POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria matematica

Tesi di Laurea

Crittografia e Mining

Relatore
prof. Antonio Jose' Di Scala

Candidato
Enrico Guglielmino

Supervisore aziendale
PING-S S.r.l.
Dott.ssa Alessandra Rostagno

Anno Accademico 2021-2022

Indice

Introduzione

L'obbiettivo di questo lavoro è stato effettuare un'**analisi di fattibilità del mining**. Nel primo capitolo vengono introdotti alcuni importanti concetti preliminari alla Crittografia. Dopo alcuni richiami di Algebra vengono trattate le funzioni Hash, in particolare SHA e Kekkak-256. Dopo una breve introduzione ai sistemi simmetrici, viene descritto nel dettaglio il funzionamento di una sistema a chiave pubblica (asimmetrico). Nello specifico i criptosistemi RSA, Rabin, ElGamal, Massey-Omura e i criptosistemi su curve ellittiche. Per ognuno di questi sono presenti degli esempi di calcolo per esplicitare e comprendere nella pratica come funzionano tali sistemi. Infine vengono descritti i principali protocolli di base, in particolare i protocolli di firma digitale e di scambio chiavi. Nel secondo capitolo viene analizzata la Blockchain partendo dalle sue caratteristiche generali. Dopo aver introdotto il concetto di Proof of Work (PoW) viene analizzata la soluzione proposta da S. Nakamoto al problema dei generali bizantini, che utilizza una catena di PoW. Infine viene descritta la Proof of Stake (PoS) e i relativi vantaggi e svantaggi rispetto alla PoW. Il terzo capitolo è incentrato su Bitcoin. Dopo una descrizione della struttura dei blocchi e dei concetti di Merkel tree e Merkel root, vengono analizzate le transazioni in Bitcoin. Prima della descrizione vera e propria del protocollo di Proof of Work vengono spiegati i processi di verifica delle transazioni e di validazione dei blocchi. Viene infine analizzato il processo di creazione di Bitcoin. Il quarto capitolo ha una struttura molto simile a quello precedente ma è incentrato su Ethereum. Dopo la spiegazione del processo di generazione di una chiave pubblica a partire da una chiave privata scelta in maniera casuale vengono analizzati gli indirizzi di Ethereum: Externally Owned Account e Contract Account. Dopo una descrizione delle transazioni viene infine descritto il cosiddetto The Merge, che ha sancito il passaggio della Blockchain di Ethereum da PoW a PoS e che ha provocato una hard fork. Nel quinto capitolo viene introdotto uno degli aspetti fondamentali legati al mining: i mining pool. Viene quindi introdotto il concetto di share, fondamentale per la descrizione del meccanismo di ridistribuzione di un pool. Vengono infine trattate le principali macchine utilizzate per fare mining: GPU, ASIC, Rig. Il sesto ed ultimo capitolo apre una panoramica sull'analisi di fattibilità del mining. Dopo una spiegazione dei passaggi principali per il collegamento di un Rig ad un pool, viene descritto uno dei principali calcolatori di profitti per i miners di criptovalute: Whattomine. Segue un modello per la derivazione del profitto medio derivante dall'attività di mining. Dopo l'analisi del modello viene infine descritto un simulatore che dati

certi dati in input, restituisce il guadagno medio giornaliero insieme ad altri importanti specifiche. La conclusione del capitolo è incentrata su una riflessione sulla Crittografia post-quantum.

Capitolo 1

Introduzione alla Crittografia

In questo capitolo vengono introdotti alcuni importanti concetti preliminari di Crittografia, necessari per la comprensione dei capitoli successivi.

1.1 Richiami di Algebra

Definizione

Siano a e b due interi. Diciamo che a divide b (o che a è divisore di b) se $\frac{b}{a}$ è intero. Scriviamo $a \mid b$ per indicare che a divide b, altrimenti $a \nmid b$.

Proprietà

Per tutti gli interi a, b, c con $a \neq 0$, si ha:

- $a \mid 0$
- $1 \mid a$
- $a \mid a$
- se $a \mid b$, e $b \mid c$, allora $a \mid c$
- se $a \mid b$, o $a \mid c$, allora $a \mid bc$

Definizione

Dati due interi a e b che non siano entrambi uguali a zero, si dice **Massimo Comune Divisore (MCD)** di a e b il più grande intero che divide sia a che b. Indichiamo il MCD di a e b con (a, b):

$$(a, b) = \max\Big\{n \in \mathbb{Z} : n \mid a \quad \text{e} \quad n \mid b\Big\} \tag{1.1}$$

Proprietà

Valgono le seguenti proprietà:

- $(a, b) = (b, a) \quad \forall a, b \in \mathbb{Z}$
- $(a,\, 0) = a \quad \forall a \in \mathbb{Z}$
- $(a,\, 1) = 1$

Lemma di Bezout

Siano a e b due interi. Allora esistono due interi x e y tali che

$$ax + by = (a, b) \tag{1.2}$$

1.1.1 Aritmetica modulare

Definizione

Siano $a, b, n \in \mathbb{Z}_n$ con $n > 0$. L'intero a si dice congruo a b modulo n e si scrive $a \equiv b \bmod n$, se $n \mid (a-b)$. Equivalentemente $a \equiv b \bmod n$ significa che a e b danno lo stesso resto quando vengono divisi per n. Introduciamo dunque la **classe dei resti modulo** n:

$$\mathbb{Z}_n = \{0,1, ..., n-1\} \tag{1.3}$$

La congruenza modulo n è una relazione di equivalenza. Infatti,

- $a \equiv a \bmod n$ (proprietà riflessiva)
- se $a \equiv b \bmod n$, allora $b \equiv a \bmod n$ (proprietà simmetrica)
- se $a \equiv b \bmod n$, e $b \equiv c \bmod n$, allora $a \equiv c \bmod n$ (proprietà transitiva)

Definizione

Siano a, x, n interi con $n > 0$. Se vale $a\,x \equiv 1 \bmod n$ allora diciamo che x è un **inverso** di a modulo n.

Teorema

Esiste un inverso di a modulo $n \iff (a, n) = 1$. In tal caso l'inverso è unico.

Corollario

Se p è un numero primo allora tutti gli interi non divisibili per p sono invertibili modulo p. Dunque $\mathbb{Z}_p$ è tale che tutti i suoi elementi (tranne lo zero) sono invertibili modulo p.

Teorema di Fermat

Sia p primo e sia $a \in \mathbb{Z}$ tale che $(a, p) = 1$. Allora

$$a^{p-1} \equiv 1 \bmod p \tag{1.4}$$

Funzione di Eulero

La **funzione di Eulero** φ è una funzione definita, per ogni intero positivo n, come il numero degli interi compresi tra 1 e $(n-1)$ che sono coprimi con n, ossia

$$\varphi(n) = \#\Big\{k \in \{1,2,...,n-1\} : (n,k) = 1\Big\} \tag{1.5}$$

In altri termini, $\varphi(n)$ è uguale al numero di rappresentanti modulo n che hanno un inverso modulo n. Ad esempio, $\varphi(8) = 4$ poichè i numeri coprimi di 8 sono 4, ossia: 1,3,5,7. La funzione $\varphi(n)$ è molto importante nella teoria dei numeri in quanto rappresenta la cardinalità del gruppo moltiplicativo degli interi modulo n, più precisamente l'ordine del gruppo moltiplicaitvo $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$. La funzione di Eulero è moltiplicativa: per ogni coppia di interi a e b tali che $\mathrm{MCD}(a, b) = 1$ si ha

$$\varphi(ab) = \varphi(a)\varphi(b) \tag{1.6}$$

Se inoltre p è primo si ha $\varphi(p) = p - 1$. Se n è semiprimo, ovvero $n = pq$ con p e q primi distinti, allora

$$\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = n - p - q + 1 \tag{1.7}$$

Teorema di Eulero

Il **teorema di Eulero** afferma che se n è un intero positivo e x è coprimo con n allora

$$x^{\varphi(n)} \equiv 1 \bmod n \tag{1.8}$$

dove $\varphi(n)$ indica la funzione di Eulero.

Osserviamo che il teorema di Eulero è una generalizzazione del teorema di Fermat. Infatti, se $n = p$ primo, allora $\varphi(n) = \varphi(p) = p - 1$.

Teorema di Eulero generalizzato

Sia n prodotto di due primi distinti. Se $m \equiv 1 \bmod \varphi(n)$ allora

$$a^m \equiv a \bmod n \quad \forall a \in \mathbb{Z} \tag{1.9}$$

Definizione

Siano a, n interi con $(a, n) = 1$ e $n > 0$. Si dice **ordine** di a modulo n, il più piccolo intero positivo t tale che $a^t \equiv 1 \bmod n$.

1.1.2 Gruppi

G insieme non vuoto si dice **gruppo** rispetto all'operazione " $\cdot$ " se:

- $\forall a, b \in G \quad a \cdot b \in G$ (chiusura)
- $\forall a, b, c \in G \quad a \cdot (b \cdot c) = (a \cdot b) \cdot c$ (proprietà associativa)
- $\exists\, e \in G :\ a \cdot e = e \cdot a = a \quad \forall a \in G$ (elemento neutro)
- $\forall a \in G, \exists\, b \in G :\ a \cdot b = b \cdot a = e$ (inverso)

Definizione

Sia $(G, \cdot)$ un gruppo. Se $\forall a, b \in G$ vale la proprietà $a \cdot b = b \cdot a$, allora G è detto **gruppo commutativo** o **gruppo abeliano**.

Lemma (unicità dell'elemento neutro)

Sia G un gruppo. L'**elemento neutro** di G è unico.

Spesso l'elemento neutro viene indicato con 1_G o semplicemente con 1.

Lemma (unicità dell'inverso)

Sia G un gruppo. Per ogni $a \in G$ l'**inverso** di a è unico.

Indicheremo l'inverso di a con a^{-1}.

Definizione

Siano G un gruppo, $a \in G$ e $n > 0 \in \mathbb{Z}$. Definiamo le potenze di a come

$$a^0 = 1 \qquad a^n = \underbrace{a \cdot a \cdot \ ... \ \cdot a}_{n \text{ volte}} \qquad a^{-n} = (a^{-1})^n$$

Proprietà

Per ogni $a, b \in G$ e per ogni $n, m \in \mathbb{Z}$, si ha:

- $a^n a^m = a^{n+m}$
- $(a^n)^m = a^{nm}$
- $(ab)^{-1} = b^{-1}a^{-1}$

Notazione moltiplicativa		Notazione additiva	
elemento neutro	1	elemento neutro	0
prodotto	ab	somma	$a + b$
inverso	a^{-1}	opposto	$-a$
potenze	a^n	ripetizioni	na

Esempi

I seguenti sono gruppi:

- I numeri interi $\mathbb{Z}$, i numeri razionali $\mathbb{Q}$, i numeri reali $\mathbb{R}$ e i numeri complessi $\mathbb{C}$ con l'operazione di somma (notazione additiva).

- $\mathbb{Q}^*, \mathbb{R}^*, \mathbb{C}^*$ con il prodotto (notazione moltiplicativa).
- Le matrici $n \times n$ con entrate reali e determinante non nullo con il prodotto di matrici (notazione moltiplicativa).
- Le funzioni biettive $S \longrightarrow S$ (con S insieme qualsiasi) con l'operazione di composizione di funzioni.

I seguenti non sono gruppi:

- I numeri interi positivi $\mathbb{Z}^+$ con il prodotto (non esiste l'inverso).
- I numeri reali $\mathbb{R}$ con il prodotto (0 non ha inverso).

Definizione

Si definisce **gruppo additivo degli intetri modulo n** l'insieme $\mathbb{Z}_n$ rispetto all'operazione somma modulo n e si denota con $(\mathbb{Z}_n, +)$.

Definizione

Si definisce **gruppo degli interi moltiplicativi modulo n** l'insieme

$$\mathbb{Z}_n^* = \left\{ a \in \mathbb{Z}_n : (a, n) = 1 \right\} \tag{1.10}$$

con l'operazione prodotto modulo n e si denota con $(\mathbb{Z}_n^*, \cdot)$.

Definizione

Sia G un gruppo e sia $H \subset G$. H si dice **sottogruppo** di G, se H è gruppo rispetto alla stessa operazione di G.

Esempi

- $(2\mathbb{Z}, +)$ è sottogruppo di $(\mathbb{Z}, +)$.
- $(1 + 2\mathbb{Z}, +)$ non è sottogruppo di $(2\mathbb{Z}, +)$.

Lemma

Siano G un gruppo e sia H un sottoinsieme non vuoto di G. Allora H è un sottogruppo di $G \iff \forall a.b \in H$ si ha $ab \in H$. In tal caso, l'elemento neutro di H è uguale all'elemento neutro di G.

Definizione

Sia G un gruppo. Si dice **ordine** di G la cardinalità dell'insieme: $\text{ord}(G) = \#G$. Se $\text{ord}(G) < +\infty$ il gruppo si dice **finito**.

Nota: $\mathbb{Z}$ e $\mathbb{Z}_n$ sono gruppi finiti di ordine rispettivamente n e $\varphi(n)$.

Lemma

Siano G un gruppo e $a \in G$. Allora l'insieme $H =< a >= \{a^0, a^1, a^2, ...\}$ è un sottogruppo di G detto sottogruppo generato da a.

Definizione

Siano G un gruppo e $a \in G$. L'ordine del sottogruppo generato da a è detto ordine di a. Equivalentemente si definisce ordine di a il più piccolo $t \in \mathbb{Z}$ tale che $a^t = 1$.

Definizione

Sia G un gruppo. Se esiste $g \in G$ tale che G è uguale al sottogruppo generato da g, cioè $G =< g >= \{g^0, g^1, g^2, ...\}$, allora G è detto **gruppo ciclico** e g è detto **generatore** di G.

Teorema di Lagrange

Se H è un sottogruppo di G allora l'ordine di H divide l'ordine di G.

Corollario

Sia G un gruppo finito. Allora $\forall g \in G \quad \text{ord}(g) \mid \text{ord}(G)$.

Se $\text{ord}(G) = p$ con p primo, allora $\forall g \in G \quad \text{ord}(g) = 1 \quad \text{oppure} \quad \text{ord}(g) = p$.

1.1.3 Anelli

A si dice **anello** rispetto a $(+, \cdot)$ se

- $(A, +)$ è un gruppo abeliano
- $\forall a, b, c \in A$ vale $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ (proprietà associativa rispetto al prodotto)

- $\forall a, b \in A \quad a \cdot b \in A$ (chiusura rispetto al prodotto)
- $\forall a, b, c \in A$ si ha $a \cdot (b + c) = ab + ac,\ (a + b) \cdot c = ac + bc$ (proprietà distributiva)

Definizione

Sia A un anello. A è detto **anello commutativo** se $\forall a, b \in A$ vale $a \cdot b = b \cdot a$. Se $\exists 1 \in A$, l'anello si dice **unitario**.

Definizione

Un anello commutativo unitario dove tutti gli elementi non nulli sono invertibili si dice **campo**.

Proprietà

$(\mathbb{Z}_n, +, \cdot)$ è un campo $\iff n$ è primo.

Nota: nelle sezioni successive utilizzeremo la notazione $\mathbb{Z}_p = \mathbb{F}_p$.

Definizione

Un campo con un numero finito di elementi è detto **campo finito**.

1.2 Funzioni Hash

Definito con Σ l'alfabeto e con Σ^* l'insieme di tutte le parole di lunghezza arbitraria ottenibili da Σ, chiamiamo **funzione hash** una funzione $h : \Sigma^* \longrightarrow \Sigma^n$, con n fissato. Tipicamente nelle applicazioni $\Sigma = \{0,1\}$ e $n = 160,256,384$ o 512. Chiamiamo $h(x)$ impronta di x (in inglese: hash o **digest**). Ovviamente $h(x)$ non può essere iniettiva. Siamo interessati a funzioni hash che hanno l' **effetto valanga**: cambiando anche solo un carattere in input, l'output cambia radicalmente.

Collisione

Una coppia di elementi $(a, b) \in \Sigma^*, a \neq b$, tale che $h(a) = h(b)$ si dice **collisione**.

Funzione hash unidirezionale

Una funzione hash si dice **unidirezionale** se, data un'impronta z, è un problema computazionalmente intrattabile ricavare x tale che $h(x) = z$.

Funzione hash debolmente priva di collisioni

Una funzione hash si dice **debolmente priva di collisioni** se per $a \in \Sigma^*$ fissato, è un problema computazionalmente intrattabile determinare b tale che $h(a) = h(b)$.

Funzione hash fortemente priva di collisioni

Una funzione hash si dice **fortemente priva di collisioni** se determinare una qualsiasi collisione (a, b) è un problema computazionalmente intrattabile.

Alcune possibili applicazioni delle funzioni hash in crittografia sono le seguenti:

- **Sicurezza di immutabilità**: dato un testo per renderlo immutabile si può calcolare e archiviare la sua hash. In qualsiasi momento ricalcolandone la hash (se coincide) si ha la certezza che il testo non è stato modificato.
- **ZKP (Zero-Knowledge-Proof)**: invece di mostrare un documento si può provare di averlo, pubblicando la sua hash. Mostrando poi il documento in un secondo momento, chiunque può verificare, calcolandone la hash e verificando che coincide con quella pubblicata, che il documento era proprio quello.
- **PoW (Proof of Work)**: con le funzioni hash si può dare la prova di aver fatto un lavoro.

1.2.1 SHA (Secure Hash Algorithm)

Con il termine **SHA (Secure Hash Algorithm)** si indica una famiglia di cinque diverse funzioni crittografiche di hash sviluppate a partire dal 1993 dalla **National Security Agency (NSA)** e pubblicate dal **National Institute of Standards and Technology (NIST)** come standard federale dal governo degli USA. SHA-1 produce un digest di 160 bits. Gli altri 4 algoritmi sono indicati genericamente come SHA-2 e sono: SHA-224, SHA-256, SHA-384 e SHA-512 (ognuno di essi produce un digest di lunghezza in bit pari al numero indicato nella loro sigla). La funzione crittografica di hash che si usa in Bitcoin è SHA-256.

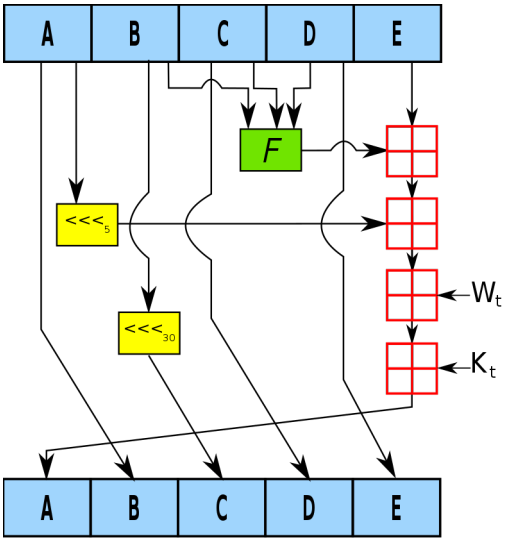

Figura 1.1. SHA-256.

1.2.2 Keccak-256

Ethereum utilizza la funzione hash crittografica **Keccak-256**. Keccak-256 è stata candidata per la **SHA-3 Cryptographic Hash Function Competition** indetta nel 2007 dal National Institute of Science and Technology. Keccak è stato l'algoritmo vincente, standardizzato come **Federal Information Processing Standard (FIPS-202)** nel 2015. Tuttavia, durante il periodo in cui Ethereum è stato sviluppato, la standardizzazione adottata dal NIST non era ancora stata finalizzata. Il NIST ha modificato alcuni parametri di Keccak dopo il completamento del processo di standardizzazione, presumibilmente per migliorare la sua efficienza. Alcune controversie derivanti dalla pubblicazione di documenti che implicavano degli interessi tra il NIST e la National Security Agency hanno portato a un significativo ritardo nella standardizzazione di SHA-3. All'epoca, Ethereum decise di implementare l'algoritmo Keccak originale, anzichè lo standard SHA-3 modificato dal NIST.

Come facciamo a sapere se la libreria che stiamo utilizzando implementa FIPS-202 SHA-3 o Kecck-256, dal momento che entrambi vengono chiamati "SHA-3"? Un semplice modo per capirlo è usare un vettore test: un output atteso per un dato input. Il test più comunemente usato riguarda l'utilizzo di una stringa vuota in input. Se si applicano le due funzioni hash alla stringa vuota si osservano i seguenti

risultati:

Keccak256(" ") = c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470
SHA3(" ") = a7ffc6f8bf1ed76651c14756a061d662f580ff4de43b49fa82d80a4b80f8434a

Indipendentemente da come viene chiamata la funzione, si può eseguire questo semplice test per capire se si tratta dell'originale Keccak-256 o dello standard NIST finale FIPS-202 SHA-3.

1.3 Sistemi simmetrici

I primi sistemi crittografici che sono stati inventati avevano la caratteristica di essere costruiti tramite un algoritmo in cui la chiave che si usa per criptare e decriptare era la stessa. Tali sistemi vengono chiamati **sistemi simmetrici**. I difetti principali di questi sitemi sono:

- necessità di un grande numero di chiavi (una per ogni coppia di utenti)
- difficoltà nello scambio sicuro delle chiavi.

Infatti nei sistemi simmetrici ogni coppia di utenti deve avere una chiave segreta condivisa (usata sia per codificare che per decodificare). Il numero di chiavi necessarie per avere scambi sicuri di documenti tra n utenti è

$$\binom{n}{2} = \frac{n!}{2(n-2)!} = \frac{n(n-1)}{2} \sim \frac{n^2}{2} \tag{1.11}$$

cosa che tende a far esplodere velocemente il numero delle chiavi necessarie. Inoltre c'è il problema di scambiarsi in modo sicuro le chiavi, cosa non problematica se gli utenti sono pochi e si possono facilmente incontrare, ma molto delicata se gli utenti sono molti e distanti tra di loro, come negli scenari più moderni.

1.4 Sistemi a chiave pubblica (asimmetrici)

La definizione di sistema a chiave pubblica è stata proposta nel 1976 da Diffie e Hellman in [4]. Nei sistemi a chiave pubblica ogni utente si genera autonomamente due chiavi, una pubblica e una privata. Di conseguenza non c'è il problema dello scambio chiavi, in quanto ogni utente se le può generare autonomamente. La chiave pubblica di un utente serve a tutti gli altri per cifrare un documento a lui destinato

e la chiave privata viene usata dall'utente per decifrare. In questo modo il numero di chiavi necessarie è soltanto $2n$, che in caso di alto numero di utenti risulta essere molto inferiore alle circa $\frac{n^2}{2}$ chiavi necessarie in un sistema simmetrico. Dal momento che la chiavi pubbliche sono note a tutti è fondamentale, nei sistemi a chiave pubblica, avere qualche sistema per garantire l'autenticità del mittente (**firma digitale**). Tuttavia nella pratica i sistemi asimmetrici sono utilizzabili per il trasferimento di un piccola quantità di dati. Invece i sistemi simmetrici riescono a criptare grandi quantità di dati molto velocemente. L'introduzione della crittografia a chiave pubblica non ha reso obsoleta la crittografia simmetrica. Questo perchè a fronte di una funzionalità migliore i sistemi a chiave pubblica hanno però in genere una maggiore lunghezza delle chiavi e necessitano di maggior tempo per la cifratura/decifratura. Spesso la crittografia a chiave pubblica e quella simmetrica si usano insieme: il sistema asimmetrico si utilizza come sistema di scambio chiavi e una volta scambiata la chiave quello simmetrico viene utilizzato per il trasferimento dei dati.

1.4.1 Sistema RSA

Il primo e più famoso dei sistemi a chiave pubblica è **RSA** (dalle iniziali dei suoi inventori Rivest, Shamir e Adleman) pubblicato nel 1977 ([4]). Ogni utente (chiamiamolo A) compie le seguenti operazioni una volta:

- A sceglie due numeri primi grandi distinti p e q.
- A calcola $n = p\,q$ e $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1) = n-p-q+1$.
- A sceglie $e \in \mathbb{Z}^*_{\varphi(n)}$ e calcola d tale che $e\,d \equiv 1 \bmod \varphi(n)$.
- A rende nota la coppia (n, e), che costituisce la sua chiave pubblica.
- A tiene segreta la coppia $(\varphi(n), d)$, che costituisce la sua chiave privata.

Lo spazio dei messaggi in chiaro e dei messaggi cifrati sono quindi $\mathbb{Z}_n$ e lo spazio delle chiavi è invece $\mathbb{Z}^*_{\varphi(n)}$ (RSA non è un criptosistema perfetto). Ricordiamo che $\mathbb{Z}^*_{\varphi(n)}$ è l'insieme degli elementi invertibili di $\mathbb{Z}_{\varphi(n)}$. Notiamo che con RSA se cifriamo due volte lo stesso messaggio otteniamo due volte lo stesso cifrato. Osserviamo inoltre che, dal momento che n è pubblico, conoscere $\varphi(n)$ è equivalente a conoscere p e q. Se un qualsiasi utente vuole cifrare un messaggio da mandare ad A è sufficiente usare la funzione unidirezionale (one-way function) di A:

$$f_A(x) = x^e \bmod n \tag{1.12}$$

A per decifrare utilizza la sua funzione di decrifratura:

$$f_A^{-1}(x) = x^d \bmod n \tag{1.13}$$

Il sistema funziona perchè

$$f_A^{-1}(f_A(x)) = f_A^{-1}(x^e \bmod n) = x^{e\,d} \bmod n \equiv x^{1+k\varphi(n)} \bmod n \equiv x \bmod n \tag{1.14}$$

in quanto per costruzione $e\,d \equiv 1 \bmod \varphi(n)$ e per il teorema di Eulero si ha che $x^{\varphi(n)} \equiv 1 \bmod n$ se x è coprimo con n. E se il messaggio x non è coprimo con n? Il sistema funziona ugualmente per il teorema di Eulero generalizzato.

Nota: il teorema vale non solo per $n = pq$, ma per ogni n squarefree. Il teorema non è piu vero se n ha dei fattori quadratici.

Esempio: $n = 12, \varphi(n) = 4, e = 3$ e quindi $d = 3$.
Se $x = 10$, la sua cifratura è

$$x^e \bmod n \equiv 10^3 \bmod 12 \equiv 4 \bmod 12 \tag{1.15}$$

La decifratura risulta essere

$$x^{ed} \bmod n \equiv 4^3 \bmod 12 \equiv 4 \bmod 12 \tag{1.16}$$

che tuttavia non riporta al messaggio $x = 10$ iniziale.

Tuttavia avendo definito n come prodotto di due primi distinti, nel sistema RSA la cifratura e la decifratura funzionano sempre, nel senso che decifrando un testo cifrato si ritorna sempre al messaggio in chiaro iniziale. Si osservi però che nel caso un utente scriva un messaggio che viene codificato in un numero x non coprimo con n, il sistema è comunque a rischio. Questo perchè se x non è coprimo con n allora p o q dividono x e quindi calcolando il massimo comune divisore tra il messaggio x e n (cosa che si può fare in modo efficiente anche per valori grandi), si determinano i due fattori di n ossia p e q. Conoscendo p e q è possibile ricavare $\varphi(n)$ in quanto n è pubblico e noto $\varphi(n)$ si può calcolare d in quanto e è pubblico, risalendo in questo modo alla chiave segreta dell'utente. Per fortuna la probabilità che questo capiti è molto piccola. I possibili numeri minori di n che sono non coprimi con n sono $n - \varphi(n) = n - (p-1)(q-1) = p + q - 1$ e quindi la probabilità che ciò accade è

$$\frac{p+q-1}{n} = \frac{p+q-1}{pq} = \frac{1}{q} + \frac{1}{p} - \frac{1}{pq} \tag{1.17}$$

che per p e q molto grandi è trascurabile.

