calcola $(g^{b*c})^a = g^{b*c*a} = g^{a*b*c}$ e lo usa come segreto.
Carol calcola g^c e lo invia ad Alice.
Alice calcola $(g^c)^a = g^{c*a}$ e lo invia a Bob.
Bob calcola $(g^{c*a})^b = g^{c*a*b} = g^{a*b*c}$ e lo usa come suo segreto.

Un intercettatore è stato in grado di vedere $g^a, g^b, g^c, g^{a*b}, g^{a*c}$, *e* g^{b*c}, ma non può usare nessuna combinazione di queste per riprodurre efficientemente g^{a*b*c}.
Più nello specifico: $(g^a(\mod p))^b(\mod p) = (g^b(\mod p))^a(\mod p)$

3.3.2 Problema del logaritmo discreto

La complessità computazionale nel risolvere questo tipo di crittaggio, è data dal cosiddetto *problema del logaritmo discreto (Discrete Logarithm Problem).*
I logaritmi discreti sono il corrispettivo dei logaritmi ordinari per l'aritmetica modulare: in generale, sia G un gruppo ciclico finito di n elementi (supponiamo una scrittura moltiplicativa). Sia b un generatore di G; allora ogni elemento $g \in G$ può essere scritto nella forma $g = b^k$ per un qualche intero k. Inoltre, se $b^k = g = b^h$ per due interi h e k, allora $h \equiv k \mod n$. Possiamo quindi definire una funzione:

$$\log_b : G \to \mathbb{Z}_n$$

dove Z_n indica l'anello degli interi modulo n, assegnando ad ogni $g \in G$ la classe di congruenza k modulo n. Questa funzione è un isomorfismo tra gruppi, detto logaritmo discreto in base b; b è detta anche radice primitiva.

La formula per il cambio di base dei logaritmi ordinari rimane valida: se c è un altro generatore di G, si ha che:

$$\log_c g = \log_c b * \log_b g$$

Il calcolo dei logaritmi discreti sembra un problema duro (hard) (non sono noti algoritmi efficienti), mentre il problema inverso dell'esponenziazione discreta non lo è. Quest'asimmetria è analoga a quella fra la fattorizzazione di interi e la moltiplicazione di interi. Entrambe queste asimmetrie sono state utilizzate per la costruzione di sistemi crittografici.

Nella crittografia basata sui logaritmi discreti scelte comuni per il gruppo G sono i gruppi ciclici $\mathbb{Z}_p$

Se abbiamo un punto generatore pubblico e lo eleviamo per un esponente privato, ciò che otteniamo è una chiave pubblica modulo p, cioè:

$$\alpha^a \equiv A \mod p$$

è molto facile andare da sx verso dx, ma è computazionalmente infattibile il viceversa.

Il protocollo è quindi sicuro contro la pratica di sniffing, con cui si definisce l'attività di intercettazione passiva dei dati che transitano in una rete telematica, ma è vulnerabile contro un tipo di attacco definito man in the middle, che consiste in un agente terzo che si inserisce nel mezzo dello scambio falsificando le chiavi pubbliche delle due parti A e B. Per questo motivo l'unico metodo per permettere che non ci sia il rischio di questo tipo di attacco è che le chiavi pubbliche di Alice e Bob vengano verificate da un algoritmo di autenticazione o da una Certification Authority, cioè un soggetto terzo degno di fiducia (pubblico o privato che sia), che abbia una procedura di certificazione, standardizzata a livelli internazionali, europei e/o nazionali, che emetta un certificato digitale.

3.3.3 Algoritmi per il problema del logaritmo discreto

Generazione di chiavi in un sistema con logaritmo discreto

Abbiamo dei parametri di dominio pubblico, (p, q, g), dove:

- p è un numero primo,
- q è un numero primo divisore di p-1,
- g è un numero appartenente all'intervallo [1, p-1] e tale che abbia ordine q, cioè che $t = q$ è il minimo intero positivo che soddisfi l'equazione modulare $g^t \equiv 1 \mod p$.

