Studio di fattibilità di implementazione di processori open-hardware in grado di integrarsi in una architettura blockchain

Report 2:

Primitive crittografiche utilizzate nelle architetture blockchain compatibili con dispositivi open-hardware

Con la collaborazione di:
Università di Trento

UNIVERSITÀ DI TRENTO
Dipartimento di Matematica
Laboratorio di Matematica Industriale e Crittografia

Fornitore:
MicroFabSolutions s.r.l.

MicroFabSolutions

Attività svolta nell'ambito dell'Avviso promosso dal Ministero dell'Università e della Ricerca per la presentazione di Idee progettuali per Smart Cities and Communities and Social Innovation di cui al D.D. n. 391/Ric. del 5 luglio 2012 e ss.mm.ii. - SIN_00968 THE LEARNING METERS NETWORK: workpackage formativo del SCN_00398 - CUP J49G14000140008.

Indice

1 Introduzione

1.1 Obiettivo dello studio

L'architettura blockchain, letteralmente "catena di blocchi", sfrutta le caratteristiche di una rete informatica di nodi e consente di gestire e aggiornare, in modo immutabile e sicuro, un registro contenente dati e informazioni (per esempio, transazioni) in maniera aperta, condivisa e distribuita senza la necessità di un'entità centrale di controllo e verifica. Dalla sua comparsa nel 2009, la blockchain ha subito ricevuto un forte interesse, per i suoi vari ambiti applicativi: crittovalute e settore finanziario, notarizzazione di dati, tracciamento di beni, memorizzazione di flussi di dati generati da sensori.

Il funzionamento e l'affidabilità della blockchain si fonda sull'utilizzo di diverse primitive crittografiche, come funzioni di hash e firme digitali. Esse basano la loro sicurezza sulla difficoltà di risoluzione di problemi matematici selezionati. In particolare, gli algoritmi di firma digitale maggiormente utilizzati sono protocolli che eseguono efficientemente operazioni definite su gruppi di punti di curve ellittiche, e basano la loro sicurezza sulla difficoltà del problema del logaritmo discreto. Esiste quindi un trade-off tra l'efficienza (selezione di curve ellittiche in cui le operazioni di gruppo possono essere efficientemente eseguite) e la sicurezza (le curve ellittiche devono essere selezionate in modo da evitare vulnerabilità indotte da proprietà algebrico-geometriche). Le funzioni di hash sono invece funzioni in cui la controimmagine di un elemento non è calcolabile in tempo polinomiale. Le funzioni di hash basano la loro sicurezza sulla difficoltà di risoluzione di equazioni nonlineari su campi finiti. Anche in questo caso occorre prestare attenzione al rapporto tra sicurezza e efficienza di esecuzione delle operazioni.

Ai fini della sicurezza e dell'efficienza, risulta utile avere pieno controllo degli algoritmi di cifratura implementati e del design dei microprocessori integrati nell'architettura blockchain.

Diverse piattaforme hardware sono specializzate nell'implementazione di primitive crittografiche, e sul mercato sono disponibili diversi processori crittografici, come, per esempio, generatori hardware di numeri casuali, coprocessori crittografici specializzati sulle versioni di AES e motori crittografici per il calcolo di hash.

Negli ultimi anni, alcune società no profit, come la Risc-V Foundation, si sono costituite con l'obiettivo di promuovere l'adozione e l'implementazione di hardware open-source. Questo tipo di progetto consente a piccole aziende di realizzare processori e microcontrollori personalizzati senza pagare i relativi diritti a società terze, grazie alla natura open-source del progetto stesso.

Lo studio della compatibilità tra dispositivi open-hardware in silicio e architetture blockchain si snoda in due punti chiave:

- Report 1: Descrizione e analisi dei dispositivi open-hardware e relativa architettura.
- Report 2: Selezione e studio delle principali primitive crittografiche utili all'implementazione di una blockchain. Analisi e confronto degli algoritmi crittografici, com-

parando le caratteristiche delle diverse primitive con le limitazioni e potenzialità dei dispositivi open-hardware esistenti.

1.2 Definizioni

Acceleratore esterno	Un componente hardware progettato per migliorare le prestazioni della piattaforma che lo ospita. In alcuni ambiti specifici permette di scaricare la CPU da tutte quelle attività che altrimenti occuperebbero buona parte delle risorse computazionali e che possono essere lasciate libere per svolgere più agevolmente altri compiti.
ASIC	Un circuito integrato creato appositamente per risolvere un preciso problema; la specificità della progettazione consente di raggiungere altissime prestazioni in termini di velocità e consumo elettrico.
Blockchain	Registro dati immutabile composto da una catena di blocchi. Ogni blocco contiene un insieme di transazioni e, escluso il primo, un puntatore (tipicamente un hash crittografico) al blocco precedente.
Circuito integrato	Un circuito elettronico miniaturizzato dove i vari transistor sono stati formati tutti nello stesso istante grazie a un unico processo fisico-chimico.
Coprocessore	Una tipologia di processore che si contraddistingue per essere ausiliaria ad un altro processore, ad esempio nei moderni computer la CPU è spesso affiancata da processori ausiliari come la GPU.
Core	Il nucleo elaborativo di un microprocessore. Quest'ultimo infatti è costituito in realtà da 2 componenti principali: il core appunto, e il package che lo contiene. In alcune tipologie di processori è visibile la posizione del core, costituito da un piccolo rettangolo nero leggermente sporgente al centro del package: esso contiene i transistor, che ne determinano funzionamento e capacità di elaborazione.
Fabless	Un modus operandi tipico di alcune aziende, per il quale viene effettuata la progettazione e vendita di dispositivi hardware e circuiti integrati mentre la effettiva fabbricazione viene esternalizzata a una azienda specializzata.

Firma digitale	Un metodo matematico teso a dimostrare l'autenticità di un messaggio o di un documento digitale inviato mediante un canale di comunicazione non sicuro, garantendo al destinatario che il mittente del messaggio sia chi dice di essere (autenticazione), che il mittente non possa negare di averlo inviato (non ripudio) e che il messaggio non sia stato alterato lungo il percorso dal mittente al destinatario (integrità).
Funzione di hash	Funzione matematica che associa a un dato di lunghezza arbitraria un valore (digest) di lunghezza fissa. Le funzioni di hash usate in crittografia devono rispettare ulteriori proprietà di sicurezza.
Logaritmo discreto	Un problema matematico sulla cui difficoltà si basano diversi protocolli crittografici. Dato un gruppo ciclico moltiplicativo G, un generatore g di G e g^a per un certo $a \in \mathbb{Z}$, il problema consiste nel determinare a.
Microprocessore	Una tipologia particolare di circuito elettronico che si contraddistingue per essere interamente costituita da uno o più circuiti integrati e per questo di dimensioni molto ridotte.
Open-Hardware	Termine generico per indicare hardware elettronici che sono stati progettati con la stessa politica del software libero ed open-source.
Periferica	Un qualsiasi dispositivo hardware che fa parte di un sistema informatico e/o di elaborazione elettronica e che funziona sotto il controllo di una unità centrale e del sistema operativo, alla quale è collegata.
PicoRV32	Un core della CPU che implementa il set di istruzioni Risc-V RV32IMC.
Pipeline	Una catena di fasi di elaborazione dei dati. Ogni stadio di questa catena provvede a ricevere in ingresso un dato o un segnale, ad elaborarlo e poi a trasmetterlo all'elemento successivo.
Pipeline in-order	Un metodo di elaborazione dati per cui la CPU esegue le istruzioni in ordine sequenziale e fino al completamento dell'istruzione corrente non eseguirà l'istruzione successiva.
Pipeline out-of-order	Un metodo di elaborazione dati per cui la CPU esegue le istruzioni in ordine non sequenziale: anche se l'istruzione corrente non è completata, eseguirà l'istruzione successiva se possibile.