La sicurezza del sistema RSA poggia sul fatto che pur conoscendo n nessuno sia in grado di determinare $\varphi(n)$. Questa assunzione è sicuramente falsa per i numeri n piccoli, per i quali si può procedere per forza bruta. E' quindi necessario scegliere n sufficientemente grande da rendere difficile la determinazione di $\varphi(n)$, noto n. Anche per n grandi però se si conosce la fattorizzazione di n diventa immediato determinare $\varphi(n)$, essendo $\varphi(n) = n - p - q + 1$. Ne segue che chi sa fattorizzare n può rompere completamente il sistema RSA, calcolando d a partire dal dato pubblico e, potendo quindi decifrare tutti i messaggi destinati ad A. Non è detto il viceversa: non è noto un algoritmo che permetta di fattorizzare efficientemente n avendone uno per rompere RSA. Dunque RSA non è equivalente al **problema della fattorizzazione**. Si noti che non serve essere capaci di fattorizzare n per rompere completamente RSA; basta saper calcolare efficientemente $\varphi(n)$ dato n. Questi due problemi risultano essere equivalenti. Infatti chi sa fattorizzare n può calcolare banalmente $\varphi(n) = n - p - q + 1$. Mentre chi sa calcolare $\varphi(n)$ dato n può calcolare efficientemente p e q. Questo perchè p e q verificano il sistema

$$\begin{cases} p\,q = n \\ p + q = n - \varphi(n) + 1 \end{cases} \tag{1.18}$$

e quindi sono le due soluzioni (intere) dell'equazione

$$z^2 - (n - \varphi(n) + 1)z + n = 0 \tag{1.19}$$

Per cui, supponendo $p < q$, segue

$$p = \frac{(n - \varphi(n) + 1) - \sqrt{(n - \varphi(n) - 1)^2 - 4n}}{2} \tag{1.20}$$

$$q = \frac{(n - \varphi(n) + 1) + \sqrt{(n - \varphi(n) - 1)^2 - 4n}}{2} \tag{1.21}$$

Siccome esiste un algoritmo efficiente per il calcolo delle radici intere di numeri interi, questo conclude la dimostrazione che conoscendo $\varphi(n)$ e n si possono calcolare efficientemente p e q, e quindi l'equivalenza tra il calcolo della fattorizzazione di n e il calcolo di $\varphi(n)$.

Esempio: Sistema RSA

- Consideriamo $p = 3, q = 5$.
- Segue quindi $n = 15$ e $\varphi(n) = \varphi(15) = \varphi(3)\varphi(5) = 8$.
- Scegliamo $e = 3$ (notiamo come e sia coprimo con $\varphi(n)$).
- Determiniamo d tale che $e\,d \equiv 1 \bmod \varphi(n)$, ossia $d = 3$.

Messaggio	Cifratura	Decifratura
0	$0^3 \bmod 15 = 0$	$0^3 \bmod 15 = 0$
1	$1^3 \bmod 15 = 1$	$1^3 \bmod 15 = 1$
2	$2^3 \bmod 15 = 8$	$8^3 \bmod 15 = 2$
3	$3^3 \bmod 15 = 12$	$12^3 \bmod 15 = 3$
4	$4^3 \bmod 15 = 4$	$4^3 \bmod 15 = 4$
5	$5^3 \bmod 15 = 5$	$5^3 \bmod 15 = 5$
6	$6^3 \bmod 15 = 6$	$6^3 \bmod 15 = 6$
7	$7^3 \bmod 15 = 13$	$13^3 \bmod 15 = 7$
8	$8^3 \bmod 15 = 2$	$2^3 \bmod 15 = 8$
9	$9^3 \bmod 15 = 9$	$9^3 \bmod 15 = 9$
10	$10^3 \bmod 15 = 10$	$10^3 \bmod 15 = 10$
11	$11^3 \bmod 15 = 11$	$11^3 \bmod 15 = 11$
12	$12^3 \bmod 15 = 3$	$3^3 \bmod 15 = 12$
13	$13^3 \bmod 15 = 7$	$7^3 \bmod 15 = 13$
14	$14^3 \bmod 15 = 14$	$14^3 \bmod 15 = 14$

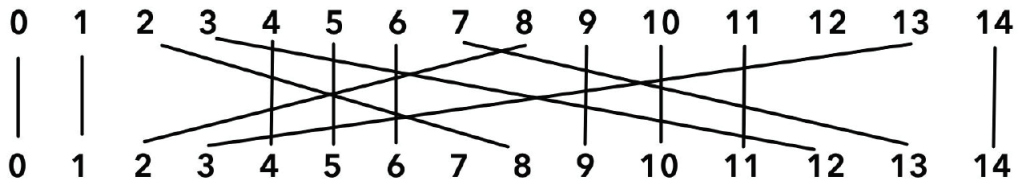

Figura 1.2. RSA: $p = 3, q = 5$.

1.4.2 Criptosistema di Rabin

Il difetto teorico più evidente del sistema RSA è che se si dispone di un algoritmo efficiente per la fattorizzazione allora si può rompere completamente il sistema (ma non è noto il viceversa). Un sistema simile a RSA, che però è più strettamente

legato al problema della fattorizzazione, è stato ideato nel 1979 da Michael Oser Rabin e funziona nel seguente modo:

- Ogni utente A sceglie due primi p e q, entrambi congrui a $3 \bmod 4$ (all'incirca dello stesso ordine di grandezza) e calcola $n = pq$.
- La chiave pubblica è n e la chiave privata è la coppia (p, q).
- La funzione di cifratura è l'elevamento al quadrato dentro $\mathbb{Z}_n$, quindi la cifratura di un messaggio M è $f_A(M) = M^2 \bmod n$.
- Per decifrare l'utente A deve calcolare la radice modulo n, che però purtroppo non è unica e questo complica un po' la decifratura.

Senza conoscere p e q è computazionalmente intrattabile calcolare la radice modulo $n = pq$, se p e q sono grandi. Conoscendo invece la chiave segreta (p, q) esiste un sistema per determinare le 4 radici modulo n. Sia $C = M^2 \bmod n$. L'algoritmo per il calcolo delle 4 radici quadrate di C è il seguente:

- Si calcolano $m_p = C^{(p+1)/4} \bmod p$ e $m_q = C^{(q+1)/4} \bmod q$
- Si osserva che $\pm m_p$ e $\pm m_q$ sono le radici quadrate in $\mathbb{Z}_p$ e $\mathbb{Z}_q$ rispettivamente
- Con l'algoritmo di Euclide si calcolano a e b tali che $ap + bq = 1$
- Si calcolano $M_1 = apm_q + bqm_p, M_2 = apm_q - bqm_p, M_3 = -apm_q + bqm_p$ e $M_4 = -apm_q - bqm_p$, che sono le 4 radici quadrate di C in $\mathbb{Z}_n$.

Rimane solo da determinare quale M_i è il messaggio M di partenza. Se M è un messaggio in una lingua qualsiasi, con enorme probabilità solo un M_i è un testo comprensibile. Se invece M non è un testo, ma sono dati qualsiasi, non è così facile capire qual'è la radice giusta. In tal caso un modo semplice per procedere è quello di concordare una sequenza standard con cui iniziare o concludere il messaggio M, o comunque stabilire una forma particolare che deve avere M. Si procede quindi come detto per determinare le 4 radici M_1, M_2, M_3 e M_4 e poi si identifica M tra le quattro radici andando a vedere qual'è l'unica che inizia o termina con la sequenza stabilita o che ha la forma particolare fissata.

La rottura del sistema di Rabin è computazionalmente equivalente alla fattorizzazione (cosa che non era vera per RSA). In una direzione è banale: chi è in grado di fattorizzare efficientemente può determinare la chiave privata (p, q) dalla chiave pubblica n e rompere completamente il sistema. La cosa interessante è che per

il sistema di Rabin vale anche il viceversa: chi è in grado di rompere il codice di Rabin, cioè chi sa calcolare efficientemente le radici modulo n, può fattorizzare n efficientemente.

Esempio: Criptosistema di Rabin

Sia $(p, q) = (7,3), M = 5$. Dunque

- La chiave pubblica è $n = pq = 7 \cdot 3 = 21$ e la chiave privata è la coppia (7,3).
- La cifratura del messaggio M è $C = M^2 \bmod n = 5^2 \bmod 21 \equiv 4$.
- Si calcolano $m_p = C^{\frac{p+1}{4}} \bmod p = 4^2 \bmod 7 \equiv 2$ e $m_q = C^{\frac{q+1}{4}} \bmod q = 4^1 \bmod 3 \equiv 1$.
- ± 2 e ± 1 sono le radici rispettivamente in $\mathbb{Z}_7$ e $\mathbb{Z}_3$.
- Si calcolano a e b tali che $ap + bq = 1$. In questo caso si ha $1 \cdot 7 - 2 \cdot 3$ e dunque $a = 1, b = -2$.
- Si calcolano $M_1 = apm_q + bqm_p = 1 \cdot 7 \cdot 1 - 2 \cdot 3 \cdot 2 = -5 \bmod 21 \equiv 16, M_2 = apm_q - bqm_p = 1 \cdot 7 \cdot 1 + 2 \cdot 3 \cdot 2 = 19, M_3 = -apm_q + bqm_p = -1 \cdot 7 \cdot 1 - 2 \cdot 3 \cdot 2 = -19 \bmod 21 \equiv 2, M_4 = -apm_q - bqm_p = -1 \cdot 7 \cdot 1 + 2 \cdot 3 \cdot 2 = 5$.

Il messaggio originario risulta quindi essere $M_4 = M = 5$.

1.4.3 Criptosistema di ElGamal

Il **criptosistema di ElGamal** è un sistema a chiave pubblica che si basa sulla presunta difficoltà di risoluzione del **problema del logaritmo discreto**. Il sistema è così definito:

- Si sceglie una volta per tutte un primo grande p e un generatore g di $\mathbb{Z}_p^*$ (p e g sono pubblici).
- Ogni utente sceglie la propria chiave privata $x \in \mathbb{Z}_{p-1}$ e rende pubblico g^x.
- Supponiamo che A abbia chiave privata x e pubblica $a = g^x$ e B abbia chiave privata y e pubblica $b = g^y$.

Se l'utente A vuole inviare un messaggio M all'utente B:

- A sceglie a caso $k \in \mathbb{Z}_{p-1}$ e invia a B la coppia (g^k, Mb^k).

- B può elevare alla y (sua chiave privata) il primo termine della coppia ricevuta: $(g^k)^y = (g^y)^k = b^k$ e quindi calcolarne l'inverso b^{-k}.
- B può quindi moltiplicare il secondo termine della coppia ricevuta da A per b^{-k} ricavando $M = Mb^k b^{-k}$.

La sicurezza del sistema è basata sulla supposizione che sia impossibile in tempi brevi ricavare k dalla conoscenza di g e di g^k (problema del logaritmo discreto), nel qual caso il malintenzionato potrebbe poi calcolare facilmente b^{-k} e quindi M. Se un utente riuscisse a risolvere in tempi ridotti il problema del logaritmo discreto potrebbe addirittura risalire da una qualsiasi chiave pubblica a quella privata, rompendo completamente il sistema.

Esempio: Criptosistema di ElGamal

Sia $p = 11, < g >=< 7 >= \mathbb{Z}_{11}$. Supponiamo che l'utente A vuole inviare il messaggio $M = 5 \in \mathbb{Z}_{11}$ all'utente B. Allora

- A sceglie la chiave privata $x = 2 \in \mathbb{Z}_{10}$ e rende pubblico $g^x = 7^2 \bmod 11 \equiv 5$.
- B sceglie la chiave privata $y = 3 \in \mathbb{Z}_{10}$ e rende pubblico $g^y = 7^3 \bmod 11 \equiv 2$.
- A sceglie $k = 4 \in \mathbb{Z}_{10}$ e invia a B la coppia $(g^k, Mb^k) = (7^4, 5 \cdot 2^4) \equiv (3,3)$.
- B eleva alla $y = 3$ il primo termine della coppia ricevuta $(g^k)^y = 3^3 \bmod 11 \equiv 5 = b^k$ e calcola quindi l'inverso $b^{-k} = 9$.
- B può quindi moltiplicare il secondo termine della coppia ricevuta da A per b^{-k} ricavando il messaggio $M = Mb^k b^{-k} = 3 \cdot 9 \bmod 11 \equiv 5 \bmod 11$.

1.4.4 Criptosistema di Massey-Omura

Il **criptosistema di Massey-Omura** è basato sull'idea del doppio lucchetto. Il sistema è così definito:

- Si sceglie un primo grande p.
- Ogni utente sceglie $e \in \mathbb{Z}^*_{p-1}$ e ne calcola l'inverso $d \equiv e^{-1} \bmod (p-1)$.

Supponiamo che A abbia la sua coppia (e_A, d_A) e B la sua coppia (e_B, d_B). Se l'utente A vuole inviare un messaggio $M \in \mathbb{Z}_p$ all'utente B:

- A calcola $C = f_A(M) = M^{e_A} \bmod p$, e lo spedisce a B.

- B applica la sua funzione di cifratura e calcola $D = f_B(C) = C^{e_B} \bmod p \equiv M^{e_A e_B} \bmod p$, che rispedisce ad A.
- A leva il suo lucchetto elevando quanto ricevuto alla d_A, ottenendo $E = f_A^{-1}(D) = D^{d_A} \bmod p \equiv M^{e_B} \bmod p$, e rimanda a B.
- Infine B eleva quanto ricevuto alla d_B e ottiene $f_B^{-1}(E) = E^{d_B} \bmod p \equiv M \bmod p$.

Questo metodo ha il difetto che se un utente C intercetta il messaggio $M^{e_A} \bmod p$ che al primo step A ha mandato a B, potrebbe calcolare $(M^{e_A})^{e_C}$ e rimandarlo ad A, spacciandosi per B. Se A non è in grado di sapere chi gli sta mandando il messaggio, leva il suo lucchetto e rimanda a C, che così potrebbe leggere il messaggio. Per tale ragione a questo sistema si accoppia sempre un sistema di firma digitale.

Esempio: Criptosistema di Massey-Omura

Sia $p = 13$. Supponiamo che l'utente A vuole inviare il messaggio $M = 2 \in \mathbb{Z}_{13}$ all'utente B. Allora

- A sceglie $e_A = 5 \in \mathbb{Z}_{12}$ e ne calcola l'inverso $d_A \equiv 5^{-1} \bmod \mathbb{Z}_{12} \equiv 5$.
- B sceglie $e_B = 7 \in \mathbb{Z}_{12}$ e ne calcola l'inverso $d_B \equiv 5^{-1} \bmod \mathbb{Z}_{12} \equiv 7$.
- A calcola $C = M^{e_A} \bmod p = 2^5 \bmod 13 \equiv 6$.
- B applica la sua funzione di cifratura e calcola $D = C^{e_B} \bmod p = 6^7 \bmod 13 \equiv 7$ che rispedisce ad A.
- A leva il suo lucchetto elevando quanto ricevuto alla d_A, ottenendo $E = D^{d_A} \bmod p = 7^5 \bmod 13 \equiv 11$, e rimanda a B.
- Infine B eleva quanto ricevuto alla d_B e ottiene $E^{d_B} = 11^7 \bmod 13 \equiv 2 = M$.

Pretty Good Privacy (PGP)

PGP è un sistema inventato da Phil Zimmermann nel 1991. E' una interessante combinazione di crittografia simmetrica e asimmetrica. Si suppone di avere a disposizione due sistemi: uno simmetrico e uno asimmetrico. IL PGP è così definito:

- Il mittente usa la chiave pubblica del ricevente per cifrare la chiave di sessione del sistema simmetrico
- Il mittente usa la chiave di sessione per cifrare con il sistema simmetrico il suo messaggio e invia al ricevente sia il messaggio cifrato che la chiave di sessione cifrata
- Il ricevente , usando la sua chiave privata, decifra la chiave simmetrica di sessione e quindi la usa per decifrare il messaggio.

1.4.5 ECC (Elliptic Curve Cryptosystems)

RSA è stato il primo e il più usato sistema a chiave pubblica per anni, fino alla scoperta degli **ECC** (**Elliptic Curve Cryptosystems**). L'introduzione della crittografia su curve ellittiche nel 1985 ha rivoluzionato la crittografia a chiave pubblica. Questi nuovi sistemi sono costruiti in modo simile a RSA, ma invece che essere costruiti sugli anelli $\mathbb{Z}_n$ sono costruiti sullo spazio dei punti di una curva ellittica. ECC è stata proposta indipendentemente da Neal Koblitz in [2] e S.Miller in [3]. Attualmente la crittografia su curve ellittiche è una tra le più valide ed efficienti alternative, compresa RSA(ECC con chiavi a 256 bit è più forte di RSA con chiavi a 4096 bit). Tuttavia la sua complessità ne ha ritardato molto l'adozione, entrando nei sistemi crittografici solo nei primi anni 2000.

Definizione

Una **curva piana algebrica** è l'insieme delle coppie (x, y) che soddisfano un polinomio della forma $f(x, y) = 0$.

Definizione

Sia K un campo. Una **curva ellittica in forma breve di Weierstrass** E definita su K è una curva algebrica definita come

$$y^2 = x^3 + Ax + B \tag{1.22}$$

dove $A, B \in K$ tale che $4A^3 + 27B^2 \neq 0$. La condizione su A e B serve a garantire che ogni punto della curva abbia una ed una sola retta tangente. Dunque su una curva ellittica E non sono presenti autointersezioni, punti angolosi, cuspidi o punti isolati.

Osservazioni

- $4A^3 + 27B^2 \neq 0 \Longrightarrow$ la curva E è regolare.
- Considereremo sempre $K = \mathbb{F}_p$, con $p > 3$, in modo tale che la proprietà di regolarità sopra definita risulti sempre soddisfatta.

Definizione

Una **curva ellittica** E può quindi essere espressa nella seguente forma

$$E(\mathbb{F}_p) = \{(x, y) \in E : x, y \in \mathbb{F}_p\} \cup \{0\} = \tag{1.23}$$
$$= \{(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p : y^2 = x^3 + Ax + B\} \cup \{0\} \tag{1.24}$$

Teorema di Hasse

Sia $\#E(\mathbb{F}_p)$ il numero di punti della curva ellittica $E(\mathbb{F}_p)$. Allora

$$p + 1 - 2\sqrt{2} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{2} \tag{1.25}$$

Teorema

Dati due punti P e Q di una curva ellittica $E(\mathbb{F}_p)$, è possibile definire un terzo punto detto somma di P e Q ed indicato con $P \oplus Q$. Con questa somma è possibile dimostrare che $E(\mathbb{F}_p)$ è un gruppo commutativo:

- $\forall P, Q \in E(\mathbb{F}_p) \quad P \oplus Q = Q \oplus P \in E(\mathbb{F}_p)$
- vale la proprietà associativa
- l'elemento neutro è 0
- $\forall P \in E(\mathbb{F}_p), \exists Q \in E(\mathbb{F}_p)$ tale che $P \oplus Q = 0$

La somma di P e Q è definita nel seguente modo:

- Si traccia la retta r passante per P e Q.
- La retta r interseche rà la curva in un terzo punto R.
- Si riflette R lungo l'asse orizzontale e il punto trovato è $P \oplus Q$.

Caso 1: $P \neq Q$.
Siano $P = (x_P, y_P), Q = (x_Q, y_Q) \in E$.

$$\begin{cases} E : y^2 = x^3 + Ax + B \\ r : y = \lambda(x - x_P) + y_P \end{cases} \tag{1.26}$$

dove

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P} \tag{1.27}$$

Si ha

$$P \oplus Q = \Big(\underbrace{\lambda^2 - x_P - x_Q}_{x_R}, \lambda(x_P - x_R) - y_P \Big) \tag{1.28}$$

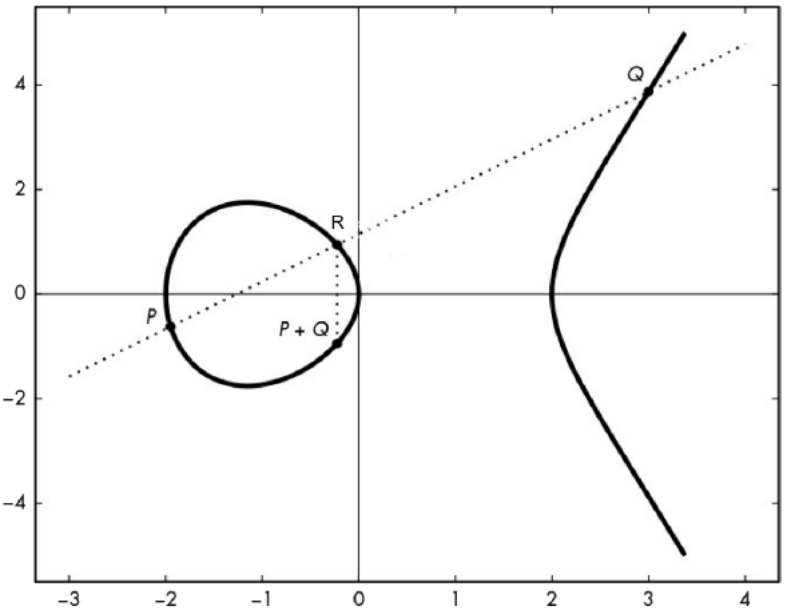

Figura 1.3. Somma di punti su una curva ellittica: Caso $P \neq Q$.

Caso 2: $P = Q$.
Sia $P = (x_P, y_P)$. In questo caso si traccia la retta tangente alla curva nel punto P e si ottiene

$$2P = P + P = \Big(\underbrace{\lambda^2 - 2x_P}_{x_R}, \lambda(x_P - x_R) - y_P \Big) \tag{1.29}$$

dove

$$\lambda = \frac{3x_P^2 + A}{2y_P} \tag{1.30}$$

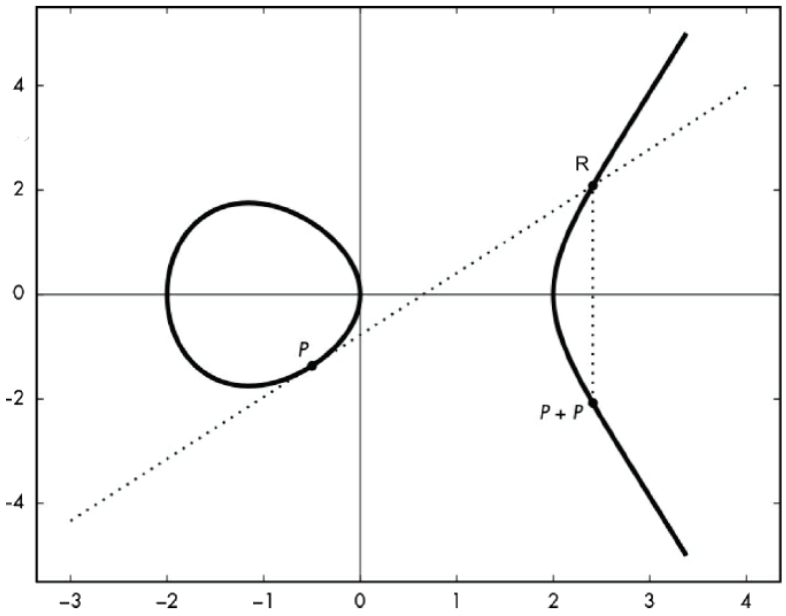

Figura 1.4. Somma di punti su una curva ellittica: Caso $P = Q$.

Nota: quando la retta tangente alla curva in P non interseca la curva in alcun punto si ha $P + P = 0$.

I grafici che sono stati mostrati per comprendere le operazioni sui punti di una curva ellittica sono stati costruiti disegnando tutti i punti del piano reale che verificano l'equazione $y^2 = x^3 + Ax + B$. Siccome però nelle applicazioni crittografiche i valori di x e y che verificano l'equazione sono presi in un campo $\mathbb{Z}_p$, e l'equazione è verificata modulo p, la vera rappresentazione è di tipo discreto. Se per esempio si considera la curva utilizzata in Bitcoin $y^2 = x^3 + 7$, con un numero p ragionevolmente piccolo come $p = 17$, si ottiene uno spazio di 18 punti (17 rappresentati sul grafico, più il punto all'infinito). Ovviamente non è una bella idea sommare k volte per calcolare kP. Con il **Fast Exponentiation Algorithm** si fa molto prima. A titolo di esempio si può calcolare $9P$ calcolando $2P$,$4P = 2P + 2P$,

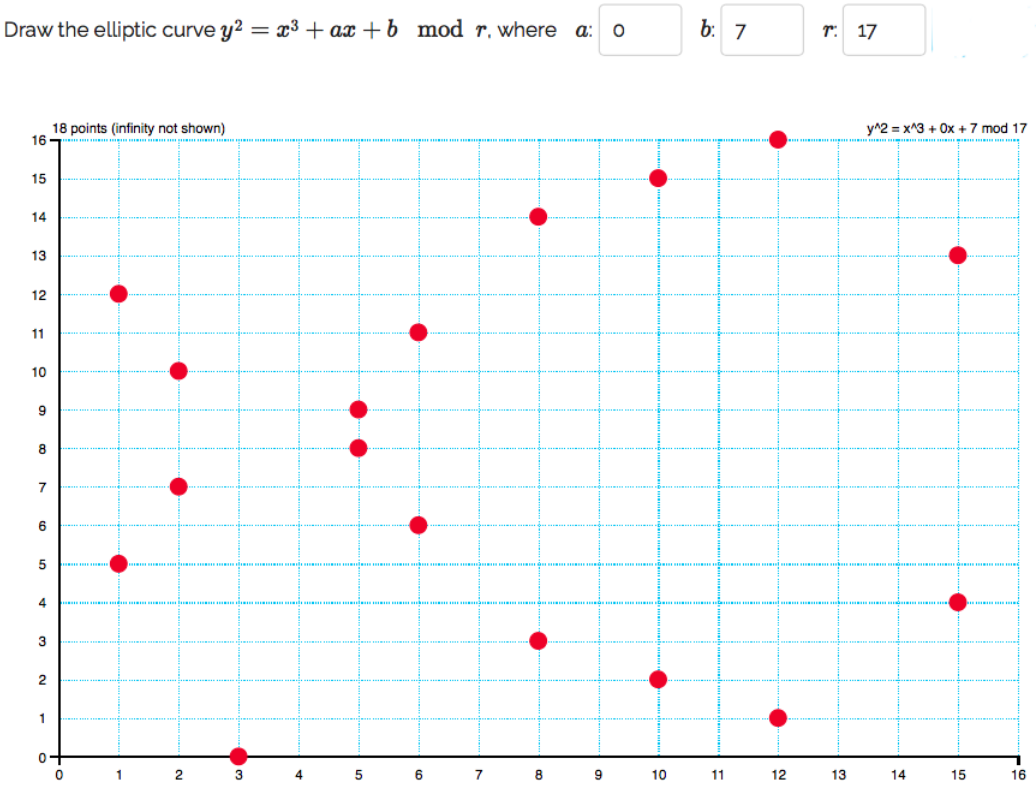

Figura 1.5. Rappresentazione sul piano $\mathbb{R}^2$ della curva $y^2 = x^3 + 7 \bmod 17$.

quindi $8P = 4P + 4P$ e infine $9P = 8P + P$, facendo 4 somme invece di 9. L'analogo del problema del calcolo del logaritmo discreto in $\mathbb{Z}_q$(determinare x, dati g e $y = g^x$, DLP-Discrete Logarithm Problem), sulle curve ellittiche diventa determinare k dati P e $Q = kP$, il cosiddetto **ECDLP (Elliptic Curve Discrete Logarithm Problem**. Il ECDLP è più complesso del DLP: si possono quindi utilizzare numeri relativamente piccoli, mantenendo il livello di sicurezza che con il classico DLP si ha con numeri di dimensione molto maggiore. Data una curva ellittica e uno spazio $\mathbb{Z}_p$ su cui la si definisce (con p molto grande per rendere inattaccabile a forza bruta il sistema) e scelto un punto G della curva, la creazione delle coppie chiave pubblica/chiave privata è molto semplice:

- si sceglie a caso un numero d (chiave privata)
- si calcola dG (chiave pubblica).