Una chiave privata sarà un intero x selezionato casualmente dall'intervallo [1, q-1], e la chiave pubblica corrispondente è $y = g^x \mod p$. Determinare il valore di x dati i parametri pubblici è il problema del logaritmo discreto.

Algoritmo di generazione dei parametri:

```
Input:  parametri di sicurezza (l, t)

Output: parametri pubblici (p, q, g)

    1. Seleziona un primo q di t-bit e un primo p di l-bit t.c. q|p-1
    2. Seleziona un elemento g di ordine q nel seguente modo:
        2.1. Seleziona un h in [1, p-1] e computa g = h^((p-1)/q)(mod p)
        2.2. Se g = 1, torna al passo 2.1.
    3. return (p, q, g)
```

Algoritmo di generazione della coppia di chiavi:

```
Input: Parametri pubblici (p, q, g)

Output: Chiave pubblica y e chiave privata x

    1. Seleziona x nell'intervallo [1, q-1]
    2. Computa y = g^x (mod p)
    3. return (y, x)
```

Crittaggio utilizzando ElGamal scheme

```
Input: DL domain parameters (p, q, g), public key y,
       plaintext m in [0, p-1].

Output: Ciphertext (c_1, c_2).

    1. Select k in R [1, q-1].
    2. Compute c_1=g^k (mod p).
    3. Compute c_2=m * y^k (mod p).
    4. Return (c_1, c_2).
```

Decrittaggio utilizzando ElGamal scheme

```
Input: DL domain parameters (p,q,g), private key x,
       ciphertext (c_1, c_2).

Output: Plaintext m.

    1. Compute m=c_2 * (c_1)^(-x) (mod p).
    2. Return(m).
```

3.3.4 Rottura logaritmo discreto

Ricordiamo che il problema del logaritmo discreto ha parametri p e q, dove p è un primo di l-bit e q è un divisore primo a t-bit di p-1; la dimensione dell'input è $O(l)$ bit. Gli algoritmi più veloci conosciuti per risolvere il problema del logaritmo discreto sono: Number Field Sieve (NFS) che ha un tempo di esecuzione previsto sub-esponenziale di $L_p[1/3, 1.923]$, e l'algoritmo rho di Pollard che ha un tempo di esecuzione previsto di $\sqrt{\frac{\pi * q}{2}}$. L'algoritmo rho di Pollard può essere facilmente parallelizzato in modo che i singoli processori non debbano comunicare tra loro e solo occasionalmente comunicare con un processore centrale. Inoltre, l'algoritmo ha solo requisiti di archiviazione e di memoria principale molto piccoli. Il metodo di scelta per risolvere una data istanza della DLP dipende dalle dimensioni dei parametri p e q, che a loro volta determinano quale delle espressioni precedenti rappresenta il minor sforzo computazionale. In pratica, i parametri DL sono selezionati in modo che i tempi di esecuzione previsti nelle espressioni siano approssimativamente uguali.

3.4 Curve ellittiche

Della crittografia a curva ellittica possiamo dire che è computazionalmente più difficile (harder) sia dell'RSA che del DLP. Infatti se in questi due abbiamo bisogno di utilizzare chiavi ad almeno 1024 bit, nell'ECC si utilizza un minimo di 160 bit, anche se nella maggior parte dei casi si usano 256 bit, come ad esempio nei Bitcoin. Una differenza significativa è che, nel caso della blockchain in cui si devono salvare tutti i dati, anche un solo bit in più è un'enorme svantaggio.