Primitiva crittografica	Un algoritmo crittografico che viene utilizzato per creare protocolli crittografici per i sistemi di sicurezza.
Risc-V	Architettura ISA sviluppata nel 2010 dall'Università della California con lo scopo di creare un set di istruzioni estensibile e open-source per uso accademico e commerciale.
Transistor	Un dispositivo a semiconduttore largamente usato nell'elettronica analogica.

1.3 Acronimi

ASIC	Application Specific Integrated Circuit.
CBC	Cipher Block Chaining.
CPU	Central Processing Unit.
ECB	Electronic Code Book.
ECDSA	Elliptic Curve Digital Signature Algorithm.
EdDSA	Edwards-curve Digital Signature Algorithm.
ISA	Instruction Set Architecture.
PDK	Process Development Kit.
PRNG	Pseudo Random Number Generator.
RACE	Research and Development in Advanced Communications Technologies.
RIPEMD	RACE integrity Primitives Evaluation Message Digest.
RISC	Reduced Instruction Set Computing.
ROM	Read-Only Memory.
SHA	Secure Hash Algorithm.
SoC	System on a Chip.
SPI	Serial Peripheral Interface.
TRNG	True Random Number Generator.
VGA	Video Graphics Array.

2 Primitive crittografiche utili nelle architetture blockchain

Una blockchain è un registro dati decentralizzato rappresentato da una sequenza espandibile di blocchi che memorizzano le informazioni in una sequenza temporale immutabile, infatti una blockchain cresce con il passare del tempo senza che sia possibile modificarne le informazioni già memorizzate. Il primo blocco di una catena viene chiamato blocco genesi, inoltre i blocchi sono concatenati attraverso delle funzioni di hash, delle mappe particolari per le quali la controimmagine di un generico elemento non è calcolabile in tempi rapidi (vedi la Sottozezione 2.1 per maggiori dettagli), così come mostrato nella Figura 1.

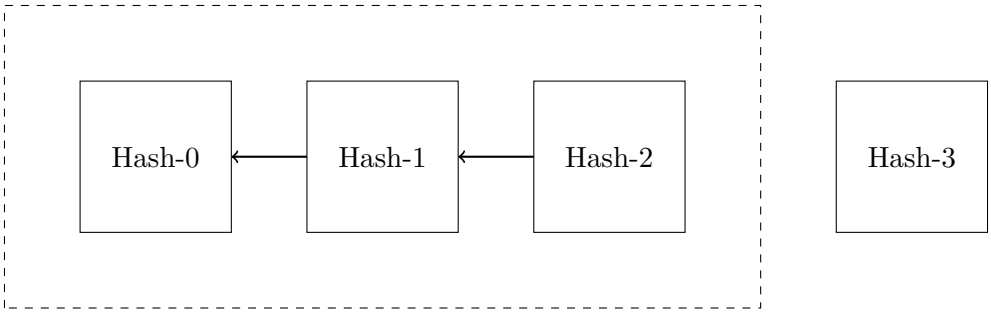

Figura 1: Architettura di una blockchain, dove a sinitra sono rappresentati i blocchi già convalidati, e a destra il nuovo blocco che sta per essere aggiunto.

Ogni singolo blocco del registro contiene svariati dati, fra cui quelli degli utenti (ad esempio i dati transazionali) e un riferimento al valore hash del blocco che lo precede; questo collegamento rende la manipolazione di una blockchain praticamente impossibile. Per comprendere questo aspetto è importante ricordare che questi valori di hash non vengono calcolati unicamente a partire dai dati degli utenti contenuti in un blocco: il calcolo del valore di hash di un nuovo blocco dati include sia il valore di hash del blocco precedente che il cosiddetto *nonce*, un valore che permette di intervenire nel risultato del calcolo stesso di un hash. Questo meccanismo è utilizzato nel contesto di una procedura di consenso nota come Proof Of Work. L'hash di un nuovo blocco viene quindi calcolato considerando il numero di blocco, il nonce, i dati transazionali e l'hash del blocco precedente, garantendo la sicurezza dell'intera architettura.

Oltre alle funzioni di hash, che svolgono un ruolo cruciale nel garantire l'immutabilità della blockchain, ne esistono altre che garantiscono il buon funzionamento della stessa, e per comprenderle è utile analizzare come un utente prende parte alla blockchain e come si relaziona con gli altri partecipanti. Un utente viene identificato nella blockchain con una coppia di chiavi, che consiste di una chiave privata e di una rispettiva (e univocamente determinata) chiave pubblica. Esiste una relazione matematica fra questi due oggetti, permettendo ad ogni utente di usare la propria chiave pubblica per identificarsi agli occhi degli altri partecipanti e di usare la corrispondente chiave privata per firma-

re digitalmente i messaggi creati e pertanto validarli. Sono quindi necessari protocolli di firma digitale, che rendano possibile e sicuro effettuare le operazioni sopra descritte. Nelle Sottosezioni 2.1 e 2.2 vediamo nel dettaglio le due famiglie di primitive descritte sopra, mostrando le dipendenze degli argomenti esposti, e risalendo quindi alle operazioni basilari implementabili da un processore. Infine, la Sottosezione 2.3 è dedicata brevemente ad altre primitive crittografiche, come i commitment, gli accumulatori e gli schemi zero-knowledge proof, che possono essere coinvolte per ottenere blockchain con caratteristiche particolari, quali anonimato e privacy degli utenti e delle corrispondenti operazioni sulla blockchain.

2.1 Funzioni di Hash

Uno strumento fondamentale per la verifica dell'integrità dei dati digitali sono le *funzioni di hash*. Una funzione di hash è una mappa non invertibile che manda una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita, detta *digest*. Una funzione di hash h è considerata *sicura* in ambito crittografico se rispetta le seguenti tre proprietà:

- *resistenza alle collisioni*: è computazionalmente difficile trovare due messaggi X_1 e X_2 tali che $h(X_1) = h(X_2)$;
- *resistenza alle preimmagini*: dato un digest Y, è computazionalmente difficile ricostruire un messaggio X tale che $h(X) = Y$;
- *resistenza alla seconda preimmagine*: dato un messaggio X, è computazionalmente difficile ottenere un secondo messaggio X' tale che $h(X) = h(X')$.

Le funzioni di hash vengono utilizzate nelle architetture blockchain per la creazione degli indirizzi pubblici a partire dalle chiavi pubbliche[1], per risolvere i problemi proposti dalla Proof of Work e negli algoritmi di firma digitale.