E' quindi immediato calcolare la chiave pubblica da quella privata, ma risalire invece dalla chiave pubblica a quella privata è il problema del logaritmo discreto

+	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
∞	∞	(1,5)	(1,12)	(2,7)	(2,10)	(3,0)	(5,8)	(5,9)	(6,6)	(6,11)	(8,3)	(8,14)	(10,2)	(10,15)	(12,1)	(12,16)	(15,4)	(15,13)
(1,5)	(1,5)	(2,10)	∞	(1,12)	(5,9)	(15,13)	(2,7)	(12,1)	(8,14)	(6,6)	(6,11)	(10,15)	(8,3)	(15,4)	(12,16)	(5,8)	(3,0)	(10,2)
(1,12)	(1,12)	∞	(2,7)	(5,8)	(1,5)	(15,4)	(12,16)	(2,10)	(6,11)	(8,3)	(10,2)	(6,6)	(15,13)	(8,14)	(5,9)	(12,1)	(10,15)	(3,0)
(2,7)	(2,7)	(1,12)	(5,8)	(12,16)	∞	(10,15)	(12,1)	(1,5)	(8,3)	(10,2)	(15,13)	(6,11)	(3,0)	(6,6)	(2,10)	(5,9)	(8,14)	(15,4)
(2,10)	(2,10)	(5,9)	(1,5)	∞	(12,1)	(10,2)	(1,12)	(12,16)	(10,15)	(8,14)	(6,6)	(15,4)	(6,11)	(3,0)	(5,8)	(2,7)	(15,13)	(8,3)
(3,0)	(3,0)	(15,13)	(15,4)	(10,15)	(10,2)	∞	(8,14)	(8,3)	(12,16)	(12,1)	(5,9)	(5,8)	(2,10)	(2,7)	(6,11)	(6,6)	(1,12)	(1,5)
(5,8)	(5,8)	(2,7)	(12,16)	(12,1)	(1,12)	(8,14)	(5,9)	∞	(10,2)	(15,13)	(3,0)	(8,3)	(15,4)	(6,11)	(1,5)	(2,10)	(6,6)	(10,15)
(5,9)	(5,9)	(12,1)	(2,10)	(1,5)	(12,16)	(8,3)	∞	(5,8)	(15,4)	(10,15)	(8,14)	(3,0)	(6,6)	(15,13)	(2,7)	(1,12)	(10,2)	(6,11)
(6,6)	(6,6)	(8,14)	(6,11)	(8,3)	(10,15)	(12,16)	(10,2)	(15,4)	(1,5)	∞	(1,12)	(2,10)	(2,7)	(5,9)	(3,0)	(15,13)	(12,1)	(5,8)
(6,11)	(6,11)	(6,6)	(8,3)	(10,2)	(8,14)	(12,1)	(15,13)	(10,15)	∞	(1,12)	(2,7)	(1,5)	(5,8)	(2,10)	(15,4)	(3,0)	(5,9)	(12,16)
(8,3)	(8,3)	(6,11)	(10,2)	(15,13)	(6,6)	(5,9)	(3,0)	(8,14)	(1,12)	(2,7)	(5,8)	∞	(12,16)	(1,5)	(10,15)	(15,4)	(2,10)	(12,1)
(8,14)	(8,14)	(10,15)	(6,6)	(6,11)	(15,4)	(5,8)	(8,3)	(3,0)	(2,10)	(1,5)	∞	(5,9)	(1,12)	(12,1)	(15,13)	(10,2)	(12,16)	(2,7)
(10,2)	(10,2)	(8,3)	(15,13)	(3,0)	(6,11)	(2,10)	(15,4)	(6,6)	(2,7)	(5,8)	(12,16)	(1,12)	(12,1)	∞	(8,14)	(10,15)	(1,5)	(5,9)
(10,15)	(10,15)	(15,4)	(8,14)	(6,6)	(3,0)	(2,7)	(6,11)	(15,13)	(5,9)	(2,10)	(1,5)	(12,1)	∞	(12,16)	(10,2)	(8,3)	(5,8)	(1,12)
(12,1)	(12,1)	(12,16)	(5,9)	(2,10)	(5,8)	(6,11)	(1,5)	(2,7)	(3,0)	(15,4)	(10,15)	(15,13)	(8,14)	(10,2)	(1,12)	∞	(8,3)	(6,6)
(12,16)	(12,16)	(5,8)	(12,1)	(5,9)	(2,7)	(6,6)	(2,10)	(1,12)	(15,13)	(3,0)	(15,4)	(10,2)	(10,15)	(8,3)	∞	(1,5)	(6,11)	(8,14)
(15,4)	(15,4)	(3,0)	(10,15)	(8,14)	(15,13)	(1,12)	(6,6)	(10,2)	(12,1)	(5,9)	(2,10)	(12,16)	(1,5)	(5,8)	(8,3)	(6,11)	(2,7)	∞
(15,13)	(15,13)	(10,2)	(3,0)	(15,4)	(8,3)	(1,5)	(10,15)	(6,11)	(5,8)	(12,16)	(12,1)	(2,7)	(5,9)	(1,12)	(6,6)	(8,14)	∞	(2,10)

Figura 1.6. Tabella delle somme dei 18 punti della curva.

sulle curve ellittiche, problema estremamente difficile se lo spazio dei punti della curva è grande.

Il problema della codifica sulle curve elittiche

ECC hanno una complicazione in più rispetto a RSA, Rabin, ElGamal: la codifica del testo. Nei classici sistemi a chiave pubblica, dato un messaggio, lo si divide in blocchi e lo si trasforma ogni blocco con qualche codifica ragionevole in un numero intero. In questo modo ogni messaggio diventa una sequenza di interi, ognuno dei quali può essere visto come un elemento di $\mathbb{Z}_n$ e può quindi essere cifrato e decifrato. Nei sistemi ECC si può procedere nel medesimo modo e trasformare il messaggio in una serie di interi, ma non è così ovvio come codificare questi interi in una serie di punti di una curva ellittica:

- Non è noto un algoritmo deterministico ed efficiente per determinare un grande numero di punti di una arbitraria curva ellittica.
- Anche se esistesse tale algoritmo non sarebbe risolutivo: servirebbe anche un modo sistematico ed efficiente per codificare valori interi m in punti della curva P_M e viceversa.

E' quindi necessario utilizzare dei sistemi di codifica probabilistica. Lavorando adeguatamente è possibile definire dei sistemi con probabilità di fallimento così basse da essere utilizzabili in pratica. Una volta codificato un messaggio M in un punto della curva P_M, serve un sistema per cifrare il punto P_M in un altro punto della curva e poi decifrare conseguentemente. Siccome i punti della curva con la relativa somma, hanno una struttura di gruppo, si possono utilizzare tutti i sistemi di cifratura funzionanti sui gruppi.

Esempio: Cifratura in stile Massey-Omura

Supponiamo di aver fissato una curva ellittica E, su un campo F_q con q grande e che sia noto il numero n dei punti della curva. Supponiamo inoltre che ogni utente scelga un numero casuale e minore di n tale che $(e, n) = 1$. Calcola poi il numero d tale che $e\,d \equiv 1 \bmod n$. Se A vuole mandare un messaggio M a B:

- A codifica M in un punto della curva P_M.
- A manda a B il punto $e_A P_M$.
- B moltiplica per e_B quanto ricevuto e rimanda ad A il punto $e_B e_A P_M$.
- A moltiplica per d_A quanto ricevuto e rimanda a B il punto $d_A e_B e_A P_M = e_B P_M$.
- B moltiplica per d_B quanto ricevuto e ottiene $d_B e_B P_M = M$.
- B decodifica il punto P_M e ottiene il messaggio M.

Secp256k1

La curva ellittica utilizzata da Bitcoin e Ethereum si chiama **Secp256k1** ed è definita nel seguente modo

$$y^2 = x^3 + 7 \bmod p \tag{1.31}$$

dove $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Questa curva è definita su un campo finito di ordine primo invece che su i numeri reali, di conseguenza è costituita da un insieme (di cardinalità elevata) di punti sparsi in due dimensioni. Come generatore della curva viene scelto il punto $G = (x_G, y_G)$, dove

$x_G = 55066263022277343669578718895168534326250603453777594175500187360389116729240$

$y_G = 32670510020758816978083085130507043184471273380659243275938904335757337482424$

L'ordine di Secp256k1, ossia il numero di punti della curva n, è invece pari a

$n = 115792089237316195423570985008687907852837564279074904382605163141518161494337$

Notiamo che p, x, y e n sono da 256 bits (circa 78 cifre decimali). Quando ad esempio Bob vuole creare una coppia chiave privata/pubblica, gli basta scegliere un valore intero d_B (chiave privata) di 256 bits in maniera casuale e poi calcolare la corrispondente chiave pubblica $d_B G = G + G + ... + G$ (d_B volte). La chiave pubblica non è quindi un intero ma un punto della curva, cioè una coppia di valori (x, y) che verificano l'equazione (1.31). In tempi brevi è possibile calcolare la chiave pubblica a partire dalla chiave privata, ma il viceversa corrisponde a risolvere l'Elliptic Curve Discrete Logarithm Problem (attualmente molto al di fuori delle capacità anche dei più potenti calcolatori esistenti).

1.5 Protocolli di base

Definizione

Un **protocollo** è una seria finita di operazioni, che coinvolgono almeno due soggetti, progettato per raggiungere un determinato scopo. Un **protocollo crittografico** è un protocollo che utilizza algoritmi crittografici.

Un protocollo crittografico può subire due tipi di attacco:

- **passivo**: l'attaccante si inserisce sul canale di trasmissione e non fa altro che ascoltare e cercare di trarre informazioni.
- **attivo**: l'attaccante cerca di alterare il protocollo a proprio vantaggio mediante inserimento di messaggi, alterazione di informazioni immagazzinate su un server o contraffazione della propria identità. Si noti che va presupposto che una delle parti coinvolte possa cercare di attaccare il protocollo dall'interno.

Per implementare i protocolli crittografici si utilizzano la crittografia simmetrica, quella asimmetrica e le funzioni hash.

1.5.1 Firma digitale

Analogamente alla firma nella vita reale, la **firma digitale** serve a garantire il mittente di un messaggio o comunque a garantire l'identità di chi compie una certa

operazione. I sistemi a chiave pubblica hanno come lato positivo il numero ridotto di chiavi e soprattutto evitano il problema dello scambio chiavi. D'altra parte, siccome chiunque può cifrare i messaggi usando la chiave pubblica, nasce il problema di autenticare il mittente del messaggio. Ogni sistema a chiave pubblica è dotato del suo sistema di firma digitale e quindi esistono le firme digitali basate su RSA, ElGamal,etc. Esistono poi dei sistemi ad hoc che permettono di creare una firma digitale, il più famoso e utilizzato dei quali è la firma **DSA (Digital Signature Algorithm)**, approvato dal NIST nell'Agosto del 1991 e firma digitale standard per anni. La firma DSA è definita su uno spazio $\mathbb{Z}_p$, con $2^{1023} < p < 2^{1024}$. E' un sistema ancora utilizzato, tuttavia è stato ormai sostituito, nelle applicazioni che necessitano di massima sicurezza, da un sistema analogo ma costruito sui punti di una curva ellittica, l'**ECDSA (Elliptic Curve Digital Signature Algorithm)**.

Schema generale per la firma digitale in un sistema a chiave pubblica

Se Alice vuole mandare il messaggio M a Bob può inviare $f_B(M)$. Solo Bob può leggerlo perchè è l'unico che conosce la chiave privata e che può quindi calcolare f_B^{-1}. Purtroppo però chiunque può inviare $f_B(M)$ a Bob, spacciandosi per Alice. Il protocollo generale di firma digitale per i sistemi a chiave pubblica prevede di inviare insieme a $f_B(M)$ anche la coppia $f_B((f_A^{-1}(s_A)), s_A)$, dove s_A è un testo che contiene il nome di Alice, l'hash del messaggio, un numero progressivo, data e ora della spedizione, IP della macchina da cui è stato spedito e altre informazioni. Bob applica $f_A f_B^{-1}$ al primo termine della coppia ricevuta e verifica che coincida con il secondo termine s_A. Si noti che:

- Solo Alice poteva creare la coppia $(f_B(f_A^{-1}(s_A)), s_A)$, perchè è l'unica che conosce f_A^{-1}.
- Si aggiunge f_B per fare in modo che solo Bob possa calcolare f_B^{-1} e verificare la firma.
- La firma è personalizzata, nel senso che è una firma di Alice che scrive a Bob. Se un terzo utente Carl intercetta il messaggio non può utilizzare la coppia $(f_B(f_A^{-1}(s_A)), s_A)$ per spacciarsi per Alice.

Rimane però un problema: Bob, una volta ricevuto un messaggio firmato da Alice, potrebbe calcolare f_B^{-1} del primo termine della coppia ricevuta e ottenere $f_A^{-1}(s_A)$. A quel punto potrebbe usare la chiave pubblica di Carl e inviare a Carl $f_C(M')$ e

la coppia $(f_C(f_A^{-1}(s_A)),\ s_A)$, spacciandosi per Alice. Un modo per ovviare a questo problema è quello di far dipendere la firma dal messaggio stesso. Alice potrebbe inviare a Bob come firma digitale insieme al messaggio cifrato $f_B(M)$ anche la coppia $(f_B(f_A^{-1}(s_A||h(M))), s_A)$, dove $h(M)$ è il digest (impronta) del messaggio M tramite una fissata e pubblica funzione hash. La firma è così relativa al messaggio M, dunque se Bob vuole firmare un qualsiasi altro messaggio M' spacciandosi per Alice, non può farlo. Infatti se Bob applica f_B^{-1} al primo termine e invia a Carl la coppia $(f_C(f_A^{-1}(s_A||h(M))), s_A)$ e $f_C(M')$, quest'ultimo nota subito l'incongruenza: una volta applicato $f_A f_C^{-1}$ al primo termine risale a $h(M)$ e applicando h ad M' si accorge che $h(M) \neq h(M')$.

Firma digitale ECDSA (Elliptic Curve Digital Signature Algorithm)

Ricordiamo che devono essere fissati e resi noti:

- la curva ellittica da utilizzare
- un punto G sulla curva
- il numero n dei punti della curva
- una funzione h di hash.

Ricordiamo inoltre che ogni utente si sceglie la propria chiave privata d e utilizza come chiave pubblica il corrispondente punto $P = dG$ della curva. Il protocollo di firma digitale ECDSA prevede che colui che firma (il mittente) segua i seguenti step:

- Il mittente calcola il digest $h = h(M)$ del messaggio in chiaro M.
- Sceglie a caso un valore $k \in \mathbb{Z}_n$.
- calcola il punto della curva $kG = (x, y)$
- Fissa $r = x \bmod n$ e calcola $s = (h + rd)/k \bmod n$.
- La firma digitale da allegare al messaggio è la coppia (r,s).

Notiamo che solo il firmatario conosce la sua chiave privata d e quindi solo lui può determinare la firma digitale (r, s). Inoltre la firma è determinata a partire da $h = h(M)$ e quindi firma solo quel messaggio e non può essere riutilizzata per

firmare altri messaggi. Il destinatario verifica la firma digitale (r, s) nel seguente modo:

- Calcola l'inverso del secondo termine della firma, determinando $w = s^{-1} = k/(h + rd)$
- Calcola $u = wh$ e $v = wr$.
- Calcola il punto $Q = uG + vP$, dove $P = dG$ è la chiave pubblica del firmatario.
- Accetta la firma solo se la prima coordinata del punto Q così calcolato, ridotta modulo n, coincide con r.

Il protocollo funziona in quanto

$$Q = uG + vP = whG + wr(dG) = \Big(\frac{kh}{h + rd} + \frac{krd}{h + rd}\Big)G = kG = (x, y)$$

Attualmente una delle firme digitali più usate e sicure è ECDSA. Bitcoin e Ethereum utilizzano questa firma per rendere sicure le transazioni. Questo tipo di firma è ancora oggi sicura, ma non molto flessibile. In alcuni contesti può essere utile avere delle firme multiple o delle firme a soglia e ECDSA permette di ottenere queste possibilità solo in modo poco efficiente.

1.5.2 Scambio chiavi

Per scambiare informazioni tra due utenti in modo sicuro è spesso utile avere una chiave condivisa. Definire un buon protocollo di scambio chiavi diventa quindi un punto cruciale per la sicurezza del sistema. Si possono definire vari protocolli adatti a questo obiettivo. Il più noto e utilizzato è il protocollo di scambio chiavi di **Diffie-Hellman (DH)**.

Protocollo di scambio chiavi di Diffie-Hellman

Descrizione del protocollo:

- A e B scelgono di comune accordo un numero primo grande p e un generatore g di $\mathbb{Z}_p^*$.
- A sceglie la sua chiave privata a e trasmette a B il valore g^a.
- B sceglie la sua chiave privata b e trasmette ad A il valore g^b.

- A calcola $g^{ab} = (g^b)^a$ e B calcola $g^{ab} = (g^a)^b$.

A e B hanno quindi una chiave comune g^{ab}, ma non possono determinare la chiave segreta dell'altro e non lo può fare neppure un'eventuale terza persona che abbia intercettato le loro comunicazioni (se p, a e b sono scelti adeguatamente). Tuttavia l'algoritmo di Diffie-Hellman è vulnerabile all'attacco "Man in the middle", durante il quale un agente terzo può falsificare le chiavi pubbliche di A e B, rispettivamente g^a e g^b, ed ingannare le due parti.

Scambio chiavi di Diffie-Hellman su curve ellittiche

Sfruttando l'analogia con il DLP, si può costruire sulle curve ellittiche uno scambio chiavi alla Diffie.Hellman:

- Si fissa una curva ellittica e un punto P sulla curva.
- A sceglie a caso k_A, calcola $P_A = k_A P$ e lo invia a B.
- analogamente B sceglie a caso k_B, calcola $P_B = k_B P$ e lo invia ad A.
- A calcola $K = k_A P_B = k_A k_B P$.
- B calcola $K = k_B P_A = k_A k_B P$.
- A e B possono usare la chiave condivisa $K = k_A k_B P$.

Normalmente, una volta scambiata la chiave, la si utilizza per la codifica mediante un sistema simmetrico.

Capitolo 2

Introduzione alla Blockchain

Creare scambi o rapporti sicuri tra sconosciuti è un problema usuale nella nostra società. La soluzione classica è quella di affidarsi a una autorità superiore di garanzia che sia terza e a cui tutti riconoscono fiducia (Sovrano,Stato,Banca,Amministrazione locale). L'obiettivo principale della tecnologia Blockchain è proprio quello di **eliminare l'autorità centrale di garanzia**: creare una rete tra persone che non si conoscono, e non hanno quindi nessun motivo di fidarsi l'uno dell'altro, permettendo transazioni (o comunque qualche tipo di interazione tra gli utenti) che siano sicure, senza l'esistenza di una autorità centrale di garanzia.

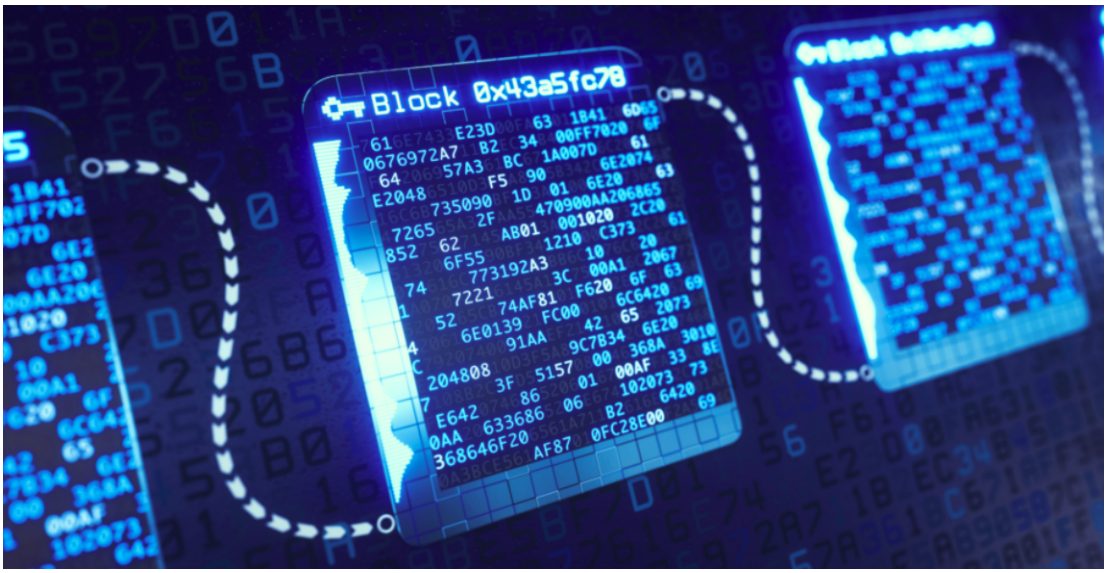

Figura 2.1. Rappresentazione della catena di blocchi della Blockchain.

Tipologie di Blockchain

Esistono 3 tipologie di Blockchain:

- Blockchain pubbliche (permissionless)
- Blockchain ibride
- Blockchain private (permissioned)

Dato che inizialmente tutte le Blockchain erano aperte, è nata una controversia sulla definizione di Blockchain: un sistema privato con dei verificatori autorizzati da un'autorità centrale può essere considerata una Blockchain? Non c' è un accordo generalizzato sulla definizione di Blockchain, ma attualmente le Blockchain più diffuse e utilizzate sono aperte.

2.1 Caratteristiche della Blockchain

La Blockchain può essere descritta come:

- **un database distribuito (Distributed Ledger Technology - DLT)**: uno dei vantaggi della DLT è la **trasparenza** e l' **immutabilità**. A differenza di un sistema centralizzato, tutti i nodi godono di pari diritti sui dati. Tutte le decisioni vengono prese collettivamente. DLT fornisce quindi una traccia di controllo immutabile e verificabile di tutte le operazioni. Un'altra caratteristica è la **resistenza agli attacchi**: la DLT è infatti un sistema più resistente agli attacchi informatici rispetto ai database centralizzati tradizionali poichè è distribuito. Non esiste un unico punto di attacco, il che rende i tentativi di hackerare tali sistemi troppo costosi e inutili.
- **immodificabile dal singolo utente o da gruppi di utenti**: solo l'intera rete tramite un sistema di consenso può accettare l'inserimento di un nuovo blocco.
- una tecnologia che introduce il concetto di **scarsità digitale**: la caratteristica di scarsità digitale è basata sulla struttura stessa della Blockchain e sulla sua immodificabilità. Quando un utente A effettua una transazione a B, quest'ultimo potrebbe utilizzare più volte questo valore e spendere ripetutamente i soldi ricevuti da A (il cosiddetto **double spending**). La soluzione di questo problema è la vera innovazione della tecnologia Blockchain.

La Blockchain è un tipo specifico di DLT in cui i dati sono salvati non in un unico file ma l'informazione è divisa in più blocchi, collegati l'uno con l'altro, in una struttura a catena. In una generica DLT si possono fare in genere 4 operazioni: Create, Retrieve, Update e Delete. In una Blockchain invece solo due: Create e Retrieve. Non si può quindi modificare alcun dato e tanto meno cancellarlo. La Blockchain è quindi una lista in continua crescita di blocchi di dati collegati tra di loro (ogni blocco ha un puntatore al blocco precedente). L'intera catena di blocchi è in possesso di tutti gli utenti della rete, che sono collegati tra di loro in una rete peer-to-peer. Una simile struttura di dati distribuita è per costruzione sostanzialmente inattaccabile e non modificabile fraudolentemente, essendo impossibile attaccare contemporaneamente migliaia o magari milioni di utenti (o trovare con loro un accordo) senza lasciare traccia. Viceversa, quando i dati sono in una sola copia in mano ad una autorità centrale di fiducia, tutta la sicurezza del sistema è basata sul livello di protezione del server centrale dell'autorità e sulla lealtà di chi lavora per l'autorità. Tuttavia se il database è pubblico non c'è privacy e tutti sanno tutto di tutti. Viceversa, quando i dati sono in possesso solo dell'autorità centrale, la privacy è garantita dalla fiducia che gli utenti ripongono nell'autorità centrale. La situazione non è però così negativa come appare a prima vista in quanto mediante l'utilizzo della crittografia si può garantire la privacy delle transazioni pur registrandole su un database condiviso e quindi completamente pubblico. Ad esempio non deve essere possibile risalire dall'indirizzo di una transazione al nome e cognome del mittente e del destinatario in quanto tutte le transazioni sono pubbliche e quindi visibili a tutti gli utenti, ma nessuno sa chi sono le persone che stanno dietro a una certa transazione. Per creare la struttura a blocchi, ogni blocco include l'hash del blocco precedente creando così il collegamento tra due blocchi. L'iterazione di questa procedura forma la catena e garantisce l'integrità del blocco precedente, questo fino al blocco di genesi (il primo blocco della Blockchain). I dati di un blocco, una volta approvato e registrato, non possono essere retroattivamente alterati senza che vengano modificati tutti i blocchi successivi ad esso.

Nel caso in cui due blocchi vengono prodotti simultaneamente si genera una biforcazione (**fork**) nella catena della Blockchain. Ogni Blockchain ha uno specifico algoritmo per gestire le biforcazioni. I blocchi non selezionati per l'inclusione nella catena sono chiamati blocchi orfani, non sono validi e le transazioni in essi contenute non sono valide. Un problema che deve necessariamente affrontare e risolvere ogni Blockchain è quello di stabilire chi e come inserisce nuovi blocchi all'interno

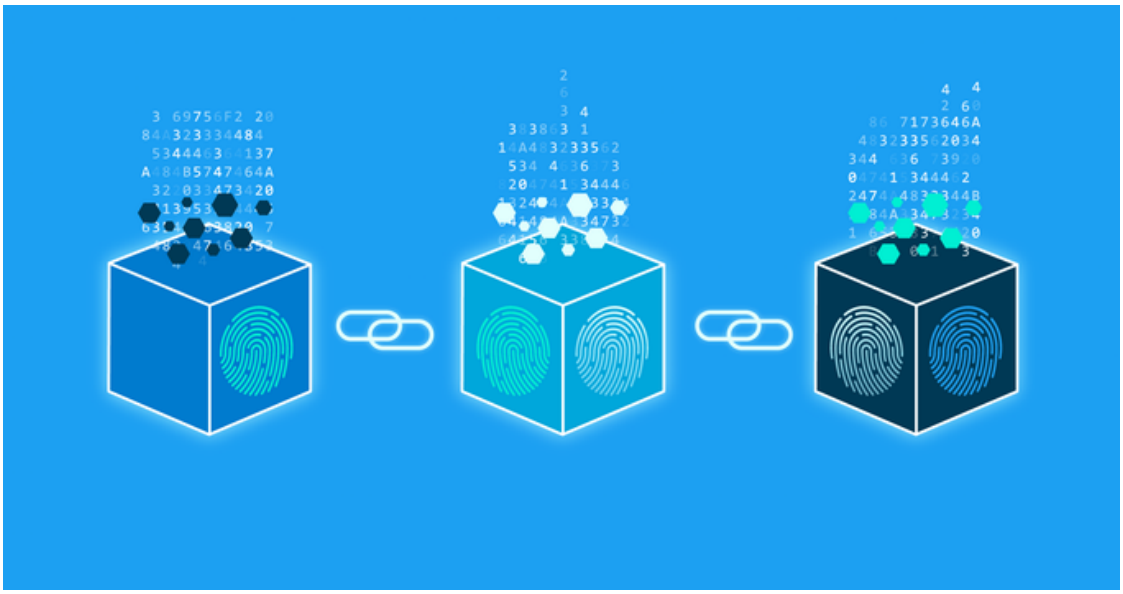

Figura 2.2. Catena di blocchi.

del database condiviso. Se qualcuno fosse in grado di inserire dati a suo piacimento nei blocchi della Blockchain potrebbe inserire transazioni a suo favore. Serve quindi un sistema di inserimento dei blocchi che garantisca la correttezza delle transazioni del blocco. Tutte le Blockchain hanno bisogno di un **protocollo di consenso**, cioè di un algoritmo che permetta a tante persone che non si conoscono, e che quindi non hanno fiducia reciproca, di accordarsi su cos'è un blocco valido da inserire nella Blockchain, nonostante il fatto che un certo numero di utenti potrebbero essere malintenzionati. Il primo dei sistemi che è stato ideato per creare un protocollo di consenso è la **Proof-of-Work** (**PoW**). Vedremo più avanti come Bitcoin sfrutta la PoW per costruire un protocollo di consenso tale da permettere l'inserimento dei nuovi blocchi della Blockchain in modo condiviso dalla rete. Il problema di trovare un consenso non è però nato con la Blockchain, ma è un classico problema dell'informatica e delle telecomunicazioni che vale la pena di approfondire.