Cos'è una curva ellittica? Una curva che segue l'equazione:

$$y^2 = x^3 + ax + b \quad \forall a, bin F_p$$

(un campo di interi modulo p, con p primo) e dove a, b soddisfano

$$4a^3 + 27b^2 \neq 0 \mod p$$

Il grafico della curva appena descritta è simmetrico rispetto all'asse delle ascisse, ed è standardizzato, in particolare va usato un gruppo ciclico con un punto generatore, che permette di creare la curva generando tutti i suoi punti in maniera randomica. L'ordine deve essere uguale a p.
Per calcolarlo può essere utilizzato l'algoritmo di Schoof, il primo algoritmo di tempo polinomiale creato appunto da René Schoof nel 1985, basato sul teorema di Hasse, che permette di contare il numero di punti su curve ellittiche su campi finiti, permettendo così di giudicare la difficoltà di risoluzione del problema del logaritmo discreto.
Un esempio classico di curva ellittica è la Bitcoin Curve, che ha come parametri a=0, b=7, quindi segue la formula $y^2 = x^3 + 7$

Un'importante operazione del gruppoè l'addizione, che consiste nel prendere due punti, congiungerli nell'unico altro punto di intersezione con la curva, proiettarlo verticalmente sul suo opposto, e il punto trovato da questa proiezione è la somma dei due.
Per sommare un punto P a se stesso basta prendere l'opposto dell'intersezione della tangente del punto P.

Pendenza (slope s) della retta tra due punti P e Q:

$$s = \frac{Y_q - Y_p}{X_q - X_p}$$

Per raggiungere un qualsiasi punto della curva, quindi, sfruttiamo l'algoritmo di moltiplicazione quadratica, grazie al quale basta calcolare il prodotto del punto generatore per se stesso per ottenere i punti generati dal prodotto del punto per le potenze di due, e poi semplicemente ottenere qualsiasi valore dalla somma dei punti ottenuti (es: Calcolo P, 2P, 4P, 8P, ecc e se voglio il valore 12P basta sommare 8P a 4P).

Chiave privata:
Corrisponde a quante volte sommo il punto generatore a se stesso. Dato che il gruppo è ciclico, avrò lo stesso concetto della crittografia DH: la chiave pubblica sono le coordinate del punto sul quale arrivo. Il tutto avviene in $\mod p$

Le funzioni di crittaggio e decrittaggio sono le stesse che vengono utilizzate nel protocollo di scambio di chiavi Diffie-Hellman:
Private Key Alice genera Public Key Alice

Private Key Bob genera Public Key Bob
Condividono le proprie chiavi pubbliche a cui ciascuno aggiunge la propria chiave privata; così formano un segreto comune da utilizzare per lo scambio.

Descriverò ora l'algoritmo di generazione di chiavi, e poi quelli di crittazione e decrittazione tramite l'utilizzo dello schema di ElGamal.

3.4.1 Algoritmi per crittazione con curva ellittica

Generazione coppia di chiavi in curve ellittiche

```
Input: Elliptic curve domain parameters (p, E, P, n).

Output: Public key Q and private key d.

    1. Select d in R[1,n-1].
    2. Compute Q = dP.
    3. Return(Q,d).
```

Crittaggio di ElGamal per curve ellittiche

```
Input: Elliptic curve domain parameters (p, E, P, n),
       public key Q, plaintext m.

Ouput: Ciphertext (C_1, C_2).

    1. Represent the message m as a point M in E(F_p).
    2. Select k in R[1, n-1].
    3. Compute C_1 = kP.
    4. Compute C_2 = M + kQ.
    5. Return(C_1, C_2).
```

Decrittaggio di ElGamal per curve ellittiche

```
Input: Domain parameters (p, E, P, n), private key d,
       ciphertext (C_1, C_2).

Output: Plaintext m.

    1. Compute M = C_2–d*C_1, and extract m from M.
    2. Return(m).
```