Normalmente, una funzione di hash h è un processo iterativo che lavora con input di lunghezza arbitraria ed è costruita a partire da una funzione di compressione f. Il messaggio X viene suddiviso in t blocchi, $X_1, \ldots, X_t$, eventualmente aggiungendo ulteriori bit al messaggio originale fino a che questo abbia una lunghezza divisibile per t (questo processo è chiamato *padding*). Ogni blocco viene utilizzato come input per una funzione di compressione f, assieme al risultato della compressione del blocco precedente. L'intero procedimento può essere schematizzato come segue:

$$\begin{cases} H_0 = IV \\ H_i = f(H_{i-1}, X_i) \qquad \text{per } 1 \leq i \leq t \\ h(X) = H_t, \end{cases} \tag{1}$$

dove H_i indica il risultato parziale al passo$i$-esimo e IV denota il valore iniziale del processo.

[1] Generalmente un indirizzo non coincide con la chiave pubblica.

Le pricipali funzioni di hash usate nelle architetture blockchain sono Secure Hash Algorithm (SHA) e RACE Integrity Primitives Evaluation Message Digest (RIPEMD), in particolare le versioni SHA-256 [1] e RIPEMD-160 [2].

La funzione di hash SHA-256 è stata pubblicata per la prima volta nel 2001 [3] e fa parte della famiglia di SHA-2, che rappresentava lo standard scelto dal NIST fino al 2012. RIPEMD-160 fa parte della famiglia delle funzioni di hash di tipo Merkle-Damgård ed è stata pubblicata per la prima volta nel 1996 [2].

Oltre SHA-256 e RIPEMD-160, diverse funzioni di hash vengono utilizzate nelle blockchain, vedi Tabella 1, ma nel presente report non saranno descritte nel dettaglio, poiché condividono la medesima impostazione di base e, in particolare, le stesse richieste implementative di SHA-256 e RIPEMD-160.

SHA-256 e RIPEMD-160 sono funzioni di hash iterative che operano come descritto in (1). In particolare suddividono l'input in sotto-blocchi che prendono il nome di *parole*. Una parola è una stringa di n-bit che può essere rappresentata come una sequenza di cifre esadecimali. Per convertire una parola in cifre esadecimali, ogni stringa di 4 bit viene convertita nel suo equivalente in cifre esadecimali. Ad esempio, la stringa di 32 bit

$$1010 \quad 0001 \quad 0000 \quad 0011 \quad 1111 \quad 1110 \quad 0010 \quad 0011$$

può essere espressa come in esadecimale come $A103FE23$ o in intero come 2701393443.

Per comprendere il funzionamento di queste primitive è utile introdurre e spiegare il concetto di *aritmetica modulare*. Tale termine indica l'aritmetica che opera su un insieme finito di numeri, ed è contrapposto all'aritmetica ordinaria, la quale opera invece sull'insieme infinito $\mathbb{N}$ dei numeri naturali. L'aritmetica modulare rappresenta quindi un caso particolare di aritmetica finita, ed è nota anche come aritmetica circolare in quanto si presta alla matematizzazione di fenomeni ciclici. Un esempio classico è quello dell'aritmetica modulo 12 (detta talvolta anche aritmetica dell'orologio): in tale sistema si assume che 12 coincide con 0 e l'insieme numerico su cui si opera è costituito dall'insieme finito di numeri $\{0, 1, 2, ..., 11\}$. Il modo di operare corrisponde a situazioni di vita quotidiana per le quali, per esempio, se si aggiungono 3 ore alle 11 antimeridiane, si arriva alle 2 pomeridiane. Dunque, in formula sintetica: $3 + 11 = 2$.
Più rigorosamente, se n è un numero naturale, si dice in generale aritmetica modulo n l'aritmetica che opera sull'insieme $\{0, 1, ..., n-1\}$, nella quale si assume che n coincida con 0: l'intero n è detto modulo dell'aritmetica. Pertanto, in una tale aritmetica risulta per esempio $n + 1 = 0 + 1 = 1$. Più in generale, ogni numero intero m può essere scritto modulo n attraverso l'algoritmo della divisione con resto: se infatti q e r sono rispettivamente il quoziente e il resto della divisione di m per n, vale allora per definizione $m = nq + r$ e dunque, poiché si è posto $n = 0$, si ottiene $m = r$ con r numero intero compreso tra 0 e $n-1$. Più correttamente, si scrive $m \equiv r \mod n$ e si legge m *è congruo a* r *modulo* n. Le regole di calcolo nell'aritmetica modulo n sono piuttosto semplici: è sufficiente eseguire i calcoli normalmente nell'insieme $\mathbb{Z}$ dei numeri interi e riportare il risultato modulo n mediante l'algoritmo della divisione con resto. Per esempio, in un'aritmetica modulo 4 si ha: $11 + 7 = 18 \equiv 2 \mod 4$; in un'aritmetica modulo 5 si ha: $6 \cdot 7 = 42 \equiv 2 \mod 5$.

SHA-256 opera su stringhe di lunghezza multipla di 512 bit (eventualmente facendo il padding del messaggio) e restituisce stringhe di 256 bit (come suggerisce il suo nome). L'algoritmo agisce per un totale di 64 round. La funzione di compressione accetta come input una variabile di concatenamento di 8 parole, una parola chiave per ogni round e una parola di 32-bit estratta dal messaggio di partenza, ed effettua quindi delle operazioni modulo 2^{32}.

RIPEMD-160 prende in input una stringa di lunghezza un multiplo di 512 bit diviso in parole di 32 bit e restituisce un output di 160 bit. L'algoritmo agisce per un totale di 5 round. La funzione di compressione agisce parallelamente su due rami (spesso chiamati destro e sinistro) in cui ogni round è composto da 16 step, per un totale di 80 step nell'intero algoritmo. Anche in questo caso, come in SHA-256, vengono usate operazioni modulo 2^{32} sui cifrati parziali all'interno dell'algoritmo.

Queste funzioni, per poter essere calcolate, richiedono l'impiego delle medesime operazioni di base, che esponiamo di seguito. Chiaramente queste funzioni differiscono poi nell'ordine e nei modi in cui queste operazioni vengono effettuate.

Operazioni richieste A livello implementativo è necessario che un hardware implementi correttamente le operazioni elencate di seguito. Il soddisfacimento di tali richieste è sufficiente a garantire una corretta implementazione del protocollo.

$\wedge$	Operatore AND bit-a-bit.
$\vee$	Operatore OR (OR-inclusivo) bit-a-bit.
$\oplus$	Operatore XOR (OR-esclusivo) bit-a-bit.
$\neg$	Operatore NOT di negazione bit-a-bit.
$+$	Somma modulo 2^{32}. L'operazione $x+y$ è definita come segue: le parole x e y rappresentano i numeri interi X e Y, dove $0 \leq X, Y < 2^{32}$. Per interi positivi U e V, sia $U \pmod V$ il resto della divisione di U per V. Calcoliamo quindi $Z := (X + Y) \pmod{2^{32}}$. Allora $0 \leq Z < 2^{32}$. Convertiamo l'intero Z in una parola z e definiamo infine $z = x + y$.
$<<$	Operatore di shift a sinistra, dove $x << n$ è ottenuto scartando gli n bit più a sinistra della parola x e successivamente facendo un padding del risultato con n zeri a destra.
$>>$	Operatore di shift a destra, dove $x >> n$ è ottenuto scartando gli n bit più a destra della parola x e successivamente facendo un padding del risultato con n zeri a sinistra.
$ROTL^n(x)$	L'operatore di rotazione circolare a sinistra (dove x è una parola di 32 bit e n un intero compreso fra 0 e 32) è dato da $ROTL^n(x) = (x << n) \vee (x >> 32 - n)$
$ROTR^n(x)$	L'operatore di rotazione circolare a destra (dove x è una parola di 32 bit e n un intero compreso fra 0 e 32) è dato da $ROTR^n(x) = (x >> n) \vee (x << 32 - n)$.