2.2 Il problema dei 2 generali

Cominciamo con una versione semplice del problema: il **problema dei 2 generali**.

Descrizione: 2 eserciti sono guidati dai generali $A1$ e $A2$ che vogliono attaccare una città che si trova nel mezzo dei due eserciti. I generali $A1$ e $A2$ possono scambiarsi informazioni solo inviandosi lettere l'un l'altro, ma c'è il problema che il postino deve passare attraverso la città, il che rende possibile la cattura del postino e quindi il messaggio potrebbe non arrivare o arrivare modificato. I generali $A1$ e $A2$ devono accordarsi sull'assalto alla città in modo da attaccare contemporaneamente, che è l'unico modo per vincere.

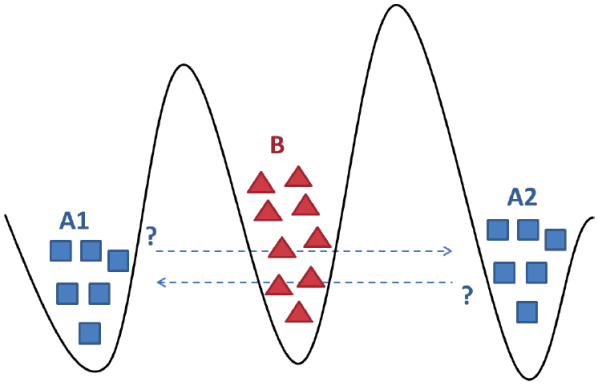

Figura 2.3. Problema dei 2 generali.

I due generali devono quindi ottenere un consenso sull'orario di attacco. Potrebbe sembrare che il problema sia facilmente risolvibile semplicemente eleggendo un comandante tra i due generali, e sarà lui che darà le tempistiche di attacco da inviare all'altro generale, ma non è così semplice. Infatti il generale A1 potrebbe comunicare ad A2 l'ora dell'attacco, ma non saprebbe se il messaggio è arrivato a destinazione. Allora A2, una volta ricevuto il messaggio, potrebbe dare conferma ad A1, ma non saprebbe se A1 l'ha ricevuto. E' quindi evidente che, non importa quanti round di conferma ci siano, non ci sarà modo per ogni generale di assicurarsi che l'altro abbia accettato il piano di attacco. Rimarranno sempre in uno stato di incertezza, chiedendosi se l'ultimo messaggio che hanno inviato è arrivato o meno a destinazione e se è arrivato non modificato. Il problema è molto semplice da enunciare, ma la soluzione non esiste. Più in generale il problema dei due generali dimostra che in un ambiente in cui possono verificarsi problemi di comunicazione, è impossibile garantire l'accordo tra le due parti. Una generalizzazione di questo problema è particolarmente interessante.

2.3 The Byzantine Generals Problem

La generalizzazione del problema dei 2 generali, detta **problema dei generali bizantini**, è stata proposta da tre informatici Leslie Lamport, Robert Shostak e Marshall Pease in un rapporto scientifico dal titolo "The Byzantine Generals Problem" nel 1982.

Descrizione: Il problema dei generali bizantini descrive un gruppo di generali dell'esercito bizantino (esercito dell'Impero Romano d'Oriente), che assedia una città. I generali possono scambiarsi informazioni per raggiungere un accordo sul piano per attaccare la città. Nel caso più semplice, devono solo concordare se attaccare o ritirarsi. Alcuni potrebbero voler attaccare, ma alcuni potrebbero voler ritirarsi, e il problema è che se non attaccano tutti insieme l'attacco fallirà. Supponiamo che i generali si possano scrivere tra di loro e che però ci possano essere dei generali traditori.

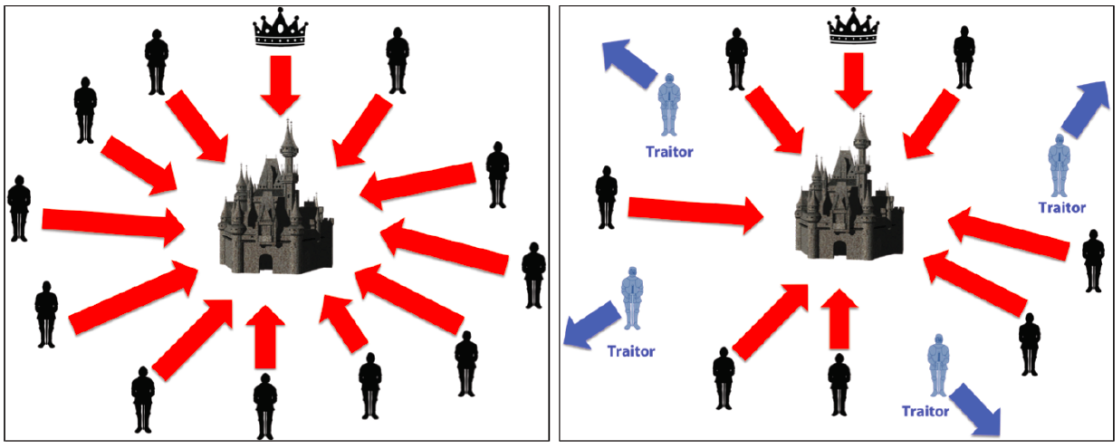

Figura 2.4. A sinistra viene mostrata una situazione di attacco coordinato che porta alla vittoria, mentre a destra notiamo come una situazione di attacco non coordinato (dovuto alla presenza di traditori), porti a una sconfitta.

Si incomincia a vedere la relazione con la Blockchain, essendo questo un problema di raggiungimento di consenso tra più persone, supponendo che alcune di loro (ma non troppe) potrebbero non essere oneste.

Esempio con 5 generali: supponiamo che 2 di loro abbiano proposto di voler attaccare e 2 invece di ritirarsi. Se il quinto generale è un traditore, potrebbe inviare un messaggio ai 2 generali che vogliono attaccare, dicendo che vuole attaccare anche lui e un messaggio ai 2 generali che vogliono ritirarsi dicendo che si ritirerà. Quindi gli attaccanti pensano che attaccare sia la scelta della maggioranza e attaccano (fallendo). Gli altri due pensando che la ritirata sia la scelta della maggioranza e si ritirano. Nonostante la presenza di un solo traditore, il consenso non 'e raggiunto. Dall'esempio si deduce che anche questo problema è tutt'atro che facile da risolvere.

2.4 Commander and Lieutenants

Esistono varie soluzioni al problema dei generali bizantini menzionate nei rapporti scientifici di Lamport, Shostak e Pease. Per risolvere il problema cominciamo col notare che il problema dei generali bizantini può essere risolto facendo riferimento al problema **Commander and Lieutenants** (comandante e luogotenenti).

Descrizione: solo il comandante invia ordini ai luogotenenti e i luogotenenti possono scambiarsi messaggi tra di loro. Tra i luogotenenti ci posso essere dei traditori e anche il comandante lo può essere. Come possono le persone leali raggiungere un consenso sulla decisione di attaccare o ritirarsi in modo da attaccare solo in maggioranza?

Lamport, Shostak e Pease hanno sviluppato l'algoritmo **Oral Messages** (OM) per risolvere questo problema. Affinchè l'algoritmo funzioni è necessario assumere che:

- Ogni messaggio, una volta inviato, verrò consegnato alla corretta destinazione.
- Il destinatario del messaggio saprà esattamente da chi riceve. '
- E' possibile conoscere l'eventuale mancanza di un messaggio (qualcuno che non invia),

Queste sono ipotesi molto forti ma che eliminano il problema "man in the middle" ossia dell'intercettatore che sta in mezzo tra il mittente e il destinatario, che può non fare arrivare il messaggio o modificarlo. Se non si fanno assunzioni di questo tipo non si riesce a risolvere tali problemi, nemmeno il più banale ossia quello dei 2 generali.

Teorema

Sia m il numero di traditori, l'algoritmo OM(m) può raggiungere un consenso se ci sono più di $3m$ generali totali.

In altre parole l'algoritmo OM raggiungerà un consenso quando più dei $\frac{2}{3}$ saranno leali (i traditori devono quindi essere meno di $\frac{1}{3}$).

Esempio
Consideriamo il caso con 4 generali (C, L1, L2 e L3) con un solo traditore.

- Caso 1: supponiamo che il traditore sia L3. C trasmetterà il messaggio v a L1, L2 e L3. L3 tradisce, quindi modifica il messaggio e manda x a L2. Tuttavia, L2 otterrà v da L1 e C, e capisce che la maggioranza ha scelto v. Analogamente i generali C ed L1 raggiungeranno il consenso su v, anche se L3 trasmette il messaggio x. Lo stesso ragionamento vale se il traditore è L1 o L2.

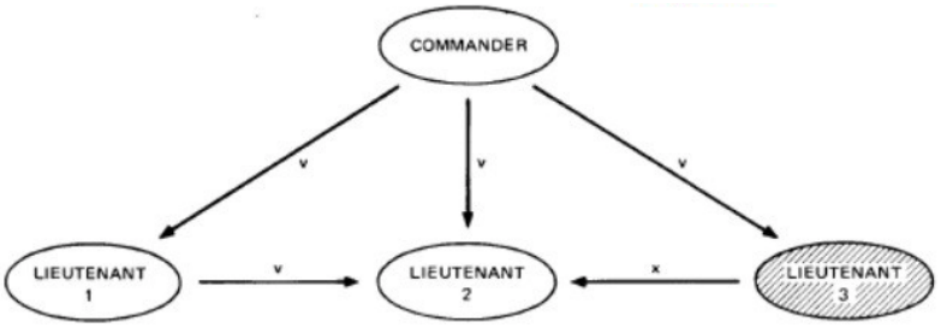

Figura 2.5. Algoritmo OM(1): L3 è l'unico traditore dunque $m = 1$. Il numero dei generali totali è $n = 4 > 3m = 3$, dunque in questo caso si raggiunge il consenso.

- Caso 2: supponiamo il traditore sia C. C può mandare 2 comandi di attaccare a due qualsiasi luogotenenti e 1 comando di non attaccare al terzo luogotenente. Siccome i luogotenenti solo leali e si scambiano i messaggi, tutti riceveranno 2 comandi di attaccare e 1 di non attaccare e attaccheranno tutti insieme. Invece se C manda 2 comandi di non attaccare e 1 di attaccare, a maggioranza i tre luogotenenti decideranno tutti di non attaccare. Ovviamente non c'è nessun problema di consenso se C manda 3 comandi di attaccare o 3 comandi di non attaccare.

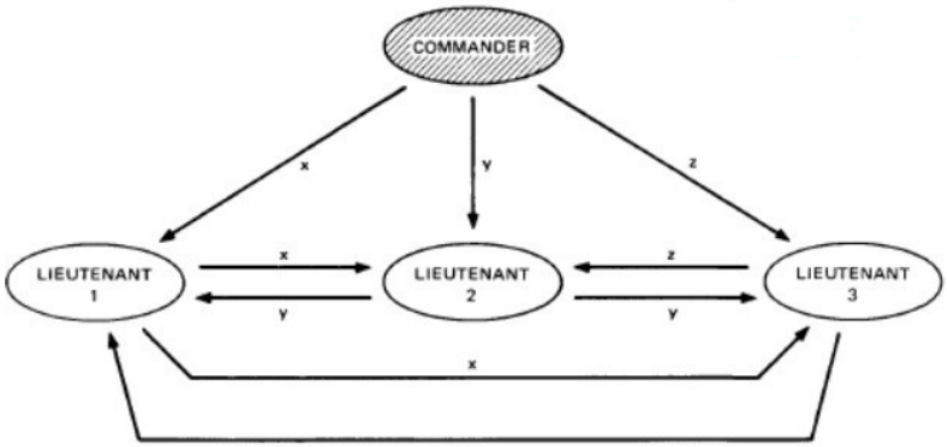

Figura 2.6. Algoritmo OM(1): il comandante C è l'unico traditore.

Con più generali lo schema si complica, ma avendo meno di un terzo di traditori, in ogni caso si può dimostrare che il consenso tra i leali può essere raggiunto. La dimostrazione ha quindi la seguente struttura: si dimostra che sapendo risolvere il problema Commander and Lieutenants è possibile risolvere anche il The Byzantine Generals Problem, e poi si dimostra che il Commander and Lieutenants è risolvibile se i traditori sono non più di un terzo del totale.

Byzantine Fault Tolerance (BFT)

Un sistema dove tutti sono traditori non può chiaramente funzionare, mentre un sistema dove tutti sono leali funziona perfettamente. C'è poi una via di mezzo: l'obbiettivo è capire quanto un sistema può tollerare delle manomissioni o degli errori casuali. La **Byzantine Fault Tolerance** (**BFT**) è la proprietà di un sistema di riuscire a resistere alla classe di fallimenti derivata dal problema dei generali bizantini. Ad esempio il livello di tolleranza del problema Commander and Lieutenants è pari a $\frac{1}{3}$. Costruire un BFT è una necessità, ma anche estremamente difficile in un sistema distribuito. La BFT esiste nei sistemi motore di aerei, missili, centrali nucleari, ossia quei sistemi che devono prendere una decisione in base alle informazioni ricevute da molti sensori diversi che non sempre coincidono.

Problema dei generali bizantini e Blockchain

Nel problema dei generali bizantini, utenti che sono collegati in modo asincrono devono mettersi d'accordo su una data e un'ora. Nel caso di una Blockchain pubblica (permissionless) invece il punto più delicato e attaccabile è l'inserimento di un nuovo blocco, che deve essere fatto raggiungendo il consenso dell'intera rete. Alcuni utenti, si spera pochi, potrebbero essere disonesti e indicare come corretto un blocco che invece non lo è. Va quindi stabilito un algoritmo di consenso (PoW) che gestisca l'esistenza di un numero (non troppo elevato) di utenti disonesti, proprio come nel caso del problema dei generali bizantini.

2.5 Proof of Work (PoW)

La PoW è una dimostrazione che l'utente che ha fatto un certo lavoro può presentare per provare a tutti di aver effettivamente eseguito il lavoro, attestando quindi il suo impegno e in un certo senso anche la sua affidabilità. La forma più semplice di PoW è realizzata tramite una funzione hash e si può descrivere nel seguente modo:

1. Dato un testo qualsiasi, si aggiunge qualche carattere scelto a caso e si calcola la funzione hash.

2. Se il risultato della hash è minore di un certo valore (**target**), la PoW termina e l'hash calcolato può essere usato come prova del lavoro svolto.

3. Se invece il risultato della hash è maggiore del target si riprende dal punto 1.

Scegliendo adeguatamente il target si può rendere la probabilità di avere una hash minore del target della dimensione voluta e quindi si può rendere la PoW più o meno impegnativa. Data la casualità del problema, si può facilmente valutare quanto ci vuole mediamente a produrre una PoW e quindi mostrando il testo e il corrispondete hash (minore del target) si può dimostrare a tutti la quantità di lavoro eseguita. Si può quindi valutare quanto tempo ci mette in media un utente per ottenere un valore minore del target calcolando la probabilità di successo, ossia la probabilità di ottenere una stringa che inizi con un certo numero di zeri. Se la probabilità di successo è p, allora il tempo medio di attesa è $\frac{1}{p}$, che è molto grande se p è molto piccolo. Per la natura stessa delle funzioni hash non è possibile scegliere ad hoc i caratteri da inserire nel testo per rendere piccolo il valore della hash e quindi l'unico modo di procedere è quello di inserire i caratteri in maniera

casuale e riprovare fino a che a forza di ripetere il procedimento non si ottiene una hash minore del target.

2.6 Catena di PoW: The Byzantine Generals Problem (S. Nakamoto)

Satoshi Nakamoto stesso in una sua mail del 2008 fa notare in modo esplicito che una **catena di proof-of-work** può essere usata per trovare una soluzione al problema dei generali bizantini.

Descrizione: Un certo numero di generali bizantini ha ciascuno un computer e vuole attaccare il wi-fi del re attaccando a forza bruta la password. Una volta che inizia l'attacco, devono decifrare la password entro un tempo limitato per entrare e cancellare i log, altrimenti verranno scoperti. Gli attaccanti hanno abbastanza potenza di CPU per decifrarla velocemente solo se la maggior parte della potenza di calcolo a loro disposizione viene utilizzata contemporaneamente. Come possono i generali mettersi d'accordo e attaccare tutti (o almeno abbastanza per avere la maggioranza della potenza di calcolo) contemporaneamente?

Il problema potrebbe sembrare facile, ma così non è. Non basta dire che il primo generale che si decide, indica un orario e gli altri lo seguono, perchè la rete è asincrona e se due generali si esprimono più o meno contemporaneamente, alcuni potrebbero ricevere prima l'orario del primo e altri prima quella del secondo, non trovando il consenso. Serve dunque un algoritmo più elaborato. La brillante soluzione proposta da Nakamoto è la seguente:

- Ogni generale può proporre un suo tempo di attacco.
- Ogni generale quando riceve una proposta di orario comincia una PoW, facendo l'hash di un testo che contiene l'ora di attacco. Il target deve essere tale che tutta la rete dei generali impiega circa 10 minuti a risolvere la PoW.
- Una volta che uno dei generali conclude con successo la sua PoW, trasmette l'hash ottenuto alla rete e tutti cambiano il calcolo della loro PoW includendo tale hash e l'orario vincente nella PoW su cui stanno lavorando.
- Ogni circa 10 minuti la catena di PoW si allunga e tutto è pubblico, e quindi a un certo punto tutti i generali sono consapevoli che c' è un orario di attacco

su cui ha lavorato la maggioranza della potenza di CPU e possono quindi attaccare in sicurezza all'ora così concordata.

2.7 Proof of Stake (PoS)

Il significato della **Proof of Stake** è quello di provare che si ha un interesse. Il principio è quello di far inserire i blocchi tenendo conto dell'ammontare di criptomonete possedute dal **validator** (validatore di blocchi), con l'idea che chi possiede grosse quantità di valuta ha tutto l'interesse a far funzionare il sistema. Dunque è più probabile che inserisca un blocco un utente cha ha tante monete rispetto a uno che ne ha poche. Ovviamente non si può usare semplicemente la quantità di moneta come indicatore per selezionare il validatore che inserirà il blocco seguente, altrimenti tutti i blocchi sarebbero inseriti sempre dalla stessa persona. Le soluzioni adottate dalle diverse Blockchain per tenere conto della quantità di moneta posseduta sono varie:

- **Random**: alcune criptovalute (per esempio Nxt e BlackCoin) utilizzano dei sistemi casuali, dove la maggior quantità di valuta posseduta fa aumentare la probabilità di diventare validatore.
- **Anzianità**: corrisponde alla quantità di monete moltiplicata per il numero di giorni che sono state possedute. L'anzianità torna a zero quando si inserisce un blocco e decade se il tempo del possesso diventa troppo grande. Esempio: Peercoin.
- **Velocità**: per incoraggiare la movimentazione di moneta, conta di più la velocità di utilizzo piuttosto che il suo accumulo. Esempio: Reddcoin.

Vantaggi e svantaggi della PoS rispetto alla PoW

Vantaggi

- Molto minori consumi di energia
- Maggiore velocità
- I miners di una Blockchain PoW sono meno coinvolti rispetto ai validators di una Blockchain PoS

Svantaggi

- Problemi a gestire le fork (**nothing at stake**)

Nothing at stake

Nel caso di una ramificazione della Blockchain (o qualsiasi altro tipo di disaccordo nel consenso), una persona potrebbe votare per entrambe le varianti, perchè ha delle poste in gioco in ciascuna delle varianti. Nel caso di PoW è possibile lavorare su entrambi i rami della fork tuttavia la potenza di calcolo è dimezzata su ciascun ramo e di conseguenza non conviene. Invece nel caso di PoS non costa molto lavorare su entrambi i rami di una fork e ciò dà la possibilità di provare a ingannare (per esempio spendendo la stessa cifra due volte in un'istanza di riorganizzazione della Blockchain). Possibili soluzioni al nothing at stake sono:

- Checkpoint
- Limite sulla riorganizzazione dei blocchi
- Punire chi lavora su due rami della Blockchain (Ethereum ha un apposito sistema: Slasher).

Capitolo 3

Bitcoin

Bitcoin (simbolo ₿) è una criptovaluta creata nel 2009 da un anonimo inventore, noto con lo pseudonimo di Satoshi Nakamoto, che sviluppò un'idea da lui stesso presentata su internet a fine 2008 (Bitcoin A Peer-to-Peer Electronic Cash System-Satoshi Nakamoto (2009)). Non è chiaro se Nakamoto sia (o sia stato) una persona

Figura 3.1. Bitcoin.

o un gruppo di persone, ma molti sospettano abbia a che fare con Nick Szabo, Hal Finney, Wei Dai e/o altri attivisti del movimento Cypherpunk. Bitcoin è una criptomoneta digitale, anonima e distribuita, gestita da una rete **peer-to-peer** di utenti e non fa uso di un ente centrale. Il suo valore è determinato unicamente da

domanda e offerta. Il 3 gennaio del 2009 è stato creato il genesis block di Bitcoin e il 12 Gennaio c'è stata la prima transazione ufficiale, tra Satoshi Nakamoto e Hal Finney di 10 BTC. Bitcoin registra tutte le transazioni tramite una blockchain pubblica, condivisa e gestita direttamente dai nodi di una rete peer-to-peer.

3.1 Blockchain: struttura blocco, nodi, wallet, Merkel tree

I blocchi sono composti di due parti: l'**header** e il **body**. Le transazioni sono racchiuse nel body del blocco, mentre nell'header sono presenti sei campi di gestione del blocco stesso. In generale un blocco presenta la struttura mostrata in figura

Field	Description	Size
Magic no	value always 0xD9B4BEF9	4 bytes
Blocksize	number of bytes following up to end of block	4 bytes
Blockheader	consists of 6 items	80 bytes
Transaction counter	positive integer VI = VarInt	1 - 9 bytes
transactions	the (non empty) list of transactions	<Transaction counter>-many transactions

Figura 3.2. Struttura di un blocco della blockchain di Bitcoin.

3.2. Il magic number di un blocco è sempre D9B4BEF9 (notazione esadecimale) e non ha un significato particolare, se non identificare ciò che segue come un blocco della Blockchain di Bitcoin. Un modo per identificare i blocchi è mediante la loro **altezza**. Con altezza si intende la distanza dal blocco di genesi, che per definizione ha altezza zero. Il primo blocco che è stato minato dopo quello di genesi ha altezza 1, il seguente ha altezza 2 e così via.

Struttura header

L'header di un blocco ha la struttura mostrata in figura 3.3. Il campo **PrevBlock** rappresenta l'**hash dell'header del blocco precedente**. **Timestamp** indica l'ora in cui il blocco è stato creato. **Bits** è il valore corrente del target. Infine **Nonce** è un valore che viene inserito in modo causale finchè la funzione di hash (eseguendo due volte SHA-256) del blocco risulta inferiore al target e solo in tal caso il blocco viene inserito nella Blockchain. Notiamo che nella struttura dell'header non c'è l'hash del blocco precedente ma solo l'hash dell'header del blocco

Field Size	Description	Data type
4	version	int32_t
32	prev_block	char[32]
32	merkle_root	char[32]
4	timestamp	uint32_t
4	bits	uint32_t
4	nonce	uint32_t

Figura 3.3. Struttura header.

precedente. Le hash dei blocchi sono quindi calcolate dai miners, e devono essere inferiori al target affinchè il blocco sia inserito, ma non sono registrate nella blockchain così come le altezze dei blocchi. Tuttavia tutti i dati si possono trovare facilemente (ad esempio su https://www.blockchain.com/explorer).

Latest Blocks

The most recently mined blocks

Height	Mined	Miner	Size
705271	4 minutes	Unknown	1,429,977 bytes
705270	21 minutes	Unknown	1,284,806 bytes
705269	29 minutes	Unknown	1,262,774 bytes
705268	42 minutes	ViaBTC	1,573,703 bytes
705267	55 minutes	Unknown	1,465,630 bytes
705266	1 hour	AntPool	1,456,985 bytes

Figura 3.4. Bitcoin: blocchi minati.

3.1.1 Tipi di nodi

Esistono due tipi di nodi in Bitcoin:

- **Full-node client**
- **Light client**

Attivare un full-node di Bitcoin significa entrare a tutti gli effetti nella community Bitcoin. Ogni full-node memorizza l'intera Blockchain e gestisce tutti gli aspetti

Hash	00000000000000000000037576a828dcabb90ea4f156e5e56dc6fe1b750213ec5
Confirmations	3
Timestamp	2021-10-16 18:34
Height	705271
Miner	Unknown
Number of Transactions	2,845
Difficulty	19,893,045,048,575.13
Merkle root	380cfb04247f139f51251e8f8e6967b829f8cb1a0405675978d9f4ad8990b10b
Version	0x20000004
Bits	386,803,250
Weight	3,993,210 WU
Size	1,429,977 bytes
Nonce	1,965,443,652
Transaction Volume	59731.53676842 BTC
Block Reward	6.25000000 BTC
Fee Reward	0.20303299 BTC

Figura 3.5. Blocco 705271 di Bitcoin.

del protocollo Bitcoin, compresa la verifica di tutte le transazioni che vengono proposte alla rete. Per attivare un full node sono necessari:

- Desktop o laptop con una versione recente di Windows, Mac OS X o Linux
- 2 GB di memoria (RAM)
- 200 GB di spazio libero sul disco
- Connessione a internet ad alta velocità con upload di almeno 50 kB/s
- Connessione illimitata
- Il Full-node deve essere attivo almeno 6 ore al giorno.

Invece un light-client, noto anche come **simple-payment-verification (SPV)**, non memorizza l'intera Blockchain ma solo gli header di tutti i blocchi. Un light-node può creare e inviare transazioni in modo indipendente alla rete. Può inoltre

verificare in modo rapido la correttezza formale di una transazione appoggiandosi a un full-node. Pur non potendo fare una verifica autonoma delle transazioni i light-client interagiscono direttamente con la rete Bitcoin senza intermediari. Non è invece in grado di verificare se chi vuole eseguire un pagamento abbia in effetti ricevuto su quell'indirizzo abbastanza Bitcoin senza averli già spesi (per fare ciò serve l'intera Blockchain).

3.1.2 Wallet

Per poter acquistare e rivendere BTC (così come le altre criptovalute) è necessario avere un software (o hardware) dedicato, detto **Wallet** (Portafoglio). Di tipi di wallet ne esistono tanti e possono essere suddivisi in varie categorie, a seconda della piattaforma utilizzata:

- **Desktop wallet**: sono dei software che si installano sul computer desktop per gestire un portafoglio Bitcoin. In genere ne esistono versioni Windows e Mac OS e hanno grande funzionalità e controllo. Il lato negativo è la dubbia sicurezza, che dipende anche dall'attaccabilità del PC.