3.4.2 Generare curve con sage

Per poter generare e lavorare semplicemente con delle curve ellittiche, ancora una volta il grande potere di calcolo di sage ci viene in aiuto: infatti esiste la classe *sage.schemes.elliptic$_c$urves.constructor.EllipticCurveFactory* che permette di costruire curve ellittiche in una semplice chiamata di funzione che può avvenire in diversi modi:

```
Inserendo come argomento l'etichetta di una curva ellittica
pre-esistente e presente nel database chiamato "Cremona":
```

```
sage: EllipticCurve('37b2')
   Elliptic Curve defined by
   y^2 + y = x^3 + x^2 - 1873*x - 31833 over Rational Field
sage: EllipticCurve('5077a')
   Elliptic Curve defined by
   y^2 + y = x^3 - 7*x + 6 over Rational Field
sage: EllipticCurve('389a')
   Elliptic Curve defined by
   y^2 + y = x^3 + x^2 - 2*x over Rational Field
```

Oppure inserendo direttamente l'equazione di Weierstrass:

```
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: EllipticCurve(y^2 - ( x^3 + x - 9 ))
   Elliptic Curve defined by
   y^2 = x^3 + x - 9 over Rational Field

sage: R.<x,y> = GF(5)[]
sage: EllipticCurve(x^3 + x^2 + 2 - y^2)
   Elliptic Curve defined by
   y^2 = x^3 + x^2 + 2 over Finite Field of size 5
```

Sage include sea.gp, una veloce implementazione
dell'algoritmo SEA (Schoff–Elkies–Atkin)
per contare il numero di punti su una curva ellittica su Fp

```
sage: v = cremona_optimal_curves([11..37])
sage: E = v[0]
sage: E.cardinality()                    # calcolato in meno di un secondo!
   99999999999371984255
```

3.4.3 Crittoanalisi

Per tentare di attaccare questo sistema si utilizza un tipo di attacco chiamato "Square root attack", poiché se avessimo una chiave lunga 2^{160}, servirebbero 2^{80} tentativi per risolverlo. Non è conosciuto un metodo più efficiente per attaccare questo sistema. Però se usassimo l'algoritmo di Shoor con dei computer quantistici, il problema verrebbe risolto con facilità, perciò c'è bisogno di migliorare gli algoritmi di cifratura in maniera tale che diventino difficili da risolvere anche per computer quantistici.

NUMS (Number Up My Sleeve) problem: essendo le curve ellittiche state standardizzate in grandissima parte dall'NSA, c'è la possibilità che siano state usate curve solo all'apparenza sicure, che però dà loro la possibilità di risolvere il problema bypassando l'enorme complessità computazionale. Anche per la curva scelta per i bitcoin, Satoshi Nakamoto non ha dato mai spiegazioni. Infatti gli esperti di crittografia hanno espresso la preoccupazione che la National Security Agency abbia inserito una backdoor cleptografica in almeno un generatore pseudo casuale basato su curve ellittiche. Memo interni trapelati dall'ex appaltatore della NSA Edward

Snowden suggeriscono che la NSA abbia inserito una backdoor nello standard Dual EC DRBG. Un'analisi della possibile backdoor ha concluso che un avversario in possesso della chiave segreta dell'algoritmo potrebbe ottenere chiavi di crittografia dati solo 32 byte (256 bits) di output PRNG.

Il progetto SafeCurves è stato lanciato al fine di catalogare le curve che sono facili da implementare in modo sicuro e sono progettate in modo completamente verificabile pubblicamente per ridurre al minimo la possibilità di una backdoor.

3.5 Firma digitale

Una **firma digitale (digital signature)** è un metodo matematico volto a dimostrare che qualcosa sia stato effettivamente mandato da noi. Per fare ciò va trovato un modo per inviare dati senza che ci sia la possibilità di essere confusi con qualcun'altro, né la possibilità che ciò che viene inviato venga mandato a nome tuo da un agente terzo. Per questo motivo è stata sviluppata la firma digitale.
Come abbiamo detto precedentemente, il crittaggio dei dati avviene a grandi linee applicando una chiave pubblica al tuo messaggio, che verrà poi decrittato con una chiave privata dal destinatario desiderato e, si spera, solo da lui. Il sistema di firma digitale è in qualche modo simile, solo che anziché applicare una chiave pubblica al messaggio, vi si applica la propria chiave privata, in modo che il ricevente possa effettivamente, utilizzando la chiave pubblica, capire che il mittente sia proprio quello desiderato.