$SHR^n(x)$ L'operatore di shift a destra è dato da $SHR^n(x) = x >> n$.

Abbiamo quindi mostrato che per poter interagire con una blockchain un processore deve saper lavorare con le funzioni di hash, in particolare SHA-256 e RIPEMD-160. Per poter garantire una corretta implementazione di queste, è necessario che il processore operi correttamente in tre fasi distinte: dapprima con gli operatori logici (AND, OR, XOR, NOT), poi con l'aritmetica modulare (le operazioni modulo 2^{32}), e infine con gli operatori di shift nei registri del processore. Uno schema riassuntivo è presentato nella Figura 2.

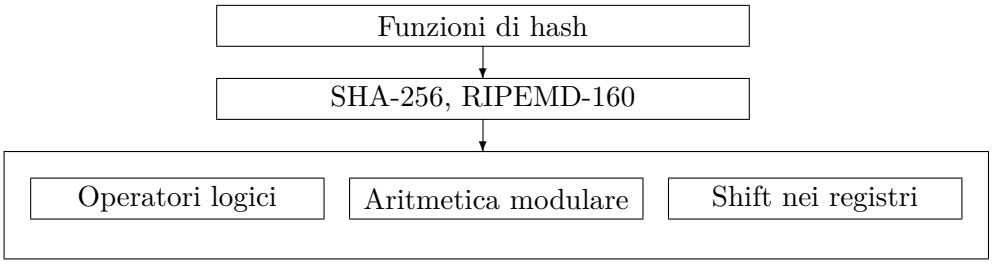

Figura 2: Catena delle dipendenze per le funzioni di hash.

Oltre alle due mappe descritte sopra, che rappresentano la quasi totalità dei modi in cui viene implementata una funzione di hash in una struttura basata su blockchain, ne esistono altre, presentate di seguito nella Tabella 1. In questa tabella sono presentate le trenta crittovalute più diffuse, e per ognuna di queste sono elencate le funzioni di hash che ne regolano il funzionamento.

	SHA-256	Ethash	SCrypt	X11	Equihash	RIPEMD-160	Keccak	Blake	X17	Lyra2rev2	X13	X14	Skein	Qubit	Curl
Bitcoin	•					•									
Ethereum	•	•				•									
Dash	•			•											
Litecoin	•		•			•									
Zcash	•				•										
Zcoin	•														

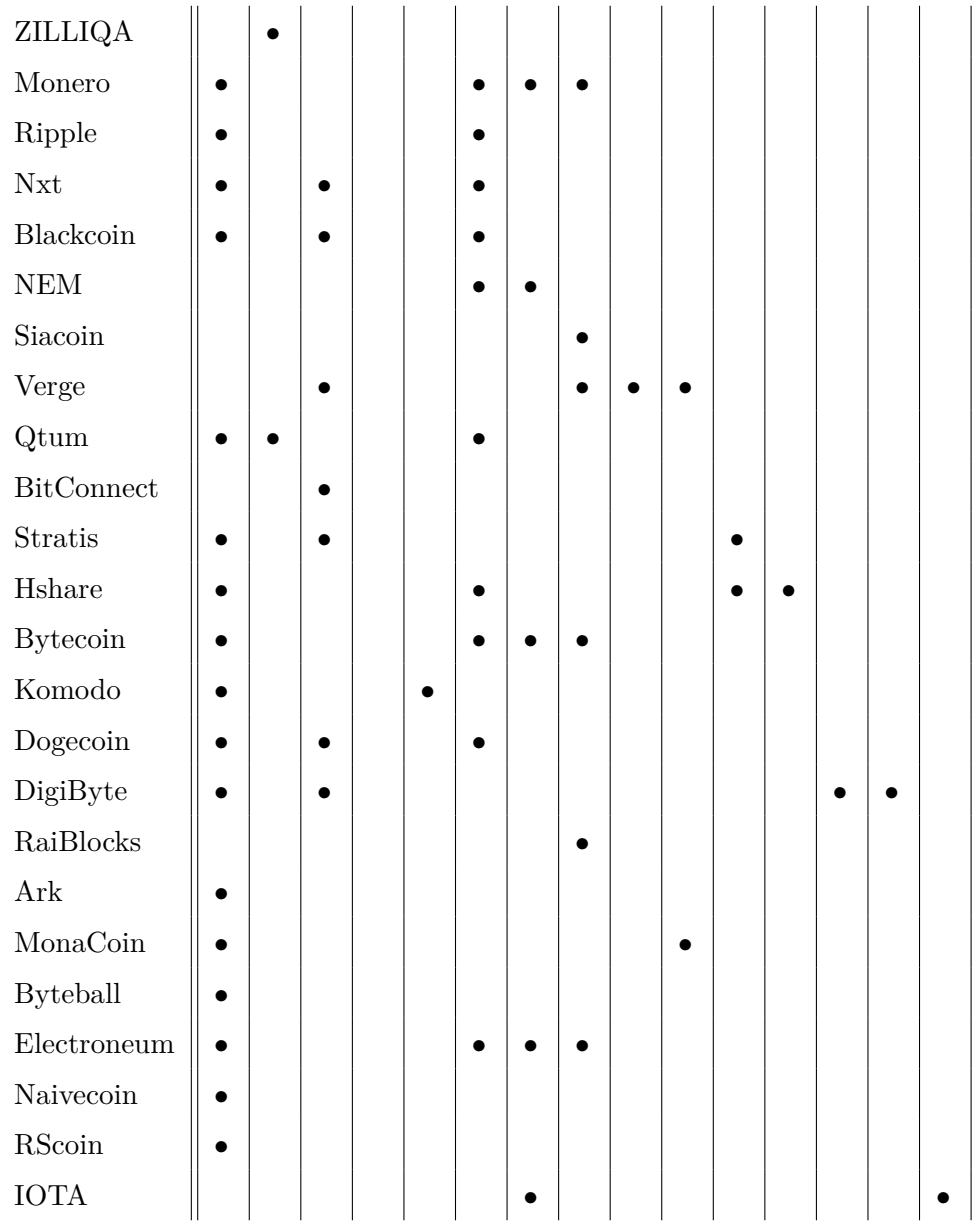

Tabella 1: Panoramica delle funzioni di hash utilizzate nelle 30 crittovalute più usate.

2.2 Firme Digitali

La *firma digitale* è una primitiva crittografica, strumento imprescindibile della blockchain, il cui scopo è quello di sostituire in un contesto digitale una firma autografa tramite un modello matematico. Oggi la firma digitale ha un ruolo fondamentale nella sicurezza di quasi ogni tipo di trasmissione di dati effettuata online attraverso canali non protetti. Le proprietà che un algoritmo di firma digitale deve possedere sono le seguenti:

- *Integrità*: chiunque può verificare che il messaggio non è stato alterato lungo il suo percorso: qualsiasi modifica al messaggio produrrebbe una firma completamente diversa.
- *Autenticità*: finché conserviamo la nostra chiave privata in segreto, gli altri partecipanti possono usare la nostra chiave pubblica per confermare che le firme digitali sono state create da noi e da nessun altro.
- *Non ripudiabilità*: una volta che la firma è stata generata, non potremo in futuro negare di avere firmato, tranne nel caso in cui la chiave privata risulti compromessa in qualche modo.