- **Mobile wallet**: è il tipo più comune e diffuso di wallet Bitcoin. Si installano sugli smartphone (Android o iOS) e sono molto leggeri e dal semplice utilizzo. Spesso non sono dotati di funzionalità complete come la versione desktop, ma ne esistono anche di adatte a utenti esperti. Come nel caso precedente, la sicurezza dipende dalla sicurezza del supporto.

- **Web wallet**: sono accessibili tramite un browser web e memorizzano le informazioni sulle chiavi delle transazioni su un server di proprietà di terzi. Non è quindi consigliabile utilizzare questo tipo di wallet per grandi quantità di Bitcoin.

- **Hardware wallet**: sono dei dispositivi mobili che si collegano al PC mediante la porta USB e sono degli hardware appositamente progettati per gestire un wallet. Sono considerati molto sicuri e adatti a utilizzare anche grandi quantità di Bitcoin.

- **Paper wallet**: le chiavi segrete e pubbliche e gli indirizzi Bitcoin possono anche essere stampati per la memorizzazione a lungo termine e in tal caso si parla di paper wallet, anche se non è detto che si usi la carta per la stampa. Quest'ultimo è un sistema low-tech ma molto sicuro.

I primi tre sono indicati come **Hot wallet** mentre gli ultimi due come **Cold wallet**. La sicurezza di un wallet, di qualunque tipo si tratti, dipende anche molto dall'utilizzo che ne fa l'utente. Se anche il sistema adottato per gestire il wallet è ben progettato e molto sicuro, rimane sempre il rischio di sicurezza dovuto al fattore umano. Per accedere al wallet vengono utilizzate password e sistemi di verifica spesso a più livelli (autenticazione a più fattori), ma è fondamentale che tali sistemi siano usati in modo corretto. Usare password banali, condividere per sbaglio le chiavi private, perdere l'accesso al wallet, sono errori umani che possono essere a volte fatali e irrecuperabili.

3.1.3 Merkel tree e Merkel root

Le transazioni di un blocco hanno una struttura ad albero come mostrato in figura 3.6. Il **Merkle root** coincide con la sommità del **Merkle tree**. Data la strut-

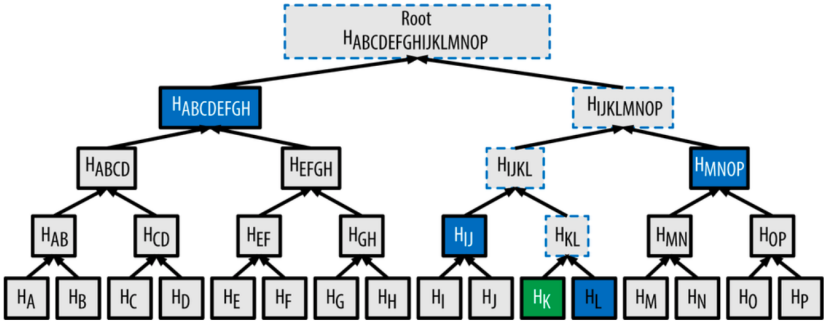

Figura 3.6. Merkle tree e Merkle root

tura ad albero, è possibile verificare che una transazione è presente in un blocco, trasferendo poco dati e in modo molto rapido. Questo perchè per costruzione per fare la verifica sono sufficienti la transazione vicina a quella da verificare e le hash che si trovano sul cammino che dalla transazione da verificare sale fino al Merkel root. Tale cammino ha una lunghezza proporzionale al logaritmo del numero delle transazioni del blocco e quindi l'algoritmo di verifica mediante il Merkle tree è estremamente più efficiente e necessita il trasferimento di molti meno dati rispetto a una verifica sequienziale su tutte le transazioni del blocco. Se ci fosse anche solo una variazione di un bit in una transazione di un blocco, allora cambierebbe il Merkle root del blocco (che si trova nell'header) e quindi cambierebbe di conseguenza l'hash dell'header che si trova nel blocco seguente. Per fare una modifica anche

di un solo bit a una transazione e lasciare la Blockchain consistente, bisognerebbe rifare tutte le PoW da quel blocco in avanti.

3.2 Le transazioni di Bitcoin

In Bitcoin non c'è un concetto di conto, con entrate e uscite, associato a ogni utente. Sarebbe altrimenti pubblico l'intero estratto conto e il saldo di ogni utente e sarebbe fin troppo facile risalire quindi all'identità. La Blockchain di Bitcoin è un enorme elenco di transazioni (passaggi di BTC da un indirizzo a un altro) anonime, dove le transazioni relative a un singolo utente non possono essere facilmente riconosciute e ragruppate. Supponiamo che Alice abbia ricevuto 1 BTC in una precedente transazione e voglia trasferirlo a Bob:

- Alice informa Bob che vuole trasferirgli 1 BTC.
- Bob crea una coppia di chiavi privata/pubblica $(d_B, d_B G)$ mediante la curva ellittica Secp256k1 e calcola l'indirizzo Bitcoin $Address_B$ associato alla chiave pubblica $d_B G$.
- Bob invia ad Alice l'indirizzo Bitcoin $Address_B$ appena creato.
- Alice scrive la transazione fissando $Address_B$ come indirizzo di output e $Address_A$ come indirizzo di input, dove $Address_A$ è un indirizzo su cui Alice ha ricevuto esattamente 1 BTC in una precedente transazione.
- Alice manda a Bob la transazione firmata digitalmente mediante l'Elliptic Curve Digital Signature Algorithm (ECDS).
- Bob verifica la firma digitale e ha la certezza che Alice si è impegnata a pagargli 1 BTC .

Non si utilizza direttamente la chiave pubblica come indirizzo su cui trasferire i BTC, ma si utilizza il cosiddetto **indirizzo Bitcoin**, che è più corto e si ricava dalla chiave pubblica nel seguente modo:

- Si calcola S = (version byte) || RIPEMD-160(SHA-256(K_{pub})).
- Si calcola C (checksum) = primi 4 bytes di SHA-256(SHA-256(S)).
- Si calcola $S_{check} = S \,||\, C$.
- Si ottiene l'indirizzo Bitcoin calcolando la codifica Base58Check di S_{check}.

RIPEMD-160 è una funzione hash, mentre Base58Check è una codifica alfanumerica. Ci sono dei vantaggi a usare l'indirizzo Bitcoin invece che direttamente la chiave pubblica:

- Avendo un check interno, è molto difficile inviare BTC a indirizzi errati.
- Avendo una fase di codifica Base58Check si evitano i classici errori 0O, lI.
- Solo gli indirizzi Bitcoin sono nella Blockchain, non le chiavi pubbliche che invece rimangono segrete fino a che i BTC non vengono nuovamente spesi.
- Grazie alle funzioni hash utilizzate nel protocollo, non è possibile risalire dall'indirizzo Bitcoin alla chiave pubblica.

Dalle ultime due osservazioni segue che se si riceve un pagamento in BTC e poi non li si usa per anni, non si concede comunque tanto tempo ad un attaccante per cercare la chiave privata, in quanto la chiave pubblica non è nella Blockchain, ma solo una sua hash: l'indirizzo Bitcoin. Solo al momento di spendere nuovamente i bitcoin ricevuti, il legittimo proprietario pubblica la chiave pubblica (che chiunque può immediatamente verificare essere la vera chiave pubblica facendone l'hash e ottenendo l'indirizzo Bitcoin che è pubblico sulla Blockchain) e utilizzando la chiave privata dimostra a tutti di essere il legittimo proprietario e può quindi mettere i BTC ricevuti come input di una nuova transazione. C'è anche uno svantaggio a usare l'indirizzo Bitcoin invece che direttamente la chiave pubblica: due chiavi pubbliche diverse potrebbero generare lo stesso indirizzo Bitcoin e quindi anche un utente diverso da quello che si vuole che riceva la transazione potrebbe spendere i relativi Bitcoin. Tuttavia la probabilità che questo accada è veramente trascurabile. D'altra parte c'è un rischio ancora più fondamentale: due utenti diversi potrebbero generare per caso la stessa coppia chiave pubblica/privata e in tal caso potrebbero spendere entrambi gli accrediti ottenuti sull'indirizzo. Sul sito di Bitcoin però si taglia corto su questo rischio: "it is more likely that the Earth is destroyed in the next 5 seconds, than that a collision occur in the next millenium".

Ovviamente, visto il relativamente alto valore del singolo Bitcoin, è possibile fare transazioni anche per una frazione di Bitcoin (fino all'ottava cifra decimale). Per comodità è stato definito il satoshi come un centomilionesimo di Bitcoin, cioè ci vogliono cento milioni di satoshi per fare un BTC. Non si possono fare transazioni per frazioni di satoshi, ma questo non è oggi un limite significativo visto il valore attuale dei BTC. Se anche in futuro i BTC moltiplicassero il loro valore per 1.000, comunque il valore di 1 satoshi sarebbe sempre una frazione di centesimo di euro.

3.2.1 Transazioni con più input e più output

Per semplicità abbiamo preso in considerazione una transazione da Alice a Bob di 1 BTC, supponendo che Alice avesse ricevuto a sua volta esattamente 1 BTC in una precedente transazione. Ovviamente in genere la situazione è più complessa, ma il protocollo di trasferimento che abbiamo presentato non cambia. Infatti basta semplicemente prevedere transazioni con più input e più output. Se Alice deve pagare 1 BTC a Bob, non c'è bisogno che abbia ricevuto in passato una transazione a suo favore di esattamente 1 BTC. Può infatti indicare nella transazione più indirizzi di input di sua proprietà che tutti insieme abbiano ricevuto accrediti per più di 1 BTC. Se la somma di tutti supera 1 BTC, Alice può facilmente gestire l'operazione specificando due indirizzi di output: uno di Bob su cui accreditare 1 BTC e uno proprio su cui accreditare la differenza (il resto).

Fees

La **fee** è l'importo che chi scrive la transizione è disposto a pagare per vedere realizzata la propria operazione. Le fees sono liberamente decise da chi fa la transazione (più è alta e più è probabile che la transazione venga inserita velocemente nella Blockchain). La fee è semplicemente la differenza tra la somma di tutti gli input meno la somma di tutti gli output. Le fees servono a remunerare (ulteriormente) i miners per il loro lavoro, ma anche a rendere resistente la rete a un attacco Denial of Service (inondando la rete con un numero enorme di transazioni).

Numero di conferme di una transazione

Quando una transizione si trova in un blocco validato e quindi inserito nella Blockchain, si dice che la transazione ha una conferma. Quando un successivo blocco viene validato e inserito nella Blockchain, la transazione avrà 2 conferme, e così via.

In generale ci sono due fasi di verifica prima che una transazione diventi effettiva:

- la verifica della transazione
- la validazione del blocco in cui si trova la transazione

3.2.2 Verifica delle transazioni

Bob quando riceve il pagamento da Alice, invia all'intera rete la transazione firmata digitalmente con la chiave privata di Alice e quindi prova a tutti che il

proprietario dell'indirizzo $Address_A$ si è impegnato a pagargli 1 BTC. Nessuno sa che il proprietario dell'indirizzo $Address_A$ è Alice, ovviamente escluso Bob. Inoltre nessuno è in grado di risalire dall'indirizzo $Address_A$ alla chiave pubblica di Alice e tanto meno a quella privata. A questo punto tutti gli utenti, in modo automatico, verificano che sull'indirizzo $Address_A$ sia arrivato in passato un accredito di 1 BTC e soprattutto che tale accredito non sia ancora stato speso. Siccome tutti gli utenti della rete hanno la medesima copia della blockchain e del software di verifica automatica delle transazioni, se non vengono rilevati errori la rete accetta come corretta la transazione che viene considerata verificata. Tuttavia Bob in questa fase non ha ancora la certezza di essere stato pagato da Alice. Questo perchè se arrivati a questo punto tutte le transazioni verificate dalla rete finissero automaticamente in un blocco della Blockchain, ci sarebbe un grosso problema dovuto alla natura asincrona della rete: **problema del double spending**. Se Alice pagasse più o meno contemporaneamente sia Bob che un altro utente, dal momento che entrambe le transazioni non sono ancora definitive e non sono quindi ancora nella Blockchain, la verifica della rete potrebbe andare a buon fine per entrambe e Alice pagherebbe due persone con il medesimo BTC ricevuto. Per questo dopo la fase di verifica della transazione c'è la fase di **validazione dei blocchi**.

3.2.3 Validazione dei blocchi

Questo è il punto più delicato di tutto il protocollo, perchè è qui che si decide la composizione del nuovo blocco da inserire nella Blockchain e quindi quali transazioni diventano definitive. Il primo passo di questa fase è quello di mettere automaticamente tutte le transazioni verificate nel passo precedente in un archivio delle transazioni. A questo punto entrano in azione i **Miners**. Ogni miner prende un certo numero di transazioni dall'archivio delle transazioni verificate dalla rete e crea un possibile nuovo blocco. In Bitcoin c 'è un vincolo alla dimensione massima di ogni blocco (1 MB) e quindi i miners non scelgono a caso le transazioni da inserire nel blocco, ma decidono in base alle fees delle transazioni. Se il blocco sarà validato ed entrerà in modo definitivo nella Blockchain il miner guadagna, oltre a una ricompensa per aver inserito il blocco, anche le fees di tutte le transazioni del blocco. Notiamo che le fees sono decise da chi fa le transazioni e quindi chi vuole che la sua transazione sia inserita al più presto nella Blockchain sarà disposto a pagare fees più alte, e rendere quindi la propria transazione più allettante per i miners. Altrimenti può proporre fees più basse rassegnandosi a vedersi passare davanti chi propone fees alte.

3.3 Protocollo di consenso: Proof of Work

Ogni miners, scelte le transazioni validate da inserire nel body del blocco, calcola la funzione di hash (ripetendo due volte di seguito SHA-256) dell'intero blocco, facendo variare il Nonce, fino a che l'output della hash non è inferiore al target. Il primo nodo che risolve il blocco, cioè che riesce ad avere un hash minore del target, lo trasmette alla rete dove viene accettato come blocco successivo nella catena. Una volta che il blocco è stato risolto, l'hash del suo header, come un'impronta digitale, lo rappresenta univocamente e sarà usato come riferimento dal blocco seguente. La rete periodicamente aggiusta in modo automatico il valore del target in modo da mantenere sotto controllo la velocità di inserimento di nuovi blocchi. Questo processo è la cosiddetta Proof of Work (PoW) ed è il protocollo di consenso per l'inserimento dei nuovi blocchi di Bitcoin. Il motivo di questa costruzione (molto costosa dal punto di vista dell'energia impiegata e dei grandi computer necessari) è che se si decidesse quale blocco accettare semplicemente a maggioranza degli utenti, a un attaccante basterebbe registrarsi più volte creando delle identità fittizie (dette identità Sybil) per controllare la rete (il cossiddetto Sybil attack). Quando una transazione si trova in un blocco validato e quindi inserito nella Blockchain, la transazione si dice che ha una conferma. Quando un successivo blocco viene vaidato e inserito nella Blockchain, la transazione avrà 2 conferme, e così via.

3.3.1 Fork

È possibile che diversi nodi concludano la validazione di un blocco più o meno contemporaneamente. In tal caso non si possono inserire entrambi, perchè inserito uno, l'altro non è più un blocco valido non contenendo l'hash dell'header dell'ultimo blocco inserito. Non è operativamente fattibile accettare semplicemente chi arriva una frazione di secondo prima, a causa della asincronia della rete. Se due blocchi sono minati a una certa distanza temporale (il secondo è minato quando tutta la rete ha già ricevuto il primo), il secondo verrà rifiutato come blocco non valido (in quanto non contiene l'hash dell'ultimo blocco ma del penultimo). Tuttavia se i due blocchi sono minati in tempi più ravvicinati si crea una biforcazione (**fork**) della catena. Quando si crea una fork, entrambi i blocchi sono considerati momentaneamente come possibili nuovi blocchi e i miners sono invitati a continuare a minare su uno qualsiasi dei due rami. Appena in una delle due biforcazioni viene validato ed aggiunto un nuovo blocco, i miners tendono a spostarsi sulla biforcazione più lunga, che ha maggiore probabilità di diventare quella definitiva. Siccome le transazioni contenute nei blocchi abbandonati (orfani) non saranno mai

validate, i miners non prenderanno la ricompensa per aver minato tali blocchi nè le relative fees. Per questo i miners tendono fortemente a spostarsi sulla biforcazione più lunga e le biforcazioni tendono a risolversi velocemente. Nella storia di Bitcoin non c'è mai stata una biforcazione più lunga di 6 blocchi e quindi mai una transazione con 6 conferme è stata annullata. Bitcoin utilizza inoltre dei check-

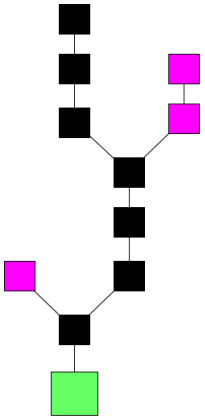

Figura 3.7. Blocchi della catena principale (blocchi neri), con il blocco di genesi (blocco verde) e i blocchi orfani (blocchi viola).

point (dopo un checkpoint è impossibile che una transazione venga annullata) per rendere la struttura della Blockchain inalterabile fino a quel punto.

Soft fork e Hard fork

Oltre alle usuali fork che sono causate (involontariamente) dalla contemporanea validazione di più blocchi da parte di diversi miners, ci sono anche altri motivi che possono portare a una fork della Blockchain. Per esempio una parte degli utenti della Blockchain possono proporre di modificare il protocollo, in qualche punto minore o anche in modo molto significativo. Se la proposta viene accettata e condivisa da tutta la rete non si crea nessuna fork. Tutti passano alla nuova versione del software e il sistema continua a funzionare con le nuove regole. Essendo però la rete di Bitcoin molto vasta è difficile che ci sia un accordo totale sul cambiamento, soprattutto se è significativo, e quindi si possono creare due gruppi contrapposti: i favorevoli al cambiamento e quelli che vogliono mantenere le vecchie regole. Se gli innovatori non intendo rinunciare, hanno due modi di procedere:

- **Soft fork** : adottare un protocollo che permetta a chi mantiene il vecchio software di continuare a operare regolarmente (e quindi le transazioni eseguite

con il nuovo protocollo devono essere riconosciute come valide anche da chi mantiene il vecchio software).

- **Hard fork** : adottare un protocollo non compatibile con quello vecchio.

Sostanzialmente una soft fork non è una vera fork, nel senso che non causa una biforcazione della Blockchain, che rimane unica. Dopo una hard fork invece si hanno due distinte Blockchain che continuano a svilupparsi autonomamente e che condividono però tutti i blocchi fino al momento della fork.

3.3.2 Velocità di validazione dei blocchi

Uno dei problemi tecnici di Bitcoin è la necessità di tenere basso il numero dei blocchi inseriti nella Blockchain e proprio per tale fine il target viene aggiornato con regolarità. Il motivo di questa scelta è che se i blocchi venissero inseriti troppo velocemente si potrebbero creare delle lunghe biforcazioni, che potrebbero durare anche molto tempo, e si allungherebbero molto i tempi per avere la certezza che la propria transazione non finisca in un blocco orfano e venga quindi annullata. Il target viene aggiornato per fare in modo che ogni blocco sia inserito in media ogni 10 minuti. Il target viene aggiornato ogni 2016 blocchi (circa 2 settimane):

- se i miners hanno impiegato mediamente per minare i 2016 blocchi meno di 10 minuti a blocco, il target scende
- altrimenti il target sale

3.3.3 Difficulty

Il rapporto tra il target iniziale e quello attuale viene chiamato **difficulty**. La difficulty è una misura della difficoltà di trovare un blocco soluzione per un dato valore del target. Essa rappresenta il numero medio di hash calcolate globalmente dalla rete per trovare un possibile blocco soluzione. Ogni criptovaluta ha un tempo medio di ricerca del blocco successivo predefinito. Se il numero di miner aumenta, l'hashrate della rete aumenta. In questo modo il tempo di ricerca del blocco successivo diventa inferiore al valore preimpostato. Di conseguenza, la rete aumenta gradualmente la difficulty. La rete continuerà ad aumentare la difficulty fino a quando il tempo di ricerca del blocco non raggiunge il valore predefinito. Stessa cosa quando il numero di miners diminuisce. In questo caso l'hashrate della rete diminuisce e di conseguenza i miners hanno bisogno di più tempo per trovare un blocco. Quindi la rete riduce la dfficulty rendendo il problema matematico più

semplice da risolvere. La difficulty e l'hashrate sono strettamente correlati. Se si divide la difficulty per l'hashrate di rete, si ottiene il tempo medio di ricerca del blocco per la rete. I blocchi di Bitcoin hanno una dimensione massima di 1 MB e ogni blocco contiene massimo 4.000 transazioni, ma mediamente sono circa 2000/3000. Siccome viene inserito un nuovo blocco ogni 10 minuti circa, la rete di Bitcoin riesce a gestire massimo 7 transazioni al secondo, che mediamente sono solo 3 o 4 al secondo. Per fare un confronto, Visa ha una velocità di punta di circa 24.000 operazioni al secondo e PayPal di quasi 200. Questo è uno dei motivi per cui senza cambiamenti sostanziali Bitcoin non potrà diventare la nuova moneta digitale a livello mondiale.

3.4 Creazione dei Bitcoin

In ogni blocco ci sono tante transazioni, che fanno passare la proprietà dei BTC da un utente all'altro, ma c'è una transazione particolare, la prima del blocco, detta **coinbase transaction**, che è diversa da tutte le altre. La coinbase transaction ha come output l'indirizzo del miner (per un totale di BTC stabilito dal protocollo) e ha come input un indirizzo speciale, detto **coinbase**. Questo è il modo con cui vengono creati i Bitcoin. Il campo coinbase ha assunto con il tempo anche un ulteriore utilizzo. La difficulty è cresciuta talmente tanto che ormai il campo Nonce rischia di non essere più sufficiente a risolvere la PoW e per questo i miners hanno dovuto trovare altre strade. Il nonce ha solo 32 bits e quindi variandolo permette di fare solo 4 miliardi di tentativi. Una soluzione potrebbe essere quella di cambiare l'ordine delle transazioni, o riscriverle in modo equivalente ma leggermente diverso. La soluzione più semplice che adottano i miners è quella di usare lo spazio previsto per la coinbase transaction, che da protocollo può variare da 2 a 100 byte di dati, che viene quindi usato come una sorta di extra nonce. Inizialmente la ricompensa per il miner stabilita da protocollo nella coinbase transaction era di 50 BTC. Il protocollo prevede che tale ricompensa sia dimezzata ogni 210.000 blocchi (essendo mediamente minato un blocco ogni 10 minuti si tratta di circa 4 anni). Il sistema è stato quindi definito in modo che la disponibilità di nuove monete cresca come una serie geometrica ogni circa 4 anni. Nei primi 4 anni (2009-2012) è stata generata metà delle possibili monete, nel seguente quadriennio (2013-2016) si è arrivati ai tre quarti e nel 2020 Bitcoin è arrivato a superare i 7/8 dei Bitcoin totali. Parallelamente il compenso per l'inserimento dei blocchi è stato progressivamente dimezzato: nei primi 4 anni (2009-2012) il compenso per ogni blocco era 50 BTC, nel seguente quadriennio (2013-2016) 25 BTC e nel quadriennio (2017-2020) 12.5

BTC. L'11 Maggio 2020 alle 21:23 (ora italiana) c'è stato l'ultimo dimezzamento e da allora il compenso è di 6.25 BTC. Il dimezzamento del compenso viene chiamato **Halving**.

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... = \sum_{k=1}^{+\infty} \left(\frac{1}{2}\right)^k = 1 \tag{3.1}$$

Essendo la serie geometrica convergente, il numero di Bitcoin tenderà asintoticamente a un valore totale di 21 milioni (50 Bitcoin a blocco per 210.000 blocchi, nei primi 4 anni sono stati creati 10 milioni e mezzo di Bitcoin che per costruzione sono la metà del totale). La serie geometrica è comunque solo un'approssimazione di quanto capiterà al sistema, perchè a un certo punto il compenso, a furia di essere dimezzato, diventerà inferiore a 1 satoshi e quindi non potrà più essere pagato. È facile verificare che questo capiterà a Gennaio del 2140, data al di là della quale i miners non avranno più nessuna ricompensa per i blocchi minati. Ovviamente già molto prima di questa data il compenso dei miners tenderà ad essere molto piccolo e quindi l'incentivo a fare il miners scemerà. Servirà comunque avere un qualche incentivo a fare i miners e il più ovvio potrebbe essere un progressivo aumento delle fees.

Difetti di Bitcoin

I difetti più gravi di Bitcoin sono:

- il costo economico e ambientale della sua PoW
- la lentezza nella gestione delle transazioni
- la scarsa scalabilità
- la tendenza alla nascita di mining pools che rischiano di mettere la validazione dei blocchi in mano a pochi soggetti
- la visibilità di tutte le transazioni e il conseguente relativo rischio per la privacy
- commissioni piuttosto elevate per far validare in tempi brevi le transazioni (per lo meno nei periodi di alto valore del BTC rispetto alle monete fiat)
- difficoltà nel far evolvere il protocollo in modo condiviso

Capitolo 4

Ethereum

Ethereum è stato fondato nel 2015 da Vitalik Buterin. Si tratta di una piattaforma globale, open source, pubblica e basata su una Blockchain. La caratteristica

Figura 4.1. Ethereum.

principale di Ethereum è quella di essere una piattaforma decentralizzata del Web 3.0 utile alla creazione e pubblicazione, su una rete di nodi peer-to-peer, di applicazioni decentralizzate (**dApps**) dette **Smart Contract**, create in un linguaggio di programmazione Turing-completo. Gli smart contract aspirano ad assicurare una sicurezza e una velocità di esecuzione superiore alla contrattualistica esistente

e di ridurre i costi di transazione associati all'intermediazione. Ethereum utilizza la Blockchain per salvare i cambiamenti di stato del sistema, insieme ad una criptomoneta chiamata **Ether** (ETH), utilizzata principalmente per comprare il **gas**, utilizzato per vincolare i costi di esecuzione di uno smart contract. Gli ETH possono essere utilizzati per le transazioni economiche e per fare trading, ma sono soprattutto uno strumento per gestire un computer globale, le cui applicazioni una volta lanciate non possono essere più bloccate. A differenza di Bitcoin, quindi la criptomoneta non viene principalmente usata per eseguire pagamenti, ma svolge una funzione di utilità per la Blockchain stessa. Dal momento che Ethereum non ha una supply, per combattere il problema dell'inflazione i creatori hanno pensato a un sistema simile a quello di Bitcoin: diminuire il premio progressivamente. Se nel 2016 il premio per ogni blocco minato era di 5 ETH, oggi il premio ammonta a 2 ETH. Gli utenti di Ethereum possono essere pseudoanonimi, tuttavia i loro indirizzi lasciano una traccia pubblicamente visibile sulla Blockchain. Gli indirizzi Ethereum sono infatti identificatori univoci derivati da chiavi pubbliche o contratti ottenuti utilizzando la funzione hash unidirezionale Keccak-256. La curva utilizzata da Ethereum per generare gli indirizzi è la cosiddetta secp256k1 (la stessa utilizzata da Bitcoin).