4 punti chiave:

	Simmetrica	Asimmetrica
Confidentiality	SI	SI
Autenticity	SI	SI
Integrity	SI	SI
Ownership	NO	SI

- Confidenzialità → Non rivelare la propria chiave privata
- Autenticità → La chiave privata ha autenticato il messaggio e nient'altro
- Integrità → Il processo rende chiaro che nessuno ha armeggiato con niente
- Proprietà (Ownership) → Possibilità di reclamare la proprietà di ciò che si sta inviando: può esistere solo nella crittografia asimmetrica, poiché nella simmetrica abbiamo la chiave in comune, quindi non c'è possibilità di autenticarsi

3.5.1 Utilizzo di DS nei bitcoin

Quando qualcuno ti invia bitcoin, li invia al tuo indirizzo.
Se voleste spendere qualsiasi bitcoin inviato al vostro indirizzo, create una transazione e specificate dove i vostri bitcoin devono andare. Una transazione di questo tipo può assomigliare a:

Trasferire 5 bitcoin da 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa (il tuo indirizzo) a 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX (l'indirizzo del destinatario).
Naturalmente, chiunque può creare una transazione che assomiglia a quella di cui sopra, quindi se fosse aggiunta alla blockchain così com'è e senza problemi, allora

perdereste oltre 192.775 euro (al 15/01/2022 almeno), che ti piaccia o no. Fortunatamente, una tale transazione non appartiene alla blockchain, perché manca di una firma digitale valida. Aggiungendo una firma digitale, puoi dimostrare di conoscere la chiave privata che corrisponde all'indirizzo 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfN. Se non conosci la chiave privata corrispondente, allora probabilmente non avresti dovuto dire alle persone di inviarti bitcoin su quell'indirizzo, dato che non sei in grado di spendere nessuno dei bitcoin inviati lì!

Quando crei un indirizzo bitcoin per te (o un indirizzo/conto per qualsiasi altra criptovaluta), generi prima la chiave privata. Dalla chiave privata, si calcola la chiave pubblica corrispondente e facendo l'hashing della chiave pubblica si ottiene il proprio indirizzo. Si spera che tu non possa scegliere prima un indirizzo e poi determinare la chiave privata da quello, altrimenti potresti determinare la chiave privata per qualsiasi indirizzo usando lo stesso metodo.

Come abbiamo visto, chiavi pubbliche, chiavi private e firme digitali formano i componenti di base della crittografia a chiave pubblica. Non importa quale base matematica sia usata per implementare un sistema crittografico a chiave pubblica, esso deve soddisfare i due requisiti seguenti, almeno per i nostri scopi:

1. È computazionalmente impossibile ricavare la chiave privata corrispondente a una data chiave pubblica.

2. È possibile dimostrare che si conosce la chiave privata corrispondente a una chiave pubblica senza rivelare alcuna informazione utile sulla chiave privata nel processo. Inoltre, tale prova può essere costruita in modo da richiedere un messaggio specifico da verificare. In questo modo, la prova forma una firma digitale per quel messaggio.

Ricapitolando, quindi, un modo per fare crittografia a chiave pubblica è con le curve ellittiche. Un altro modo è con RSA. La maggior parte delle criptovalute - Bitcoin ed Ethereum inclusi - usano le curve ellittiche, perché una chiave privata a curva ellittica a 256 bit è sicura quanto una chiave privata RSA a 3072 bit. Le chiavi più piccole sono più facili da gestire e con cui lavorare.

Bibliography

[1] W.M. Baldoni, C. Ciliberto, G.M. Piacentini Cattaneo, Aritmetica, crittografia e codici, 2006

[2] Jean-Philippe Aumasson, SERIOUS CRYPTOGRAPHY, A Practical Introduction to Modern Encryption, 2018

[3] Neal Koblitz and Alfred Menezes and Scott Vanstone, Guide to Elliptic Curve Cryptography, 2004