Queste caratteristiche garantiscono la sicurezza nell'utilizzo delle firme digitali in molte applicazioni quotidiane. La struttura standard di una firma digitale, in accordo con quanto stabilito da Goldwasser, Micali e Rivest in [**?**] è costituita da tre algoritmi da concludersi in tempo polinomiale:

1. *La generazione della chiave*: prende in input un parametro di sicurezza k e restituisce in output la coppia di valori $(\texttt{PK}, \texttt{SK})$, formata da una chiave pubblica `PK` e la corrispondente chiave privata `SK`.
2. *La firma*: prende in input un messaggio m e la chiave privata `SK` e restituisce in output una firma $\sigma(m)$.
3. *La verifica*: prende in input un messaggio m, la firma $\sigma(m)$ relativa ad esso e la chiave pubblica `PK`. L'algoritmo controlla se la firma è valida restituendo in output un valore booleano, che identifica se la firma è stata *accettata* oppure *rifiutata*.

I tre algoritmi devono essere compatibili: la verifica applicata agli input restituiti dagli algoritmi di generazione della chiave e di firma deve restituire il valore booleano corrispondente all'accettazione della firma. In molte applicazioni pratiche, non viene generata direttamente la firma di un messaggio, bensì si combina l'algoritmo di firma con una funzione di hash crittograficamente sicura (paradigma *hash-then-sign*), ossia si genera la firma del digest del messaggio. Un vantaggio consiste nel fatto che le funzioni di hash riducono significativamente la lunghezza del messaggio in input, aumentando quindi l'efficienza del protocollo.

La sicurezza degli algoritmi di firma più comunemente utilizzati si basa, almeno da un punto di vista formale, sul problema matematico del logaritmo discreto. Dato un gruppo ciclico moltiplicativo G e indicato con g un generatore di tale gruppo (questo

significa che ogni elemento del gruppo può essere visto come potenza dell'elemento g), un'istanza del problema del logaritmo discreto corrisponde al seguente problema: dato un elemento g^a di G, per un certo intero a, calcolare a. Il nome di questo problema è dovuto al fatto che la struttura algebrica su cui si opera è costituita da un insieme *finito* di elementi. L'esempio più immediato è quello del logaritmo discreto per l'aritmetica modulare, già descritta in precedenza: data la congruenza $3^k \equiv 13 \mod 17$, risolvere il logaritmo discreto associato equivale a calcolare quale k rende l'espressione vera. Esistono infinite soluzioni intere, per la natura modulare del problema; tuttavia generalmente si cerca la più piccola soluzione non negativa, che in particolare sarà compresa tra 1 e 16, e in questo caso è $k = 4$.

Originariamente, le firme digitali erano basate sull'algoritmo RSA. Successivamente, nel 1991, Kravitz ha pubblicato il *Digital Signature Algorithm* (DSA) [4], e nel 1994 il NIST lo ha adottato come schema standard per le comunicazioni. DSA basa la sua sicurezza sulle elevazioni a potenza modulari e sul problema del logaritmo discreto. Nel 1992 Scott Vanstone ha introdotto l'algoritmo di firma digitale su curve ellittiche (ECDSA) [5].

Nel 1998 è stata accettata come standard ISO (International Standards Organization) (ISO 14888-3), nel 1999 come standard ANSI (American National Standards Institute) (ANSI X9.62) e nel 2000 come standard IEEE (Institute of Electrical and Electronics Engineers) (IEEE 1363-2000) e standard FIPS (FIPS186-2).

Nonostante ECDSA condivida molte analogie con DSA, l'aspetto più importante per il quale l'algoritmo su curve ellittiche è da preferire alla sua versione classica è il fatto che offre lo stesso livello di sicurezza con chiavi pubbliche drasticamente più piccole.

L'algoritmo di firma digitale con curve di Edwards (EdDSA) è una variante della firma di Schnorr [6] sviluppato nel 2012 da Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe e Bo-Yin Yang [7]. La grande innovazione di questo schema è l'impiego di un tipo particolare di curve ellittiche, dette *twisted Edwards curves*, non scritte nella loro forma di Weierstrass.

La maggior parte delle blockchain utilizzate da crittovalute utilizza come algoritmi di firma digitale ECDSA e EdDSA, come mostrato nella Tabella 2.

2.2.1 Curve ellittiche, ECDSA e EdDSA

Le curve ellittiche [8] svolgono un ruolo centrale nella derivazione della chiave pubblica e di conseguenza nel processo di firma digitale. Ai fini di questo report è sufficiente pensare a una curva ellittica come la curva algebrica piana definita dall'equazione di Weierstrass

$$y^2 = x^3 + Ax + B$$

con i coefficienti A, B appartenenti ad un dato campo finito $\mathbb{F}_q$, più un punto speciale, denotato con $\mathcal{O}$ e chiamato *punto all'infinito* o *elemento neutro* della curva. Oltre a questo, è richiesto che la curva sia non singolare, ovvero che non presenti cuspidi o auto-intersezioni. L'insieme dei punti di una curva ellittica può essere dotato di una operazione di somma, creando una struttura algebrica particolare e naturalmente adatta

allo sviluppo di protocolli crittografici. Nel dettaglio: all'interno di una blockchain i partecipanti concordano pubblicamente una curva comune E e un punto P su di essa; ogni utente sceglie la sua chiave privata come un numero casuale k, e calcola la sua chiave pubblica K come

$$K = k \cdot P.$$

Se la curva ellittica è "buona" e se il campo su cui è definita è abbastanza grande, allora la relazione fra le due chiavi è tale per cui è possibile ricavare K da k, ma non il contrario, ossia il problema del logaritmo discreto risulta difficile da risolvere. Questo è inoltre il motivo per cui una chiave pubblica può essere condivisa con chiunque senza rivelare in questo modo la nostra chiave privata. Il primo aspetto implementativo da considerare è quindi la possibilità per un open-hardware di calcolare il prodotto

$$k \cdot P = \underbrace{P + P + \cdots + P + P}_{k \ \ volte}.$$

Per raggiungere questo scopo è sufficiente considerare la fattibilità dell'operazione $P+P$. Ricordiamo quindi brevemente come è definita la somma di due punti: dato un campo finito con q elementi, indicato nel seguito con la notazione $\mathbb{F}_q$, sia E una curva ellittica descritta dall'equazione $y^2 = x^3 + Ax + B$, con coefficienti in $\mathbb{F}_q$. Dati $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ due punti su E, con $P_1, P_2 \neq \mathcal{O}$, la somma $P_1 + P_2 = P_3 = (x_3, y_3)$ è definita nel seguente modo:

- Se $x_1 \neq x_2$, allora

$$(x_3, y_3) = \left(\left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2, \left(\frac{y_2 - y_1}{x_2 - x_1} \right)(x_1 - x_3) - y_1 \right).$$

- Se $x_1 = x_2$ ma $y_1 \neq y_2$, allora $P_1 + P_2 = \mathcal{O}$.
- Se $P_1 = P_2$ e $y_1 \neq \mathcal{O}$, allora

$$(x_3, y_3) = \left(\left(\frac{3x_1^2 + A}{2y_1} \right)^2 - 2x_1, \left(\frac{3x_1^2 + A}{2y_1} \right)(x_1 - x_3) - y_1 \right).$$

- Se $P_1 = P_2$ e $y_1 = 0$, allora $P_1 + P_2 = \mathcal{O}$.