4.1 Chiavi private

Le chiavi private non sono utilizzate direttamente in Ethereum in alcun modo; esse non vengono mai trasmesse o archiviate sulla piattaforma. Una chiave privata è rappresentata da un numero scelto in maniera casuale. La proprietà e il controllo della chiave privata sono alla base del controllo di tutti i fondi associati all'indirizzo Ethereum corrispondente: perdere la chiave privata significa perdere per sempre i fondi relativi all'indirizzo Ethereum associato a quella chiave privata. La chiave privata viene utilizzata sia per creare la firma necessaria per spendere gli ETH, sia per dimostrare la proprietà dei fondi in una transazione. Un semplice modo per generare una chiave privata in maniera random è il seguente:

- Alle due facce di una moneta si associano rispettivamente i valori 0 e 1.
- Si lancia la moneta 256 volte.
- Si ricava la corrispettiva stringa binaria di 256 bits. Una volta ottenuta la chiave privata, quest'ultima può essere utilizzata in un Ethereum wallet. La chiave pubblica e l'indirizzo sono generati a partire dalla chiave privata.

4.1.1 Generazione di una chiave privata a partire da un numero random

Il primo e più importante passo nella generazione delle chavi è trovare una fonte sicura di entropia o casualità. Creare una chiave privata di Ethereum implica la scelta di un numero intero compreso tra 1 e 2^{256}. Il metodo utilizzato per scegliere il numero non è rilevante, fintanto che non è prevedibile o deterministico. Ethereum utilizza un generatore di numeri casuali per produrre 256 bits casuali. Solitamente il generatore di numeri casuali è inizializzato con una fonte di casualità umana, motivo per il quale il sistema potrebbe chiedere di muovere il mouse per alcuni secondi o premere dei tasti casuali sulla tastiera. Un'alternativa valida potrebbe essere la rilevazione delle radiazioni sul canale del microfono del computer. Più precisamente, una chiave privata può essere qualsiasi numero diverso da zero fino a un numero molto grande leggermente inferiore a 2^{256}: un enorme numero di 78 cifre, circa $1.158 \cdot 10^{77}$ Il numero esatto, che indichiamo con N, condivide le prime 38 cifre con 2^{256} ed è definito come l'ordine della curva ellittica utilizzata da Ethereum (secp256k1). Per creare una chiave privata, si sceglie casualmente un numero di 256 bits e si verifica (tramite la conversione binario-decimale) che sia compreso tra 1 e N. In termini di programmazione, il numero associato alla chiave privata si può ottenere a partire da una stringa ancora più grande di bits casuali (raccolti da una fonte di casualità crittograficamente sicura) di cui si calcola l'hash tramite l'algoritmo Keccak-256 che produce quindi un numero di 256 bits. Se il risultato rientra nell'intervallo valido, si ottiene una chiave privata adatta. Altrimenti, si riparte da una nuova stringa di bits casuale fino a quando non si ottiene un risultato valido. A causa dell'alta cardinalità dello spazio delle chiavi private di Ethereum, scegliendo una chiave privata in maniera puramente casuale è altamente improbabile che qualcuno possa mai indovinarla o scegliere la stessa chiave. Il processo di generazione della chiave privata è offline; esso non richiede alcun tipo di comunicazione con il network di Ethereum. Utilizzare un cattivo generatore di numeri causali (come la funzione rand pseudocasuale utilizzata nella maggior parte dei linguaggi di programmazione) è sicuramente una pessima scelta. Per tale motivo, per scegliere un numero che nessun altro sceglierà mai, è necessario che questo sia scelto in maniera veramente casuale. E' fondamentale utilizzare un generatore di numeri pseudo-casuali crittograficamente sicuro (come il CSPRNG) con un seme ottenuto da una fonte di sufficiente entropia. La corretta implementazione dello CSPRNG è fondamentale per la sicurezza delle chiavi private.

4.2 Chiavi pubbliche

Una chiave pubblica di Ethereum è un punto della curva ellittica secp256k1, ed è quindi costituita da una coppia di coordinate (x, y) che soddisfano l'equazione della curva. La chiave pubblica è calcolata a partire dalla chiave privata utilizzando la moltiplicazione su curve ellittiche:

$$K = k\,G$$

dove k è la chiave privata, G è il generatore della curva e K è la chiave pubblica risultante. Il generatore G è specificato dallo standard secp256k1. La moltiplicazione di k con G è equivalente a $G + G + ... + G$, (k volte). Dunque per produrre una chiave pubblica K da una chiave privata k, basta sommare il generatore G a se stesso k volte. Poichè il generatore è sempre lo stesso per tutti gli utenti di Ethereum, una chiave privata k moltiplicata per G darà sempre come risultato la stessa chiave pubblica K. La relazione tra k e K è fissata, ma può essere solo calcolata in una direzione, da k a K. Questo è il motivo per cui un indirizzo Ethereum (derivato da K) può essere condiviso con chiunque senza tuttavia rivelare la chiave privata k dell'utente. Risalire dalla chiave pubblica alla chiave privata significa saper risolvere il problema del logaritmo discreto su curve ellittiche (ECDLP), problema attualmente non risolvibile computazionalmente. Nella maggior parte dei wallet le chiavi privata e pubblica vengono archiviate insieme per comodità. Tuttavia, la chiave pubblica può essere banalmente calcolata da quella privata, quindi è anche possibile memorizzare solo la chiave privata. In Ethereum le chiavi pubbliche sono solitamente rappresentate da una stringa di 130 caratteri esadecimali (65 bytes). Tale formato è stato proposto dal consorzio di industrie **Standards for Efficient Cryptography (SEC1)**. Lo standard definisce quattro possibili prefissi che possono essere utilizzati per identificare i punti sulla curva ellittica. Ethereum utilizza chiavi pubbliche uncompressed, di conseguenza l'unico prefisso esadecimale rilevante è 04. Lo standard prevede la concatenazione delle coordinate x e y della chiave pubblica:

$$04 + x - \text{coordinate(32bytes/64hex)} + y - \text{coordinate(32bytes/64hex)}$$

4.3 Indirizzi

Vogliamo ora analizzare la produzione di un indirizzo a partire da una chiave pubblica. In Ethereum esistono due tipi di indirizzi:

Prefix	Meaning	Length (bytes counting prefix)
`0x00`	Point at infinity	1
`0x04`	Uncompressed point	65
`0x02`	Compressed point with even `y`	33
`0x03`	Compressed point with odd `y`	33

Figura 4.2. Identificazione di un punto sulla curva ellittica a partire dal prefisso.

- **Externally Owned Account**
- **Contract accounts**

4.3.1 Externally Owned Account

Un indirizzo Ethereum di tipo **EOA** si costruisce nel seguente modo:

- Ogni utente sceglie la propria chiave privata

  k = f8f8a2f43c8376ccb0871305060d7b27b0554d2cc72bccf41b2705608452f315

- Si deriva la corrispondente chiave pubblica K e si concatenano la coordinata x e la coordinata y scritte in esadecimale

  K = 6e145ccef1033dea239875dd00dfb4fee6e3348b84985c92f103444683bae07b83b5c38e5e...

- Si utilizza Keccak-256 per calcolare l'hash della chiave pubblica

  Keccak256(K) = 2a5bc342ed616b5ba5732269001d3f1ef827552ae1114027bd3ecf1f086ba0f9

- Si tengono solo gli ultimi 20 bytes (quelli meno significativi)

  001d3f1ef827552ae1114027bd3ecf1f086ba0f9

Nella maggior parte dei casi viene aggiunto il prefisso 0x che indica che l'indirizzo è codificato in esadecimale:

0x001d3f1ef827552ae1114027bd3ecf1f086ba0f9

4.3.2 Contract accounts

I **Contract Account** (**CA**) rappresentano degli smart contracts. Ad ogni CA viene associato un indirizzo, che differentemente dagli EOA non è derivato da una chiave privata. Un contract address è calcolato deterministicamente dall'indirizzo del suo creatore e in base al numero di transazioni completate che ha inviato da quell'indirizzo al momento del deployment del contratto.

Esempio

- Dall'indirizzo 0x220a530fBBfE397C9F95279117fEf25e4490dA90, con **output transaction count** (anche chiamato **nonce**) uguale a 2, viene fatto deployment di un contratto

- L'indirizzo e il nonce vengono concatenati e ne viene fatta la codifica rlp e in esadecimale il risultato è

  $\text{rlp}_{\text{input}}$ = d694220a530fbbfe397c9f95279117fef25e4490da9002

- Come per gli EOA, anche se con input diverso, viene calcolato l'hash

  Keccak256($\text{rlp}_{\text{input}}$) = b6fb3c9e1165999e4de35978d196e1105e638d71ea0a03f902ccd...

Osserviamo inoltre che:

- Il nonce è un valore scalare uguale al numero di transazioni inviate dall'indirizzo. Il nonce è un attributo dell'indirizzo mittente e ha senso solo in tale contesto. Esso non è memorizzato esplicitamente nella Blockchain, ma è calcolato dinamicamente contando il numero di transazioni confermate che hanno origine da un indirizzo.

- Anche i contratti hanno i nonces. Il nonce di un contratto viene incrementato solo quando da esso viene generato un altro contratto. Quando viene invocata una funzione del contratto, il nonce non viene incrementato, in quanto un CA è soltanto il destinatario di una transazione.

- Dall'implementazione di EIP-161 il nonce di un contract address che è stato appena creato è inizializzato ad 1. Nei contratti rilasciati prima di questo standard il nonce era inizializzato a 0.

4.4 Transazioni in Ethereum

Ogni volta che si invia una transazione, vanno specificati due campi: **GasPrice** e **GasLimit**. Nel campo GasPrice specifichiamo quanto siamo intenzionati a spendere in Ether per ogni unità di gas. Il prezzo viene misurato in wei per unità di gas. Il prezzo del gas può essere cambiato a piacimento dall'utente: più è alto, più è probabile che la sua transazione venga inserita prima in un blocco. Nel campo GasLimit specifichiamo invece il massimo nuemero di unità di gas che siamo disposti a spendere per una certa transazione. Ogni riga di codice di uno smart contract che viene eseguita ha un costo in gas. Per ogni transazione deve essere specificato il limite massimo di gas che si vuole utilizzare e a questo punto si possono avere due scenari:

- se la transazione giunge al termine non utilizzando tutto il gas allora va a buon fine e il gas in eccesso viene resituito al mittente sotto forma di Ether
- se invece la transazione non è ancora giunta al termine ma è stato consumato tutto il gas, allora la transazione viene annullata e gli Ether utilizzati per comprare il gas non vengono restituiti al mittente.

4.5 The Merge

Il 6 settembre 2022 la Blockchain di Ethereum ha realizzato l'update per la transizione dalla Proof of Work alla Proof of Stake, il cosiddetto **The Merge**. Questa transizione ha quindi cambiato il modo in cui blocchi vengono validati all'interno della Blockchain. Ethereum è attualmente la Blockchain con il più grande numero di applicazioni e utilizzi esistente. Si tratta della blockchain sulla quale vengono scambiati il maggior numero di NFT e su cui vengono eseguiti il maggior numero di Smart contract ogni giorno a livello globale. Ethereum ha però un problema ossia gli ingenti consumi energetici. A differenza del sistema precedente la PoS apporterà notevoli migliorie, inclusa l'efficienza energetica della Blockchain. Un aspetto altrettanto rilevante risiede nel volume di transazioni che la Blockchain è in grado di gestire. Attualmente, Ethereum gode di una soglia massima di circa trenta transazioni al secondo (30 TPS). La **Ethereum Foundation** sostiene che il Merge ridurrà i consumi energetici del network del 99,95 %, e porrà le basi per futuri miglioramenti della scalabilità. Ciò significa che l'attività del nodo potrà essere svolta più facilmente, anche da tablet o smartphone aumentando la decentralizzazione della rete. Vitalik Buterin ha affermato che Ethereum sarà in grado

di elaborare 100.000 transazioni al secondo dopo il completamento di cinque fasi chiave:

- The Merge
- The Surge
- The Verge
- The Purge
- The Splurge

Il 15 settembre alle 6:42:42 UTC è stata azionata la c.d. difficulty bomb. Quest'ultima si riferisce a un aggiornamento insito nel protocollo che rende esponenzialmente complessi i calcoli richiesti dalla PoW, con l'effetto di rendere poco redditizio il mining e indurre i nodi a migrare verso l'algoritmo di consenso alternativo. Una diretta conseguenza è stato il crollo dell'hashrate di Ethereum. Ci si aspettava

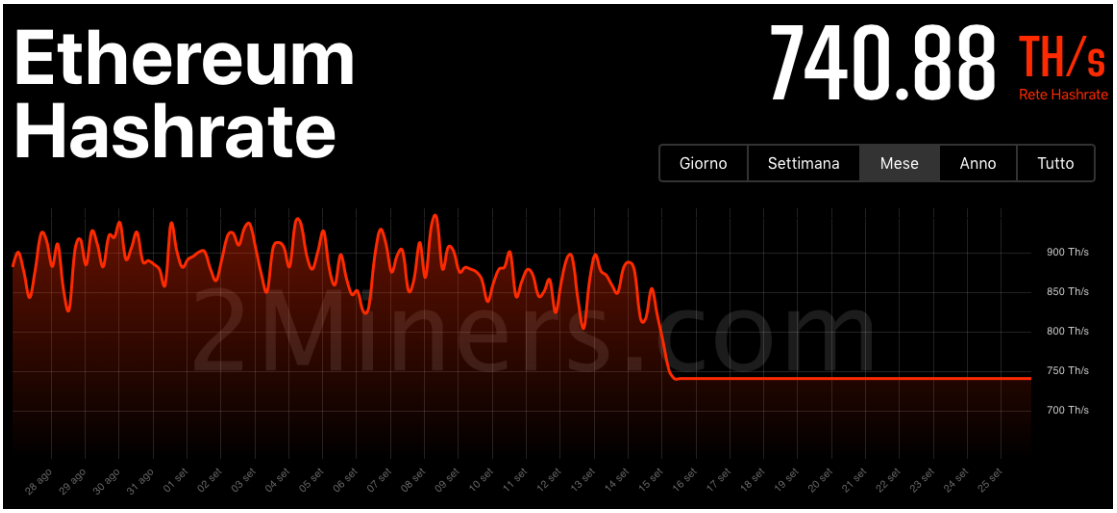

Figura 4.3. Crollo dell'hashrate di Ethereum.

una diminuizione anche un po prima, ma i miner devono aver spremuto le loro macchine fino all'ultimo per minare ETH finchè era possibile. Ora non si può più e dunque hanno dovuto spegnere i macchinari. Ciò porterà ad una probabile fork: ETH e ETHPoW. Inoltre secondo alcune stime, il solo mining di ETH consumava circa lo 0,2 % di tutta l'elettricità prodotta nel mondo, e visto che il mining di ETH ora praticamente non esiste più, quel consumo è già stato azzerato.

Capitolo 5

Mining Pool, GPU, Asics, Rig

Per scegliere il giusto mining pool ci sono alcuni aspetti da considerare. Prima di tutto bisogna valutare quale tipo di software o hardware richiede il pool. Inoltre più grande è il pool, più stabile sarà il guadagno: un pool più grande sarà infatti in grado di inserire più blocchi. Allo stesso tempo però la Blockchain preferisce maggiore decentralizzazione. Bisogna dunque trovare la giusta via di mezzo tra grandezza, stabilità del pool e possibilità di guadagno.

5.1 Pools

I **Mining Pools** sono gruppi di miners cooperanti che accettano di condividere i premi ottenuti in proporzione al loro potere di hash che hanno messo a disposizione. I mining pool sono nati nel momento in cui la difficulty è aumentata al punto che i singoli miner avrebbero dovuto aspettare secoli prima di riuscire a inserire un blocco nella catena. La soluzione a questo problema era che i minatori dovevano unire le loro risorse in modo da poter inserire blocchi più rapidamente, ricevendo una parte della ricompensa del blocco proporzionale alla potenza di calcolo da loro messi a disposizione per il pool. Far parte di un pool è infatti molto positivo per un singolo miner, perchè ottiene remunerazioni più uniformi e prevedibili.

Supponiamo che un miner acquisti un Asic, ad esempio Bitmain Antminer S19j Pro che ha un hashrate di $100\,Th/s$. Consideriamo un hashrate di Bitcoin di circa $180\,Eh/s$. Allora la probabilità che ha un miner di inserire un blocco ogni 4 anni

è

$$P = \frac{100 \cdot 10^{12}}{180 \cdot 10^{18}} 210240 \approx 0{,}12$$

dove 2102140 è il numero medio di blocchi minati in 4 anni. Il miner avrà dunque una probabilità del 12% di inserire un blocco ogni 4 anni. Tuttavia la possibilità di inserire un blocco in un periodo di 4 anni dipende dalla fortuna del minatore (potrebbe riuscire a inserire un blocco e realizzare un profitto molto grande, così come potrebbe non riuscirci). A ciò si aggiunge il fatto che l'hashrate della rete potrebbe aumentare in maniera significativa nel corso del tempo, di conseguenza l'hardware comprato inizialmente potrebbe risultare non competitivo (ciò comporterebbe la sua sostituzione con un hardware più potente). I pagamenti regolari di un pool invece aiutano il singolo miner ad ammortizzare il costo dell'hardware e del consumo elettrico senza correre un enorme rischio finanziario.

L'aumento dei mining pool sta creando un livello di aggregazione che va contro la filosofia e la forza della Blockchain, che è la decentralizzazione. C'è infatti il rischio che se la concentrazione dovesse aumentare, per esempio con la fusione dei mining pools esistenti, o con la diminuzione del numero e la crescita di dimensione di quelli che rimangono, il controllo del protocollo di consenso per l'inserimento dei nuovi blocchi finisca in poche mani, con il conseguente rischio per la sicurezza. Il primo mining pool ad essere stato creato è stato SlushPool. L'intenzione del suo creatore era di unire le forze dei miner dalle scarse prestazioni computazionali. Il risultato di questa interessante visione è stato sorprendente consentendo a questi miner di realizzare profitti migliori in gruppo piuttosto che individualmente.
Un pool funziona essenzialmente come coordinatore per i membri del pool attraverso protocolli specializzati. I singoli miner configurano le proprie apparecchiature di mining connettendosi a un server del pool, dopo aver creato un account. Il compenso ricavato dai blocchi vincenti viene inviato a un indirizzo del pool, piuttosto che ai singoli miner. Il server del pool eseguirà periodicamente i pagamenti agli indirizzi dei singoli miner, una volta che la loro quota dei premi abbia raggiunto una determinata soglia. In genere un pool nomina infatti dei coordinatore incaricati di organizzare i miner. Queste figure si assicurano che i miner utilizzino valori differenti di nonce, in modo da non sprecare hash power creando gli stessi blocchi. Le funzioni da loro svolte prevedono la gestione della fase di inserimento dei blocchi, la registrazione del lavoro svolto da ciascun membro del pool e l'assegnazione di quote di ricompensa a ciascun membro del pool in proporzione al lavoro svolto ossia alla potenza di calcolo messa a disposizione. Il pool può anche addebitare una commissione a ciascun membro in merito al servizio fornito. Il

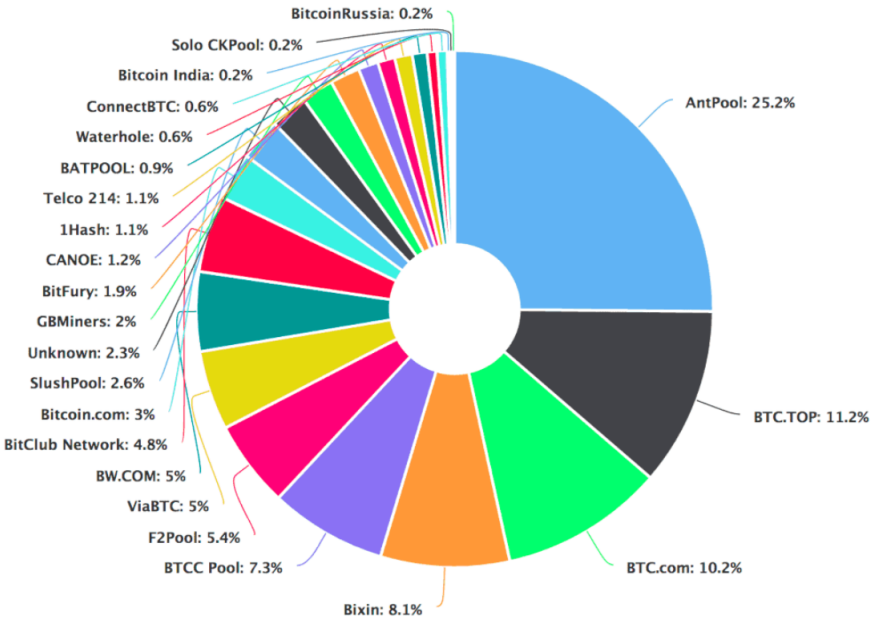

Figura 5.1. Partizione della potenza di calcolo totale tra i più importanti pool.

lavoro a ciascun membro del pool può essere assegnato in due modi. Il metodo tradizionale prevede l'assegnazione ai membri di un'unità di lavoro composta da un particolare intervallo di nonce. Una volta che il membro del pool ha completato il lavoro sull'intervallo assegnato, invia una richiesta per l'assegnazione di una nuova unità di lavoro. Un secondo metodo consente ai membri del pool di scegliere liberamente tutto il lavoro che desiderano senza alcun incarico proveniente dal pool. La metodologia garantisce che nessun membro lavori sullo stesso range di nonce. Proviamo ora a calcolare il numero medio di blocchi minati al giorno da un pool. Sia H_{Eth} l'hashrate di Ethereum e H_{Pool} l'hashrate del pool. Sia inoltre A_b l'average block time di Ethereum, N_{Eth} il numero medio di blocchi minati in un giorno dall'intere rete e N_{Pool} il numero medio di blocchi minati in un giorno dal Pool.

- Considerando che in media ogni A_b secondi viene minato un blocco, il numero medio di blocchi minati in un giorno dall'intera rete sarà dato da:

$$N_{Eth} = \frac{1}{A_b} * 3600 * 24$$

- Dividiamo l' hashrate del pool per l'hashrate di Ethereum e indichiamo con

p tale quantità:

$$p = \frac{H_{Pool}}{H_{Eth}}$$

- Il numero medio di blocchi minati dal pool in un giorno sarà allora dato da:

$$N_{Pool} = p\,N$$

5.2 Mining pool share

Lo **share** è il concetto principale che sta alle base delle operazioni svolte da un mining pool. Lo share è l'hash di un blocco il cui output soddisfa certe condizioni definite dal mining pool. Lo share rappresenta quindi l'hash di un blocco il cui output non è sotto il target (e che quindi non rappresenta un possibile blocco soluzione), ma è comunque un output molto piccolo e difficile da trovare. In sostanza lo share serve per stimare e provare il lavoro svolto dalle macchine dei vari membri del pool. Quando il pool riesce a inserire un blocco riceve una ricompensa che viene poi distribuita tra gli utenti del pool in base al meccanismo di condivisione del pool. Gli shares descrivono quanto la macchina di un particolare membro sta contribuendo al mining pool. Esistono due tipi di shares:

- **accepted**
- **rejected**

Accepted shares indicano che il lavoro svolto da un membro del pool sta contribuendo sostanzialmente al processo di inserimento del blocco e queste vengono premiate. Rejected shares rappresentano invece un lavoro che non contribuisce al processo di mining e quindi non vengono pagate. Anche se il computer di un membro esegue il lavoro con successo ma lo invia in ritardo per quel particolare blocco, questo costituisce un lavoro rifiutato. Ogni membro del pool vuole chiaramente che i propri shares vengano accettati. Tuttavia i rejected shares sono inevitabili poiché è impossibile che tutti i calcoli sulla macchina di un membro siano utili e vengano sempre inviati in tempo. I membri del pool vengono premiati in base alle loro accepted shares che hanno aiutato a inserire un nuovo blocco. Un accepted share non ha un valore effettivo e funge semplicemente da metodo contabile per mantenere equa la distribuzione del premio. Ci sono diversi metodi usati per calcolare il lavoro svolto da ciascun miner e retribuirli opportunamente:

- **Pay-Per-Share (PPS)**: in questo schema gli utenti ricevono un importo fisso per ogni accepted share inviato al pool. Nei sistemi PPS, vieni retribuito anche se il pool non riesce a inserire un blocco. Assumendosi il rischio è quindi probabile che in questo schema il pool imporrà una commissione considerevole agli utenti o all'eventuale block reward.
- **Pay-Per-Last-N-Shares (PPLNS)**: a differenza del PPS, il PPLNS retribuisce i miner solo quando il pool mina un blocco con successo. Quando il pool mina un blocco controlla l'ultima quantità N di share totali inviati. Per calcolare il pagamento di un utente, il pool divide il numero di share che l'utente ha inviato per N e a questo punto moltiplica il risultato per la block reward sottraendo i costi di commissione.

Stratum

L'ecosistema di mining è costituito essenzialmente da due figure: i miner e i mining pool. Le comunicazioni tra i pool e i miner avvengono quasi esclusivamente tramite Stratum. **Stratum** è un protocollo di comunicazione tra pool e miners tramite testi in chiaro, costruito su TCP/IP utilizzando il formato JSON-RPC. Attraverso tale protocollo i miner risolvono i lavori a loro assegnati e inviano i risultati ottenuti sotto forma di shares. La mancanza di protezione crittografica delle comunicazioni ha purtroppo reso il protocollo Stratum vulnerabile a diversi tipi di attacchi. Un attaccante in grado di osservare le comunicazioni tra un miner e il server di un pool può infatti dedurre in maniera accurata i guadagni del miner. Questi attacchi, soprattutto data l'ampia adozione del protocollo Stratum, mostrano come sia Bitcoin che le altcoin non riescano a garantire la privacy e la sicurezza della comunità dei minatori. Inoltre, gli attacchi rivelano che anche un uso esaustivo della crittografia non riesce a garantire la privacy dei minatori, poiché l'accesso al solo timestamp relativo al traffico delle comunicazioni permette a un utente malintenzionato di prevedere i payouts di un miner. Inoltre, un notevole sovraccarico di crittografia rende tale soluzione poco attraente per i pool, che devono gestire il traffico relativo a migliaia di miners contemporaneamente.

5.3 GPU

La **GPU** è un un'unità di elaborazione grafica progettata per accelerare la creazione di immagini. Le GPU moderne, sebbene operino a frequenze più basse delle CPU, sono molto più veloci di esse nell'eseguire i compiti in cui sono specializzate.

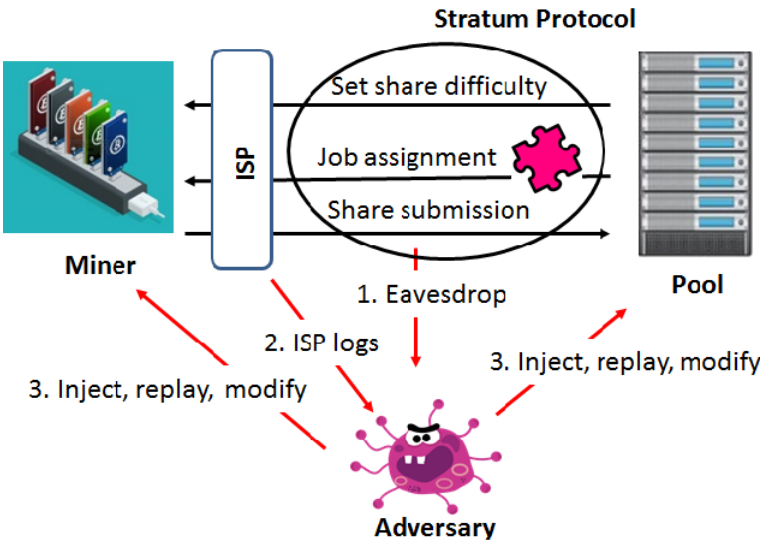

Figura 5.2. Modello di un sistema costituito da un pool, i miners e un attaccante. Il pool e i miners comunicano attraverso il protocollo Stratum che prevede l'assegnazione dei lavori da parte del pool e l'invio dei risultati (shares) da parte dei miners. L'attaccante è in grado di intercettare e modificare le comunicazioni tra i miners e il pool.

Le schede video ad alte prestazioni attualmente presenti sul mercato possono arrivare ad essere anche fino a 800 volte più veloci delle CPU più evolute. Di per sé, le GPU sono concepite per svolgere attività ripetitive a, differenza delle CPU che invece sono progettate per svolgere attività diversificate; le CPU sono difatti più lente ma perfette per eseguire più attività in contemporanea. Inoltre le GPU ono programmabili e vengono usate per l'intelligenza artificiale, elaborazione di dati, gaming, rendering di video e immagini.

Esempi di GPU

- MSI Gaming GeForce RTX 2070
- Nvidia GeForce RTX 2080 Ti

5.4 ASIC

Un circuito integrato per applicazione specifica, noto come **ASIC** o **Application Specific Integrated Circuit**, è un circuito integrato creato appositamente per

Figura 5.3. MSI Gaming GeForce RTX 2070.

Figura 5.4. Nvidia GeForce RTX 2080 Ti.

risolvere un problema specifico (**special purpose**): la specificità della progettazione, focalizzata sulla risoluzione di un unico problema, consente di raggiungere delle prestazioni in termini di velocità di processamento e consumo elettrico difficilmente ottenibili con l'uso di soluzioni più generiche (**general purpose**). Gli ASIC hanno iniziato a diventare popolari con Bitcoin, divenendo l'insieme per eccellenza di processori specifici ottimizzati per il mining dei blocchi. Attualmente la potenza di calcolo della rete Bitcoin è così alta che risulta impossibile fare mining in un altro modo. Il primo ASIC al mondo per Bitcoin è stato sviluppato dalla società cinese Avalon (ora nota come Canaan), il 17 settembre 2012. Recentemente, gli ASIC hanno iniziato a dedicarsi ad altri protocolli di mining e altre criptovalute. Questi sistemi hanno caratteristiche uniche a seconda della valuta a cui sono destinati. Sono infatti progettati specificamente per offrire le migliori

prestazioni possibili per il mining di una certa criptovaluta.

Principali svantaggi degli ASIC

Per quanto riguarda le performance gli ASIC hanno ottime prestazioni in quanto sono stati creati e programmati per fare una determinata e unica operazione. Tuttavia non sono programmabili, dunque se l'algoritmo per il quale sono stati progettati subisce qualche variazione, essi non sono più utilizzabili. I principali difetti degli ASIC sono:

- Centralizzazione del mining
- Difficoltà di acquisto: Di solito è abbastanza difficile ottenere un ASIC nuovo o di seconda mano poiché questi si vendono rapidamente.
- Costo di acquisto elevato
- Consumo energetico: questi computer consumano grandi quantità di energia. Per questo motivo generano molto calore e quindi necessitano di potenti sistemi di raffreddamento.
- Temporalità: Data la rapida evoluzione dell'hardware, gli ASIC diventano rapidamente obsoleti il che li rende non solo un'opzione costosa, ma anche poco duratura.
- ASIC Resistant: Alcuni progetti di criptovalute rifiutano la centralizzazione che questi dispositivi comportano: ecco perché si basano su algoritmi resistenti contro gli stessi. Tale resistenza impedisce a chi li possiede di trarre maggiori profitti dal mining utilizzando ASIC.

Esempi di ASIC

- **Canaan (Old Avalon)**: Questa società si occupa della produzione dei criptominer Avalon. Questi ASIC, specializzati nel mining di Bitcoin, hanno la caratteristica principale di essere molto economici e di avere una potenza elevata.
- **Bitmain**: questa società è la più grande progettista al mondo di ASIC per il mining di Bitcoin. La società gestisce anche Antpool, storicamente uno dei più grandi pool di mining di Bitcoin.

Figura 5.5. AvalonMiner 1246. Hasharate: 90 TH/s, Potenza: 3420 W.

Figura 5.6. Antminer S19 XP. Hasharate: 140 TH/s, Potenza: 3010 W.

5.5 Rig

In questa sezione vengono descritte le componenti di un Rig prendendo spunto da quello realizzato dal professore Antonio Di Scala e dall'ingegnere Gabriele Vernetti presso la sede PING-S s.r.l., Cuneo. Si tratta di un Rig di GPU costituito da 8 NVIDIA 1080 Ti.

Un mining Rig è una disposizione di elementi hardware, CPU, GPU, FPGA o ASIC che sono stati predisposti per eseguire il mining di una determinata criptovaluta.

Le principali componenti di un rig sono:

- Motherboard
- Alimentatore
- GPU (nel caso in questione)
- Random access memory (RAM): in molti casi la dimensione della RAM non ha un impatto diretto sulla performance del mining. Tuttavia vi sono alcune linee guida come i requisiti minimi del sistema operativo che devono essere seguite affinchè l'impianto di mining funzioni in maniera stabile.
- Power supply unit (PSU): nella scelta di una PSU è fondamentale considerare la potenza totale richiesta dal mining rig. OuterVision Power Supply Calculator è uno dei calcolatori più accurati di consumo energetico che, in base alla potenza totale richiesta dal mining rig raccomanda di utilizzare una determinata Power supply. Tra le specifiche che la PSU deve soddisfare compare la potenza minima raccomandata per i componenti selezionati. Infatti un alimentatore di potenza inferiore a quella richiesta aumenta il rischio che il sistema diventi instabile e il possibile spegnimento dell'alimentatore. Viene inoltre consigliato un determinato UPS rating: UPS è un dispositivo che fornisce protezione da sbalzi di tensione e che consente a un computer di continuare a funzionare per almeno un breve periodo in caso di perdita della fonte di alimentazione principale.
- Powered risers: permettono di collegare le GPU alla motherboard. Le GPU vanno collegate ai risers, i quali a loro volta poggiano sul mining frame a una certa distanza tale che le ventole (delle GPU) possano funzionare in maniera

adeguata. I risers dispongono inoltre di un circuito elettronico che permette di collegarli alla motherboard tramite cavo USB.

- Heatsink fan (ventola di raffreddamento per la CPU): è consigliato posizionare una ventola di raffreddamento sopra la CPU.
- Mining rig frame

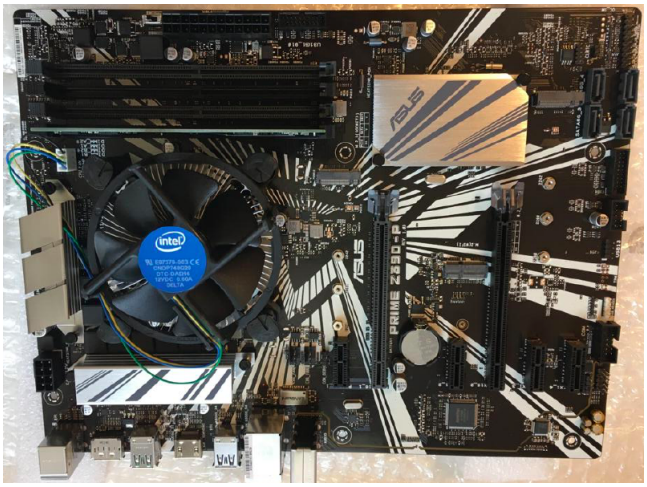

Figura 5.7. Motherboard.

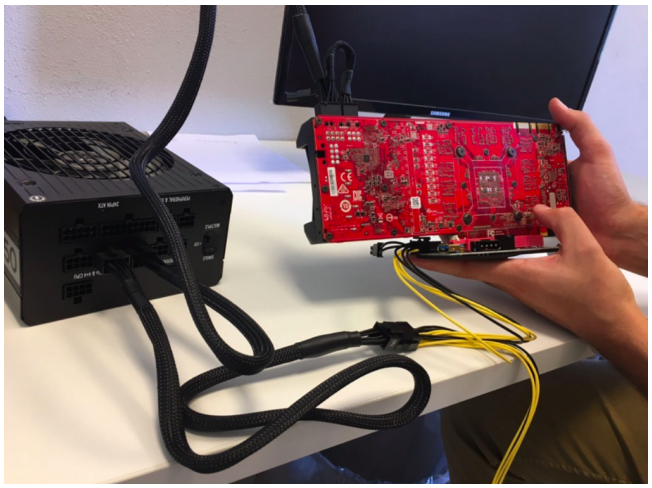

Figura 5.8. Alimentatore e GPU.

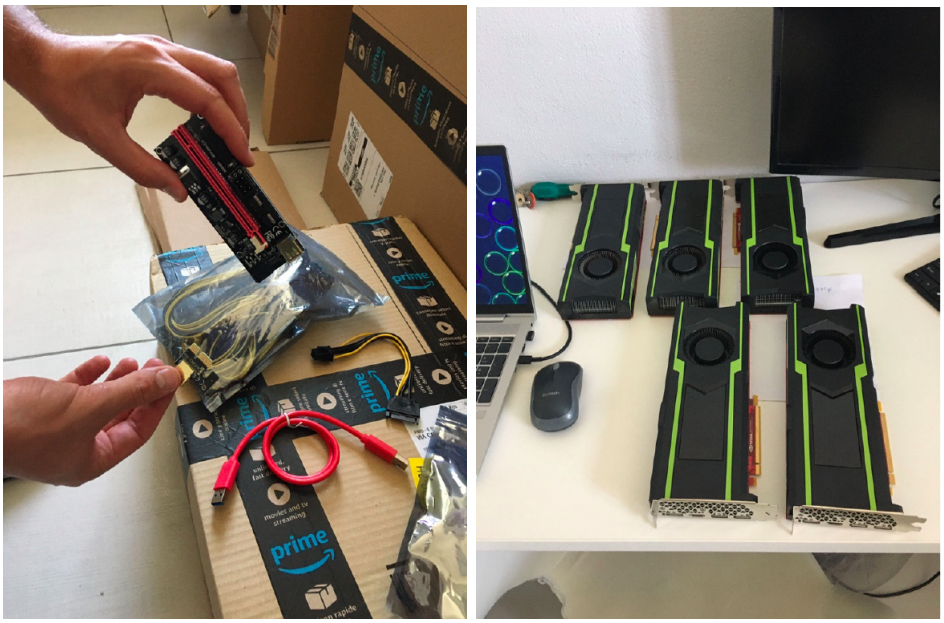

Figura 5.9. Risers e GPU.

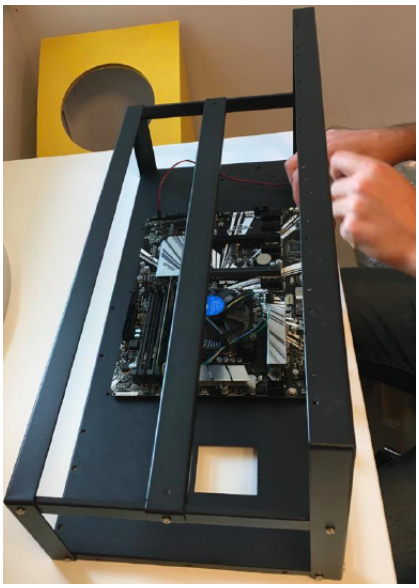

Figura 5.10. Rig frame.

Figura 5.11. Rig: 8 GPU GEFORCE GTX 1080 Ti.

Figura 5.12. Rig: 8 GPU GEFORCE GTX 1080 Ti.

Capitolo 6

Analisi di fattibilità del mining

In prima analisi i due fattori principali che caratterizzano una macchina utilizzata per fare mining sono:

- Hashrate $[h/s]$
- Potenza $[W]$

Il primo descrive la potenza computazionale della macchina in termini di numero di hash al secondo, mentre il secondo è legato al consumo elettrico. L'Hashrate può variare in base alla moneta minata. Nella sezione successiva le schermate di Hive OS sono relative al Rig mostrato alla fine del Capitolo 5.

6.1 Rig: collegamento ad un Pool

Una volta assemblato, è possibile monitorare un Rig attraverso il collegamento ad un pool. Per fare ciò è necessario:

- installare il sistema operativo (nel caso in questione Hive OS)
- creare l'account sul sito del pool (nel caso in questione Hiveon)

Hive OS è un sistema operativo sviluppato sulla base del Linux distribution Ubuntu 16.04 LTS. La sua missione principale è fornire un'interfaccia semplice ed efficace per il mining di criptovalute su GPU e la gestione dei rigs.

Figura 6.1. Hive OS su App Store.

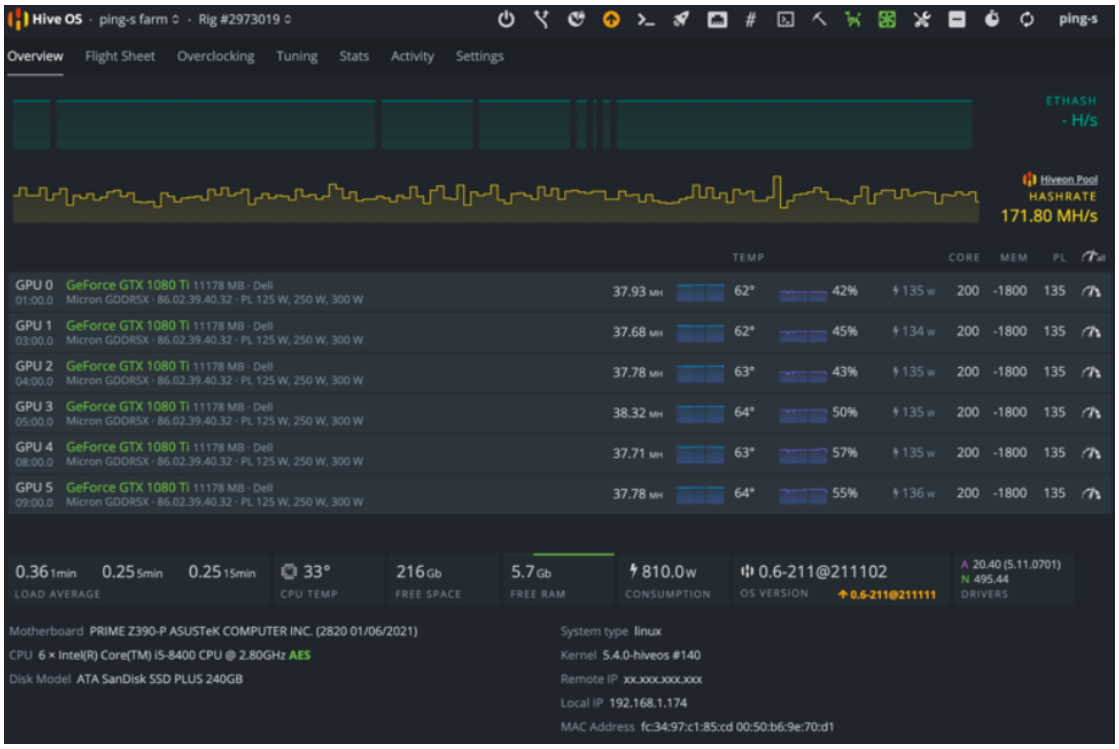

Figura 6.2. Schermata di Hive OS.

Hiveon Pool, basato sul sistema operativo Hive OS, ha una Pool Fee dello 0%, il che significa che il pool non fa pagare commissioni. Inoltre Hiveon ha un payout minimo di 0.1 ETH, di conseguenza un miner non verrà pagato dal pool fino a quando non avrà raggiunto tale soglia. Creato l'account nel pool è possibile sperimentare le impostazioni di overlocking e controllo temperatura fornite integralmente da Hive OS. Ad esempio la funzione "Autofan-mode" è molto utile per mantenere il controllo sulla temperatura delle GPU. Tramite questa funzionalità è

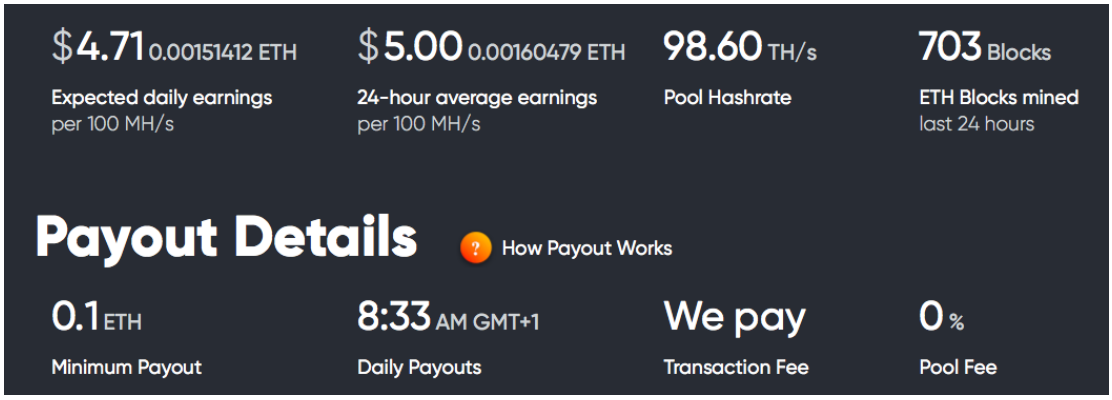

Figura 6.3. Hiveon Pool.

infatti possibile stabilire una temperatura target da mantenere, così da permettere ad Hive OS di regolare automaticamente l'intensità delle ventole di ogni scheda video, in funzione della temperatura massima stabilita. Dopo aver provato diverse impostazioni, l'obbiettivo è quello di trovare la migliore combinazione di parametri in termini di rapporto hash/potenza. Esistono inoltre dei sistemi che essendo collegati al WI-FI permettono di monitorare e controllare da remoto il consumo elettrico del rig.

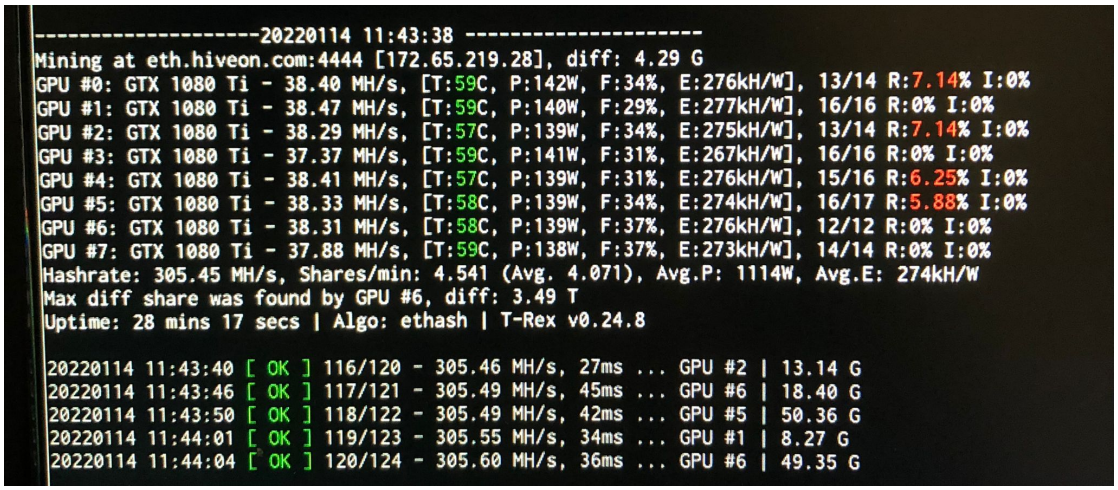

Figura 6.4. Schermata di HiveOS: prestazioni del Rig.

La figura 6.5 mostra una schermata del sistema operativo Hive OS relativa alle

shares prodotte dal rig nel tempo. Come possiamo vedere esistono 3 tipi di share:

- **Valid**: rappresentano gli shares accettati come validi. Questo significa che il computer ha risolto il compito matematico assegnato dalla rete e ha inviato la soluzione in tempo. Questi shares portano profitto.
- **Stale**: sono gli shares inviati in ritardo ossia quando il pool sta già lavorando per l'inserimento del blocco successivo. Ciò significa che il computer aveva trovato la soluzione corretta, ma era troppo tardi per inviarla. Questi accadono occasionalmente, non portano profitto, ma allo stesso tempo non influenzano il processo di mining.
- **Invalid**: sono le soluzioni errate che il computer potrebbe inviare. Di solito sono un segnale di un'errata configurazione del processo di mining.

Gli invalid shares potrebbero essere causati da un problema con il software che si sta utilizzando (impostazioni errate) o con l'hardware. La figura 6.6 mostra

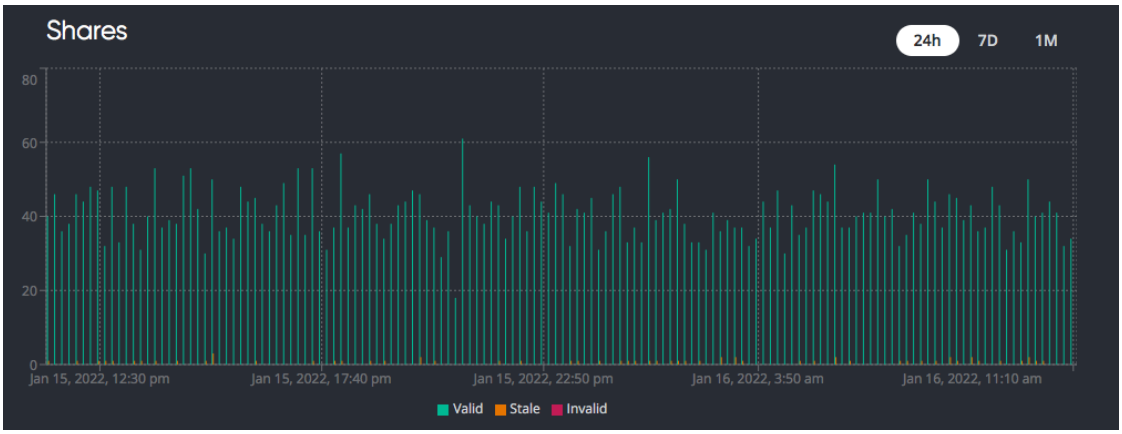

Figura 6.5. Shares: Valid, Stale, Invalid.

l'andamento dell'hashrate complessivo del rig in funzione del tempo. La curva azzurra, che è l'unica che presenta brusche oscillazioni, rappresenta il reale andamento nel tempo dell'hashrate del rig. Mentre la curva arancione e quella rossa rappresentano l'hashrate medio previsto e quello effettivamente riportato. Come si può osservare dalla figura, il valore medio dell'hashrate del rig, nell'intervallo di tempo mostrato in figura, è stabile intorno ai 300 Mh/s. La figura 6.7 mostra i pagamenti ricevuti dal pool: da dicembre ad Aprile i pagamenti sono stati di circa 0.1 ETH ogni 22/24 giorni. Invece i pagamenti da Aprile ad Agosto, a seguito di

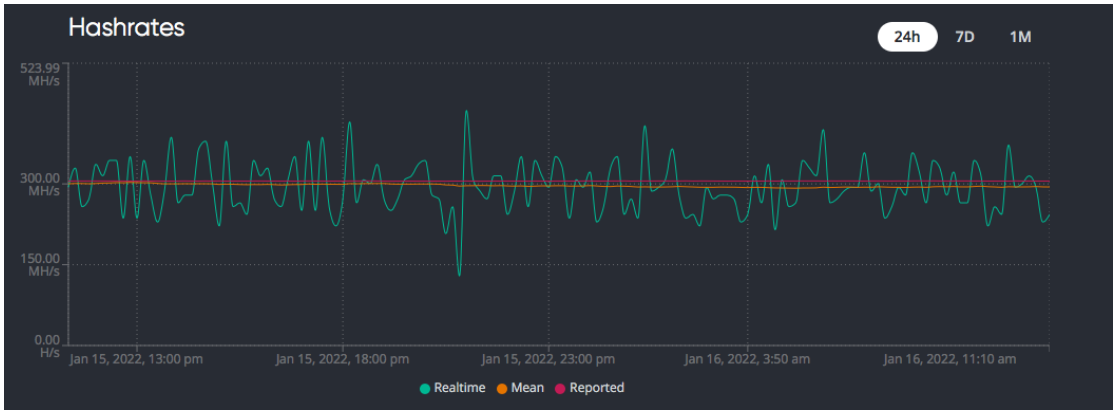

Figura 6.6. Hashrates: Realtime, Mean, Reported.

una modifica delle impostazioni di payout, sono stati di circa 0.2 ETH; ciò chiaramente ha circa raddoppiato l'intervallo di tempo tra un payout e un altro. La

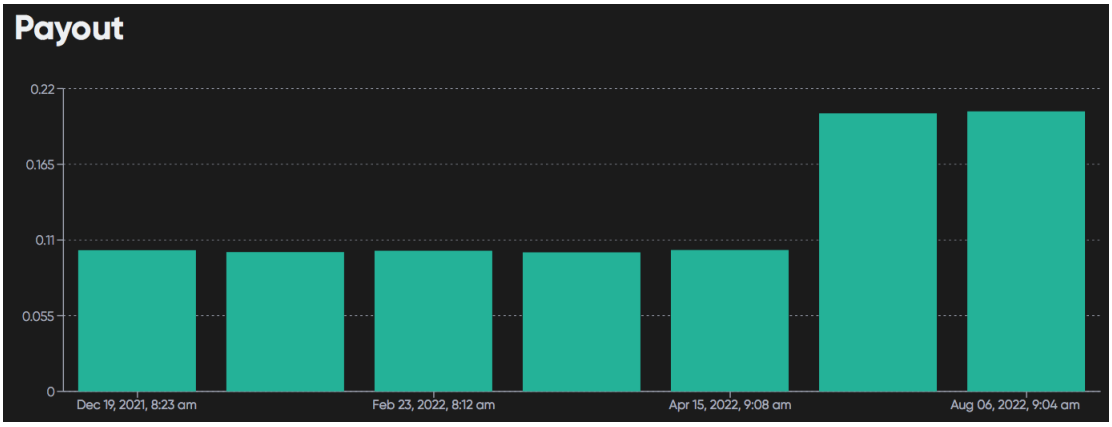

Figura 6.7. Pagamenti ricevuti dal pool.

figura 6.8 mostra l'elenco delle transazioni inviate dal pool: la prima risale al 19 Dicembre 2021 mentre l'ultima risale al 6 Agosto 2022.

6.2 Whattomine

Whattomine è un famoso calcolatore di profitti per i miners di criptovalute. Il primo step è quello di selezionare le macchine che si vogliono utilizzare per l'attività di mining tra quelle proposte nel sito. In alternativa si possono inserire

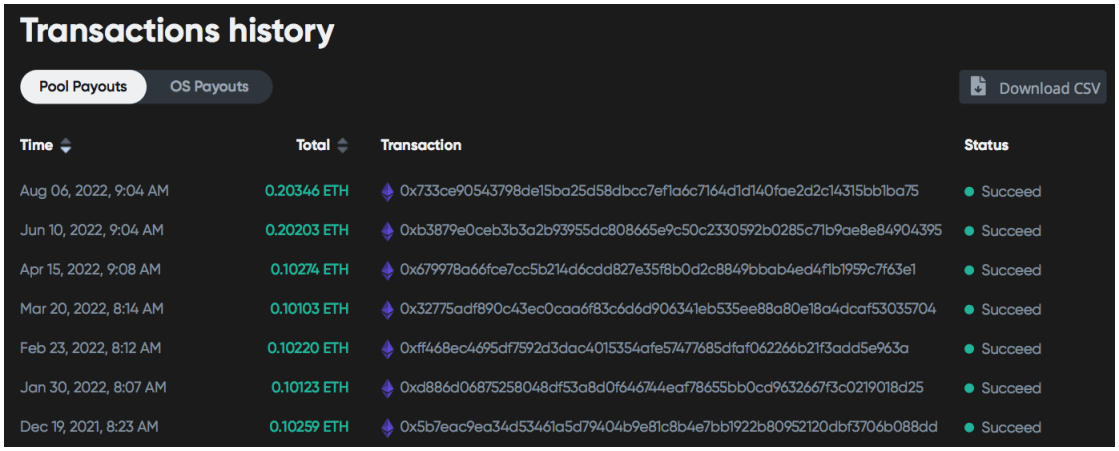

Figura 6.8. Elenco delle transazioni inviate dal pool.

manualmente l'hasharate e la potenza di una generica macchina. Successivamente si seleziona l'algoritmo che si vuole utilizzare (ad esempio nel caso di Bitcoin si seleziona SHA-256 mentre nel caso di ETHW si seleziona Ethash). Notiamo come le prestazioni della macchina in termini di hashrate e consumo elettrico varino a seconda dell'algoritmo selezionato. Questo chiaramente permette di vedere quanto sia profittevole minare una certa moneta piuttosto che un'altra. Infine si dà in input il costo di 1 kWh e a questo punto clicclando sul pulsante "Calcola" si ottengono i vari output.