Inoltre $P + \mathcal{O} = \mathcal{O} + P = P$ per tutti i punti P di E.

Risulta quindi chiaro che per realizzare le funzionalità sopra descritte è sufficiente che il processore implementi correttamente l'aritmetica dei campi finiti. È un risultato noto in letteratura [9] che il numero di elementi di un tale campo è necessariamente della forma p^n, dove p è un primo (che costituisce la *caratteristica* del campo) e n è un intero positivo (che costituisce la *dimensione* del campo). Di seguito indicheremo un campo siffatto con la notazione $\mathbb{F}_{p^n}$. Gli elementi di $\mathbb{F}_{p^n}$ possono essere rappresentati come polinomi su $\mathbb{F}_p$ con grado strettamente minore di n. Le operazioni sono quindi effettuate modulo $R(x)$, dove $R(x)$ è un polinomio irriducibile di grado n a coefficienti in $\mathbb{F}_p$, ad esempio utilizzando la divisione polinomiale standard.

Addizione e sottrazione in un campo finito L'addizione e la sottrazione di due numeri in un campo finito della forma $\mathbb{F}_{p^n}$ vengono eseguite sommando o sottraendo i due polinomi che li rappresentano e riducendo il risultato modulo la caratteristica p del campo. Per effettuare i calcoli con i polinomi è opportuno rappresentare questi ultimi tramite dei vettori; la rappresentazione è semplice ed è effettuata nel seguente modo: dato un polinomio per un generico elemento di $\mathbb{F}_{p^n}$, gli si associa un vettore di lunghezza n e i cui elementi siano in corrispondenza con i coefficienti del polinomio stesso. Ad esempio nel campo $\mathbb{F}_{2^8}$ il polinomio 1 è rappresentato dal vettore $\{00000001\}$ mentre il polinomio $(x^6+x^5+x^4+1)$ è rappresentato come $\{01110001\}$. Di seguito è riportato un semplice esempio di somma nel caso in cui il campo sia $\mathbb{F}_{2^8}$, con $R(x) = x^8+x^4+x^3+x+1$.

Polinomi	$(x^6 + x^5 + x^4 + 1) + (x^7 + x^5 + x^2 + x) = x^7 + x^6 + x^4 + x^2 + x + 1$
Rappresentazione	$\{01110001\} + \{10100110\} = \{11010111\}$

Moltiplicazione e divisione in un campo finito La moltiplicazione in un campo finito è la moltiplicazione modulo il polinomio irriducibile utilizzato per definire il campo finito. Detto altrimenti: è la moltiplicazione seguita dalla divisione usando il polinomio $R(x)$ come divisore. Il risultato dell'operazione è il resto di questa ultima divisione. Per quanto riguarda la divisione tra elementi in un campo finito, essa corrisponde alla moltiplicazione per l'inverso modulo p, che può essere calcolato utilizzando l'algoritmo di Euclide esteso. Di seguito è riportato un esempio di moltiplicazione. Il contesto è lo stesso di quello descritto nell'esempio precedente.

Polinomi	$(x^6 + x^5 + x^4 + 1) \cdot (x^7 + x^5 + x^2 + x)$
	$= x^{13} + x^{12} + x^{10} + x^9 + x^8 + 3x^7 + x^2 + x$
	$= R(x) \cdot (x^5 + x^4 + x^2 - 1) + \frac{x^5+1}{R(x)}$
Rappresentazione	$\{01110001\} \cdot \{10100110\} = \{000100001\}$

Abbiamo quindi mostrato che per poter interagire con una blockchain un processore deve saper lavorare con l'aritmetica delle curve ellittiche, e di conseguenza con i campi finiti della forma $\mathbb{F}_{p^n}$; questo si riduce infine al saper operare con polinomi con coefficienti in campi finiti della forma $\mathbb{F}_p$, pertanto è sufficiente che un processore sappia lavorare con l'aritmetica modulare. In aggiunta a ciò, poiché per il corretto funzionamento di questi protocolli è necessario identificare un punto iniziale su una curva, richiederemo la capacità di un processore di saper generare numeri pseudocasuali, per cui è necessario avere un PRNG, il quale può tuttavia essere facilmente implementato a livello software utilizzando uno a scelta dei molti algoritmi presenti in letteratura. In particolare nell'ultimo anno la fondazione Risc-v ha studiato e implementato un TRNG, un hardware per generare numeri casuali [10]. Uno schema riassuntivo di quanto appena esposto è presentato in Figura 3.

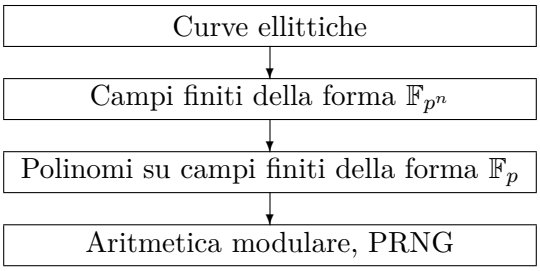

Figura 3: Catena delle dipendenze per le curve ellittiche.

Potrebbe essere utile ricordare che molte implementazioni utilizzano la libreria crittografica OpenSSL [11] per eseguire i calcoli su curve ellittiche. Ad esempio, per derivare la chiave pubblica viene utilizzata la funzione `EC_POINT_mul()`.

I due schemi di firma digitale maggiormente usati in ambito blockchain sono ECDSA e EdDSA. Entrambi basano la loro sicurezza sulla difficoltà di risolvere il problema del logaritmo discreto su curve ellittiche. La differenza fra questi due protocolli è la curva di base su cui si sceglie di lavorare. In particolare, l'equazione di una curva di Edwards su un campo finito $\mathbb{F}_q$ è la seguente:

$$Ax^2 + y^2 = 1 + Dx^2y^2, \qquad \text{dove } A, D \in \mathbb{F}_q \setminus \{0\}.$$

Ci sono varie scelte che possono essere fatte per ottimizzare l'algoritmo di firma EdDSA, e una delle versioni più spesso implementate di EdDSA è Ed25519. Questo schema utilizza la funzione di hash SHA-512 e lavora con una curva birazionalmente equivalente a Curve25519 (che è una curva ellittica nota per le ottime proprietà crittografiche).

Per concludere, a parità di sicurezza fornita, EdDSA si dimostra essere più efficiente di altri algoritmi di firma digitale. Tuttavia, la caratteristica più importante che differenzia ECDSA o DSA da EdDSA, è che in quest'ultimo non è necessario utilizzare un numero casuale univoco per ogni firma, requisito invece fondamentale per la sicurezza di ECDSA o DSA.