Esempio: mining su EthereumPoW.
Consideriamo un rig costituito da una GEFORCE RTX 3070 e una GEFORCE GTX 1080 Ti e supponiamo di voler svolgere l'attività di mining sulla Blockchain di EthereumPoW.

Figura 6.9. Selezione delle GPU: GEFORCE RTX 3070, GEFORCE GTX 1080 Ti.

Come si può osservare dalla figura 6.13 il guadagno lordo giornaliero (MOL) in ETHW è di 0.029111. Tuttavia il profitto è di -$1.26 il che significa che i consumi elettrici superano il ricavo derivante dalla quantità di criptomoneta estratta.

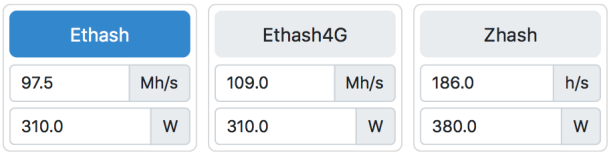

Figura 6.10. Selezione dell'algoritmo: Ethash.

Figura 6.11. Costo $kWh = 0.22$

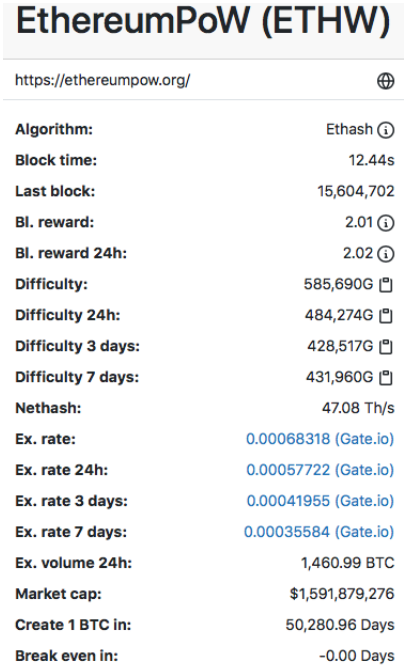

Figura 6.12. Whattomine: parametri di input.

Estimated Rewards

Per	Fees	Est. Rewards	Rev. BTC	Rev. $	Cost	Profit
Hour	0.000000	0.001213	0.000001	$0.02	$0.07	-$0.05
Day	0.000000	0.029111	0.000020	$0.38	$1.64	-$1.26

Figura 6.13. Whattomine: Output.

Esempio: mining su Bitcoin.
Supponiamo di voler fare mining sulla Blockchain di Bitcoin con l'ASIC ANTIMNER S19 XP. Anche in questo caso dalla figura 6.16 si può osservare come il

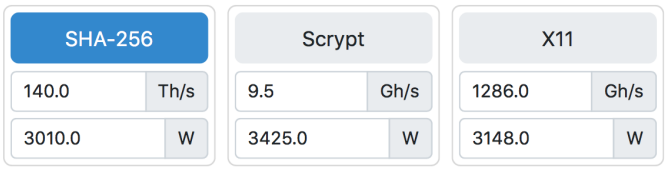

Figura 6.14. Selezione ASIC e algoritmo: ANTIMNER S19 XP, SHA-256.

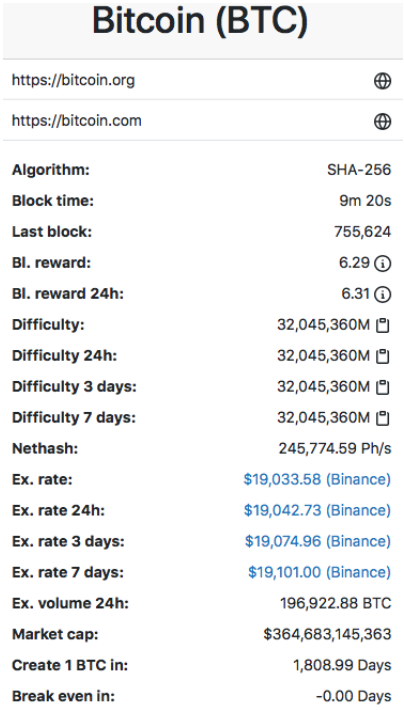

Figura 6.15. Whattomine: parametri di input.

profitto sia negativo. Ciò significa che attualmente non è conveniente fare mining nemmeno con uno degli ASIC più potenti ed efficienti che ci sono in circolazione.

Estimated Rewards

Per	Fees	Est. Rewards	Rev. BTC	Rev. $	Cost	Profit
Hour	0.000000	0.000023	0.000023	$0.44	$0.66	-$0.22
Day	0.000000	0.000553	0.000553	$10.52	$15.89	-$5.37

Figura 6.16. Whattomine: Output.

6.3 Modello e derivazione del profitto medio giornaliero

In seguito verrà indicata con [crypto] l'unità (di misura) della criptomoneta utilizzata. Sia H l'hash rate della rete e h l'hash rate di una certa macchina . Sia X una variabile aleatoria discreta con distribuzione di Bernoulli di parametro p, dove

$$p = \frac{h}{H} \tag{6.1}$$

è la probabilità che la macchina inserisca il blocco successivo nella Blockchain. Indicando con $X = 1$ il successo, ossia l'inserimento del blocco, e con $X = 0$ il fallimento si ha

$$\begin{cases} \mathcal{P}(X = 1) = p \\ \mathcal{P}(X = 0) = 1 - p \end{cases} \tag{6.2}$$

$$E(X) = p \tag{6.3}$$

dove $E(X)$ indica il valore atteso di X. Per calcolare il profitto medio derivante dall'attività di mining è necessario conoscere i seguenti dati:

- h: hashrate della macchina $[h/s]$
- H: hashrate della rete $[h/s]$
- v: valore attuale della criptomoneta [€]
- B_t: Block time $[s]$
- R_b: ricompensa per l'inserimento di un blocco [crypto]
- P: potenza della macchina $[W]$
- c : costo di 1 kWh [€]

Ogni B_{time} secondi la rete guadagna complessivamente R_b criptomonete. Possiamo dunque immaginare che ogni B_{time} secondi la macchina permetta di guadagnare in media una quantità di crittomonete L_{block}, dove

$$L_{block} = p\, R_b = \frac{h}{H} R_b \tag{6.4}$$

Il guadagno lordo al giorno L_{day} sarà allora dato da

$$L_{day} = L_{block} \frac{60}{B_t} \cdot 60 \cdot 24 \tag{6.5}$$

Per ottenere il guadagno netto al giorno bisogna sottrarre ad L_{day} i costi relativi al consumo elettrico. Il costo dell'energia al giorno E_{day} è dato da

$$E_{day} = \frac{P}{1000}\, c\frac{1}{v} \tag{6.6}$$

Il guadagno netto al giorno MOL, in unità di criptomoneta, sarà quindi dato da

$$MOL = L_{day} - E_{day} \tag{6.7}$$

Nota: i calcoli sopra riportati si basano su valori medi di conseguenza il risultati finali possono subire delle variazioni. Nel caso in cui si abbiano più macchine si può ripetere lo stesso ragionamento dove h è la somma delle potenze di calcolo delle varie macchine e P è la somma della potenze.

6.4 Simulatore e analisi di fattibilità del mining

Al fine di effettuare un'analisi di fattibilità del mining, tramite il programma Numbers ho realizzato un simulatore che permette di calcolare il guadagno netto giornaliero (MOL). La prima cosa da fare è scegliere la tipologia di macchine che si vogliono utilizzare per il proprio rig. Per ogni tipologia di macchina selezionata vengono richiesti i seguenti dati in input:

- Numero di macchine (della tipologia selezionata)
- Costo di acquisto di una singola macchina [€]
- Hashrate $[h/s]$
- Potenza $[W]$

Sono inoltri necessari i seguenti parametri relativi alla Blockchain e alla crypto a cui si fa riferimento:

- Hashrate della rete $[h/s]$
- Valore cripto [€]
- Average Block time $[s]$
- Reward per block [crypto] (include le fees)

L'Average Block time indica ogni quanti secondi viene inserito un blocco nella Blockchain mentre il Reward per block indica la ricompensa per l'inserimento di un blocco. Infine l'ultimo parametro da inserire in input è il seguente:

- Costo kWh [€]

Il simulatore restituisce in output:

- Ricavo lordo al giorno [crypto]
- Mol al giorno [crypto]
- Mol al giorno [€]
- Break even [mesi]
- Rapporto hashrate/potenza (adimensionalizzato)

Il Break even indica dopo quanto tempo un miner riesce a recuperare i soldi spesi per l'acquisto delle macchine. Il rapporto hashrate/potenza è invece un parametro che ho inserito, dopo svariati confronti con Alessandra Rostagno (responsabile dell'azienda), per valutare l'efficienza di una macchina rispetto ad un altra: più è alto tale rapporto e più una macchina è efficiente. Nel modello non ho considerato tutti i costi, come i costi relativi alla manutenzione, in quanto è molto difficile trovare un prezzo di riferimento e per tale motivo spesso si ipotizza una percentuale di costo.

Ciò che si può notare facilmente è che il consumo elettrico incide in maniera determinante sul MOL. Dunque da questo punto di vista la potenza della macchina e il costo di un kWh sono quindi i due parametri fondamentali per il calcolo dei consumi: più tali valori sono alti e più chiaramente incideranno sul MOL in maniera significativa. Proprio per tale ragione tra i vari input ho inserito infine anche il

MOL giornaliero desiderato a cui corrisponde in output il massimo costo che deve avere un kWh per ottenere quel determinato MOL.

Esempio: mining su EthereumPoW.
Consideriamo lo stesso esempio della sezione precedente per il mining sulla Blockchain di EthereumPoW. Le GPU che consideriamo sono una GEFORCE RTX 3070 e una GEFORCE GTX 1080 Ti. Mettendo in input nel simulatore gli stessi parametri che utilizza Whattomine si ottengono i seguenti risultati.

	GEFORCE GTX 1080 Ti	GEFORCE RTX 3070		
Costo di acquisto [Euro]	550	1300	Hashrate della rete [h/s]	47080000000000
Costo di acquisto [coin]	42,406	100,231	Valore crypto [Euro]	12,97
Quantità	1	1	Average Block time [s]	12,44
Hashrate [h/s]	37500000	60000000	Reward per block [crypto]	2,02
Potenza [W]	180	130	Costo kWh [Euro]	0,22
Hashrate totale [h/s]	37500000	60000000	Ricavo lordo al giorno [crypto]	0,029112
Potenza totale [W]	180	130	Mol al giorno [crypto]	-0,09709
Consumo elettrico mensile [Euro]	28,512	20,592	Mol al giorno [Euro]	-1,2592
Consumo elettrico al giorno [coin]	0,0733	0,0529	Break even [mesi]	-49,0
rapporto hashrate/ potenza	0,208	0,462	Mol al giorno desiderato[crypto]	0,01
			Costo kwh massimo [Euro]	0,03

Figura 6.17. Simulatore.

Confrontando i risultati ottenuti con quelli di Whattomine si può osservare l'accuratezza del simulatore. Inoltre notiamo come il rapporto hashrate/potenza della 3070 sia più del doppio di quello della 1080 Ti; ciò significa che a livello di performance la 3070 è senza dubbio migliore avendo un hashrate maggiore e una potenza dissipata inferiore rispetto alla 1080 Ti (motivo per il quale la 3070 costa di più).

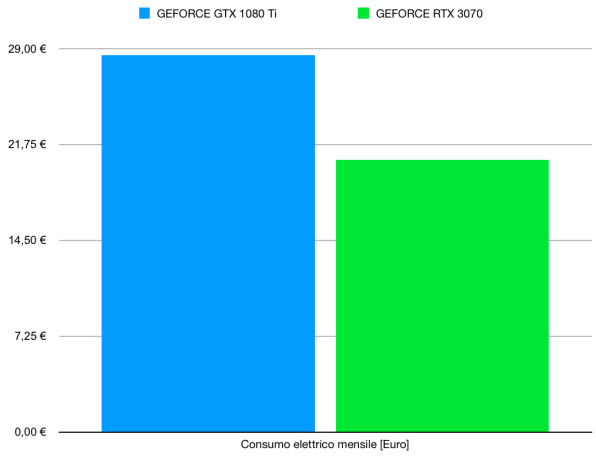

Figura 6.18. Consumi elettrici mensili.

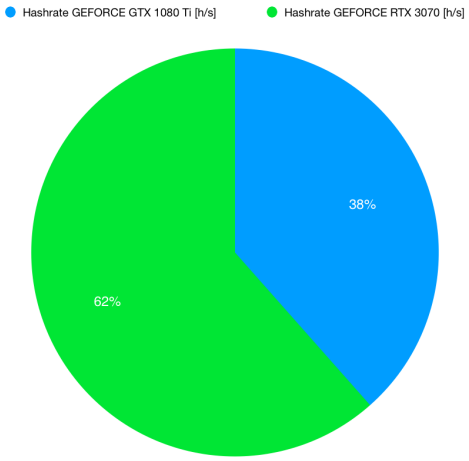

Figura 6.19. Partizione della potenza di calcolo complessiva.

Esempio: mining su Bitcoin.
Consideriamo lo stesso esempio della sezione precedente per il mining sulla Blockchain di Bitcoin. L'ASIC che consideriamo è un ANTIMNER S19 XP. Mettendo in input nel simulatore gli stessi parametri che utilizza Whattomine si ottengono i risultati mostrati in figura 6.20. Anche in questo caso confrontando i risultati ottenuti con quelli di Whattomine, notiamo come i due simulatori restituiscano in output pressochè gli stessi valori.

	ANTMINER S19 XP			
Costo di acquisto [Euro]	8000	0	Hashrate della rete [h/s]	245774590000000000
Costo di acquisto [coin]	0,421	0,000	Valore crypto [Euro]	18996,97
Quantità	1	0	Average Block time [s]	560
Hashrate [h/s]	140000000000000	0	Reward per block [crypto]	6,31
Potenza [W]	3010	0	Costo kWh [Euro]	0,22
Hashrate totale [h/s]	140000000000000	0	Ricavo lordo al giorno [crypto]	0,000555
Potenza totale [W]	3010	0	Mol al giorno [crypto]	-0,00028
Consumo elettrico mensile [Euro]	476,784	0	Mol al giorno [Euro]	-5,36
Consumo elettrico al giorno [coin]	0,0008	0,0000	Break even [mesi]	-49,8
rapporto hashrate/ potenza	0,465		Mol al giorno desiderato[crypto]	0,00001
			Costo kwh massimo [Euro]	0,14

Figura 6.20. Simulatore.

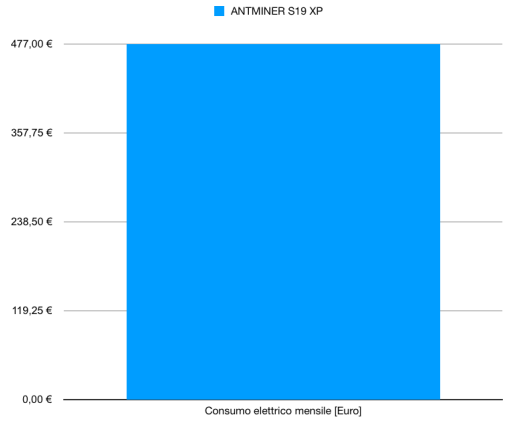

Figura 6.21. Consumi elettrici mensili.

6.5 Cenni alla crittografia post-quantum

I sistemi crittografici oggi in uso basano la loro sicurezza su alcuni problemi matematici che non sono risolvibili, in tempi ragionevoli, dagli attuali algoritmi implementati sui computer classici. Tuttavia, sono stati esibiti algoritmi che, con l'avvento dei computer quantistici, potrebbero risolvere efficientemente tali problemi, rendendo quindi inutilizzabili gli attuali schemi crittografici. Nel 1994 il

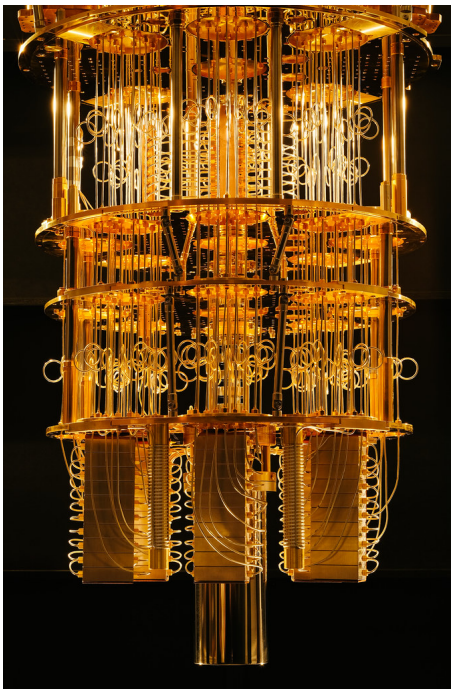

Figura 6.22. Computer quantistico.

crittografo Peter Shor pubblicò l'algoritmo che porta il suo nome, per la fattorizzazione dei numeri interi in tempo polinomiale. Shor riuscì a dimostrare che molte delle primitive crittografiche in uso (come DSA, RSA e la crittografia su curve ellittiche) sarebbero diventate non sicure se quel suo algoritmo fosse stato lanciato da un computer quantistico. Questa è stata una svolta fondamentale, in quanto la sicurezza di RSA si basava proprio sull'assunzione che la fattorizzazione dei numeri interi non disponesse di di un algoritmo efficiente. Da questo punto di vista l'interesse scientifico va verso la Quantum Safe Cryptography il cui obbiettivo principale è quello di capire quali sistemi crittografici saranno sicuri contro i computer quantistici. Il fatto che i computer quantistici possono rompere sistemi

come RSA, DSA e ECDSA, non implica tuttavia la fine della crittografia. Esistono infatti molte alternative valide.

6.5.1 Lattice based model

I lattice based model risultano essere ad oggi uno tra i più validi candidati per la crittografia post-quantum. A differenza degli schemi a chiave pubblica come RSA, Diffie-Hellman o i crittosistemi su curve ellittiche, che potrebbero essere rotti utilizzando l'algoritmo di Shor su un computer quantistico, alcune costruzioni basate su reticolo sembrano essere resistenti agli attacchi sia dei computer classici che di quelli quantistici. Nel luglio 2020 il National Institute of Standards and Technology (NIST) aveva annunciato gli algoritmi arrivati al terzo round del processo di standardizzazione della crittografia post-quantum. Tra gli algoritmi di cifratura a chiave pubblica e algoritmi di firma digitale, era stati in tutto 15 i candidati selezionati. Inoltre proprio negli ultimi mesi il NIST ha annunciato i finalisti del terzo round e tra gli algoritmi di cifratura a chiave pubblica c'è NTRU che è il principale public key encryption system basato su un lattice based model. E' di assoluta importanza che la sicurezza evolva di pari passo (se non più velocemente) del quantum computing. Ecco perchè la ricerca va nella direzione di creare modelli matematici sempre più complicati in modo tale che l'avvento dei computer quantistici non metta a rischio la sicurezza e la privacy degli utenti.

Cryptosystem	**Broken by Quantum Algorithms?**
RSA public key encryption	Broken
Diffie-Hellman key-exchange	Broken
Elliptic curve cryptography	Broken
Buchmann-Williams key-exchange	Broken
Algebraically Homomorphic	Broken
McEliece public key encryption	Not broken yet
NTRU public key encryption	Not broken yet
Lattice-based public key encryption	Not broken yet

Figura 6.23. Criptosistemi rotti dagli algoritmi quantistici.

6.6 Conclusioni

Il mondo del mining è un panorama complesso in cui i fattori che entrano in gioco sono molteplici:

- **Scelta della moneta**: la profittabilità di una certa macchina varia a seconda della moneta che si vuole minare e quindi dipende dall'algoritmo utilizzato. Questo permette di fare dei confronti per capire quanto sia profittevole minare una certa moneta piuttosto che un'altra.
- **Andamento del prezzo della moneta**: le fluttuazioni del valore della moneta sono uno dei principali fattori che influisce in maniera preponderante sulla profittabilità dell'investimento. Questo perchè se la moneta che si sta minando presenta un brusco calo del proprio valore di acquisto (come è successo nell'ultimo anno a Bitcoin, Ethereum e molte altre Altcoin) allora risulta veramente difficile ottenere un guadagno tramite il mining, come visto alla fine della sezione 6.4.

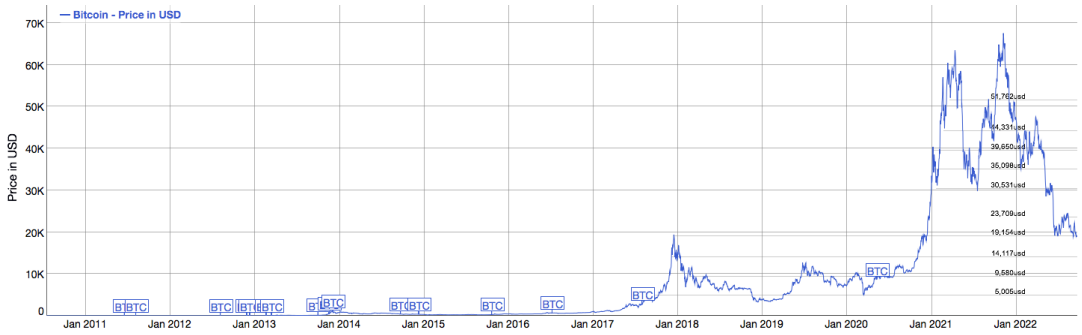

Figura 6.24. Andamento del prezzo di Bitcoin.

- **Hashrate della macchina e potenza dissipata**: una buona macchina da mining deve avere una buona potenza computazionale e consumi elettrici modesti. Nel momento in cui i costi legati all'energia superano il guadagno lordo si ottiene un MOL negativo, il che rende l'attività di mining chiaramente non profittevole (conviene spegnere la macchina). In tal caso è chiaramente più sensato acquistare una moneta attraverso un exchange. Tuttavia la volatilità del mercato può portare anche a considerazioni di tipo speculativo. Ad esempio una grande azienda che fa mining può decidere di accumulare grandi quantità di criptomoneta senza effettuare una conversione in Euro. Di conseguenza tale azienda non potrà momentaneamente recuperare i soldi relativi al

consumo elettrico. Tuttavia potrà cercare di recuperare tali costi attraverso altre entrate e quindi accumulare criptomoneta sperando in un andamento crescente del prezzo di acquisto di quest'ultima.

- **Hashrate della rete**: quando Bitcoin è stato lanciato sul mercato nel 2009, chiunque con un normale PC poteva fare mining e competere con tutti i miners della rete per l'inserimento del blocco successivo nella Blockchain. Questo perchè la difficulty e l'hasharate della rete avevano valori molto più piccoli rispetto a quelli attuali e di conseguenza non erano necessari hardware specializzati per il mining. L'hashrate della rete è però fortemente aumentata

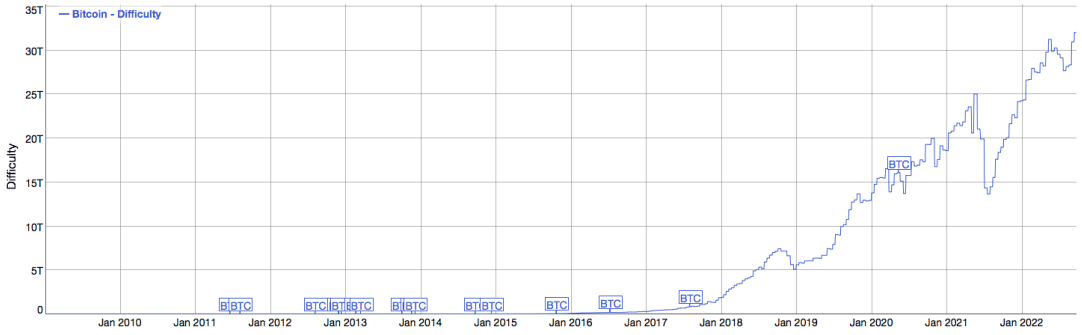

Figura 6.25. Bitcoin Difficulty.

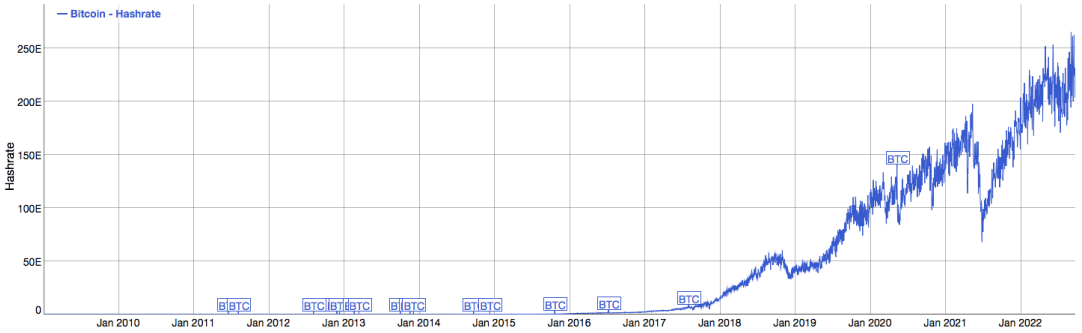

Figura 6.26. Bitcoin Hashrate.

nel tempo e questo ha provocato due importanti cambiamenti nell'ecosistema: la sostituzione delle CPU dei computer con GPU, FPGA e ASIC e la nascita dei mining pool (vedi il Capitolo 5).

- **Costo di acquisto elevato delle macchine**: il break even indica proprio dopo quanto tempo, tramite i ricavi derivanti dal mining, si recuperano i soldi spesi per l'acquisto della macchina.

Nel caso di Bitcoin c'è inoltre un ulteriore fattore: il reward per block si dimezza ogni volta che avviene un halving. Dunque se ciò non viene controbilanciato da una crescita del valore della moneta, allora fare mining col passare del tempo diventerà sempre meno profittevole.

Bibliografia

[1] W.Diffie, M.E. Hellman. *New Directions in Cryptography*. IEEE Transactions on Information Theory,1976.

[2] Neal Koblitz. *Elliptic curve cryptosystems*. Mathematics of Computation, Vol. 48, N. 177 (1987), 203-209.

[3] S.Miller. *Use of elliptic curves in cryptography*. CRYPTO, Lecture Notes in Computer Science 85, 417-426.

[4] R. L. Rivest, A. Shamir, L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Comm. of ACM. 21(2), 120-126, 1977.

[5] Andreas M. Antonopoulos. Mastering Bitcoin: Unlocking Digital Cryptocurrencies, O'Reilly Associates Inc, 978-1449374044, 2014.

[6] ndreas M. Antonopoulos and Dr. Gavin Hood. Mastering Ethereum: Building Smart Contracts and Dapps, O'Reilly Associates Inc, 978-1491971949, 2018.

[7] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen. *Post-Quantum Cryptography*. Springer Nature, 2009

[8] https://www.nvidia.com/it-it/

[9] https://whattomine.com

[10] https://www.blockchain.com

[11] https://bitcoin.org/it/

[12] https://www.kryptex.com/it/

[13] https://academy.bit2me.com/it/