2.2.2 Ring Signature e altre firme digitali

Oltre a ECDSA e EdDSA esistono altri algoritmi di firma digitale, alcuni di essi basati su problemi computazionali diversi da quello del logaritmo discreto, i quali permettono di migliorare la privacy e l'anonimità delle transazioni. Un esempio su tutti è quello delle cosiddette *ring signature* ideate da Ronald L. Rivest, Adi Shamir e Yael Tauman nel 2001 [12]. Esse permettono ad un utente di firmare messaggi a nome di un gruppo di partecipanti alla blockchain che include lui stesso, senza però rivelarsi; l'utente può inoltre comporre questo gruppo senza l'autorizzazione degli altri membri. In questo modo non è possibile identificare il firmatario, inoltre nessuno può capire se vengono generate due firme dallo stesso firmatario. Le ring signature permettono da una parte di garantire una perfetta tutela della privacy nei confronti degli utenti, ma dall'altro aprono

anche la strada per scopi illeciti. Questo report non descrive però nel dettaglio tutte queste primitive, dal momento che non sono fondamentali per il corretto funzionamento di una blockchain, bensì vengono utilizzate per raggiungere funzionalità addizionali. Per completezza espositiva tali primitive sono comunque elencate nella Tabella 2.

	ECDSA	EdDSA	Ring	One-Time	Borromean	Multi-signature	EC-Schonrr	EC-KCDSA
Bitcoin	•					•		
Ethereum	•							
Dash	•					•		
Litecoin	•					•		
Zcash	•			•				
Zcoin	•					•		
ZILLIQA						•	•	
Monero		•	•	•	•			
Ripple	•					•		
Nxt								•
Blackcoin	•							
NEM		•				•		
Siacoin		•				•		
Verge	•					•		
Qtum	•					•		
BitConnect	•					•		
Stratis	•					•		
Hshare	•					•		
Bytecoin		•	•	•	•			
Komodo	•			•				
Dogecoin	•					•		
DigiByte	•					•		

RaiBlocks	•								
Ark	•					•			
MonaCoin	•					•			
Byteball	•					•			
Electroneum		•	•	•	•				
Naivecoin		•							
RScoin	•								
IOTA				•		•			

Tabella 2: Firme digitali utilizzate nelle 30 crittovalute più usate.

2.3 Altre Primitive Crittografiche

Oltre alle funzioni di hash e di firma digitale, all'interno delle blockchain è possibile identificare altre primitive, il cui utilizzo non è fondamentale per il corretto funzionamento dell'intera architettura. Possiamo distinguere questi protocolli in tre grandi famiglie: protocolli di commitment, funzioni di accumulazione e zero-knowledge proof.

Uno schema di commitment è una primitiva crittografica che consente ad un utente di scegliere un valore mantenendolo nascosto agli altri, con la possibilità e la capacità di rivelarlo poi in un secondo momento. Gli schemi di commitment sono progettati in modo tale che non si possa modificare il valore o l'affermazione dopo che questa è stata decisa la prima volta. Detto altrimenti, questi schemi sono vincolanti. Le interazioni in uno schema di commitment si svolgono quindi in due fasi: dapprima c'è la fase di *commit* durante la quale un utente si impegna a scegliere un valore specifico. Successivamente, avviene la fase di *rivelazione*, durante la quale il valore viene rivelato e verificato. Un esempio di commitment è il Pedersen commitment [13, 14] che viene utilizzato nella blockchain Monero per permettere agli utenti di nascondere l'importo delle proprie transazioni.

L'accumulatore è una funzione unidirezionale (come le funzioni di hash) che un utente può utilizzare per fornire la prova di appartenenza a un gruppo senza rivelare alcun singolo membro. Questa primitiva crittografica può essere utilizzata per costruire uno schema di commitment e per le ring signature. Si rimanda il lettere a [15, Table 2] per una tabella riassuntiva delle differenti caratteristiche di un accumulatore.

Un'idea naturale per proteggere la privacy e l'anonimato delle transazioni è renderle non collegabili tra di loro. Tuttavia, i sistemi di pagamento elettronico devono verificare se l'operazione richiesta da chi spende il denaro è lecita. A tale scopo sono state introdotte nelle architetture blockchain le *zero-knowledge proof* (ZK-proof) [16]. Queste sono un protocollo tra due utenti, un *prover* e un *verifier*, che permette al primo di convincere

il secondo della validità di un'affermazione senza rivelare nessun'altra informazione. Le ZK-proof sono un elemento fondamentale per gli schemi crittografici in quanto consentono di dimostrare, nel rispetto della privacy dell'utente, che determinate operazioni sono state eseguite correttamente. Tramite l'utilizzo delle ZK-proof, un utente di una blockchain può dimostrare di poter spendere una certa quantità di denaro, senza mostrare effettivamente quanto denaro è in suo possesso. Due esempi di ZK-proof utilizzati sono ZK-SNARK [17] nella blockchain Zerocash e Bulletproofs [18] nelle blockchain Monero ed Elements.

Le primitive menzionate in questa sottosezione esulano però dagli scopi di questo report, essendo considerate come primitive "opzionali" e non fondamentali, motivo per il quale si è deciso di sintetizzarle nella Tabella 3. Per una discussione più approfondita si faccia riferimento a [19] e [20].

	Commitment	Accumulator	ZK-SNARK	Bulletproofs
Bitcoin				•
Ethereum				
Dash				
Litecoin	•	•	•	•
Zcash	•	•		
Zcoin				
ZILLIQA	•			•
Monero				
Ripple				
Nxt				
Blackcoin				
NEM				
Siacoin				
Verge				
Qtum				

	Commitment	Accumulator	ZK-SNARK	Bulletproofs
BitConnect	•	•		
Stratis				
Hshare				
Bytecoin	•			
Komodo	•	•	•	
Dogecoin				
DigiByte				
RaiBlocks				
Ark				
MonaCoin				
Byteball				
Electroneum	•			
Naivecoin				
RScoin				
IOTA				

Tabella 3: Primitive di commitment, di accumulazione e zero-knowledge proof utilizzate nelle 30 crittovalute più usate.

3 Confronto delle primitive con le limitazioni dei dispositivi

Come illustrato nel Report 1, lo stato dell'arte dei dispositivi open-hardware vede Risc-V come core principale, pertanto per comprendere cosa è implementabile in questi chip è necessario innanzitutto considerare le potenzialità espressive di Risc-V. Questo processore racchiude le operazioni matematiche in un insieme minimo di istruzioni con addizione, sottrazione, shift, logica bit-wise e controllo dei rami di esecuzione. Oltre a questo insieme di operazioni ne esistono altri specifici (insiemi di istruzioni per moltiplicazione intera, operazioni in virgola mobile, ...), che tuttavia possono essere simulati (per la maggior parte) dalle operazioni sopra descritte, grazie all'implementazione software.

Le particolarità di Risc-V rendono quindi immediato concludere che sia le primitive crittografiche basate su curve ellittiche, che richiedono di saper lavorare con l'aritmetica modulare, sia le primitive crittografiche basate su funzioni di hash, che invece richiedono anche di saper lavorare con operazioni logiche bit-a-bit e di saper effettuare shift nei registri, sono implementabili a livello software su qualsiasi piattaforma che utilizzi Risc-V come core per processare le istruzioni. Oltre a ciò, nell'ottica più ampia di descrivere lo stato dell'arte a riguardo, dobbiamo tener presente che spesso questi chip vengono ausiliati di alcuni coprocessori specifici per la crittografia. Di seguito descriviamo i vari chip specifici per la crittografia e che lavorano con PULP, indipendentemente dal fatto che il core di quest'ultimo sia OpenRISC oppure Risc-V.

- Halley (2013): un coprocessore sviluppato ad-hoc per le curve ellittiche. Halley è un processore di tipo "application-specific instruction-set" (ASIP), ovvero appartiene a quella tipologia di processori realizzati per risolvere uno specifico problema. In particolare è stato progettato per verificare le firme digitali secondo l'Elliptic Curve Digital Signature Algorithm (ECDSA). Il set di istruzioni fornito da Halley è universale e consente calcoli arbitrari nei campi finiti richiesti per completare la verifica di una firma come definito nello standard NIST B-233. Inoltre, sebbene il set di istruzioni di questa unità sia personalizzato per la verifica delle firme ECDSA, può anche essere utilizzato per generare firme digitali di tipo ECDSA. Per poter eseguire i vari algoritmi basati sulle operazioni nei campi finiti, Halley ospita una memoria a doppia porta da 2 KB per l'archiviazione dei dati e una memoria di istruzioni a porta singola da 6 KB. Usando un cambio di coordinate opportuno[2] per rappresentare i punti della curva ellittica, il chip finale termina una verifica della firma in circa 1.9 Mcicli (a seconda dell'input). Ciò equivale a 11 ms quando il chip lavora alla sua frequenza massima di 167 Mhz, consumando nel frattempo una potenza media di 70 mW.

- Sir10us (2013) [21]: una reimplementazione di Halley, in cui ora il processore è stato sostituito da un core OpenRISC implementato con Or10n[3]. Mentre in Halley

[2]Nel dettaglio: le coordinate proiettive di Lopez-Dahab.

[3]Or10n è un'implementazione del core OpenRISC, sviluppata per interagire meglio con PULP.

il programma deve essere scritto in assembly, in questo chip è possibile utilizzare strumenti di programmazione standard di GNU. Rispetto a Or10n, Sir10us contiene una memoria di istruzioni più grande e una memoria dati più piccola. L'accelerazione di Open-RISC con questo coprocessore accelera la verifica della firma ECDSA di un fattore 15 rispetto a un'implementazione C. La verifica sul coprocessore non solo è più veloce, ma riduce anche la dissipazione di energia di un fattore 21.

- Fulmine (2015): un'implementazione di PULP con 4 core di tipo Sir10us. Il coprocessore (HWCRYPT) implementa un'unità di accelerazione dedicata per una varietà di operazioni crittografiche, sfruttando i vantaggi dell'architettura a memoria condivisa del SoC. L'HWCRYPT si basa su due motori crittografici paralleli, il primo implementa il cifrario a blocchi AES-128 [22] e il secondo implementa la funzione di hash KECCAK-f[400] [23] (una versione più piccola di SHA-3).

Un'attenzione particolare va data al progetto nato dalla cooperazione fra Efabless, Google e SkyWater. L'idea alla base dell'iniziativa è dare la possibilità a chiunque di poter progettare e veder realizzato il proprio chip. Normalmente per veder realizzato questo obiettivo bisognerebbe innanzitutto ottenere l'accesso a un kit di progettazione del processo (PDK) da una casa di produzione, e successivamente bisognerebbe avere una disponibilità economica sufficiente per finanziare la produzione. Per ovviare a questi due problemi Skywater ha fornito un PDK aperto a tutti, chiamato SKY130, mentre per risolvere il secondo punto Google ha fornito e fornirà cicli di produzione di chip completamente gratuiti: il primo nel mese di Novembre 2020, i successivi nel corso del 2021. L'unico vincolo imposto è che il progetto sia completamente open-source. Di seguito elenchiamo i SoC nati in questo modo e che sono utili alla nostra analisi; tutti i core associati sono basati su PicoRV32.

- Crypto Accelerator Chip. Si tratta di un acceleratore crittografico ASIC progettato utilizzando SKY130. L'obiettivo finale di questo progetto è utilizzarlo per applicazioni di sicurezza embedded e IoT. Per quanto riguarda l'architettura, il chip contiene 4 componenti principali: un acceleratore per AES128/256, un acceleratore per SHA-3 e una demo di gioco sperimentale basata su VGA. Per quanto riguarda l'acceleratore di AES esso supporta entrambe le modalità ECB e CBC, senza penalizzare le prestazioni per l'utilizzo di CBC, inoltre offre un buon equilibrio fra spazio occupato e velocità di calcolo in quanto può cifrare o decifrare blocchi di 16 byte in 20 cicli per AES-128 o 28 cicli per AES-256. Per quanto riguarda l'acceleratore di SHA-3, questo può eseguire l'hash di un singolo blocco a 512 bit in 66 cicli, ed è inoltre in grado di elaborare più blocchi immediatamente, ad esempio quando si esegue l'hash di un file di grandi dimensioni.

- SKY130 SHA3 Miner Caravel SoC. Una ASIC realizzata per calcolare gli hash di SHA-3 con l'esplicito riferimento che questo chip è pensato per quelle applicazioni che lavorano con tecnologie blockchain che implementano questo protocollo. Attualmente questo progetto utilizza una pipeline a 12 stadi in quanto questo è il massimo che può essere stipato nello spazio disponibile dello stampo.

- Skywater 130 Decred Miner. Un Soc pensato per lavorare in simbiosi con la blockchain Decred [24]. Il progetto implementa quattro unità di hash per la funzione BLAKE-256r14, ottimizzate per la blockchain Decred (cioè, non un'unità hash BLAKE-256r14 generica). Oltre alle unità di hash il core include anche un'unità SPI con spazio di registro indirizzabile e un dispositivo di interruzione, il tutto da utilizzare con una scheda controller separata. Il core è implementato nel processo SKY130 di Skywater. L'analisi statica della temporizzazione mostra che le unità supportano una frequenza di clock di almeno 50 MHz; a questa frequenza si prevede che quattro unità hash producano un totale di circa 6 MH/s (megahash al secondo).

4 Conclusioni

Questa sezione sintetizza in modo coinciso i punti cruciali di quanto illustrato nelle sezioni precedenti: lo scopo di questo report è quello di studiare le primitive crittografiche fondamentali per l'implementazione di una blockchain; a tal fine sono stati identificati i protocolli fondamentali a riguardo, distinguendo quelli basati su operazioni su curve ellittiche (firme digitali) e quelli basati su funzioni di hash. Per ognuno di questi sono stati analizzati a ritroso i livelli implementativi fino a giungere alle funzionalità base richieste ad un processore affinché tali protocolli possano essere eseguiti correttamente. Nel primo caso l'aritmetica sulle curve ellittiche è stata ricondotta alla semplice aritmetica modulare, più la presenza di un generatore di numeri casuali, nel secondo caso invece l'implementazione di funzioni di hash è stata ricondotta alla fattibilità di effettuare rispettivamente operazioni logiche bit-a-bit, aritmetica modulare e shift di registri. Oltre a SHA-256 e RIPEMD-160, sono state elencate anche le funzioni di hash usate nelle blockchain delle crittovalute più comuni, sottolineando allo stesso tempo come le richieste implementative di queste siano le medesime delle due funzioni già descritte. È comunque importante specificare che le crittovalute rappresentano solo una modalità in cui può essere utilizzata un'architettura blockchain, e non rappresentano in nessun modo la totalità delle applicazioni e delle potenzialità di quest'ultima.

In riferimento all'analisi sui processori open-hardware presentata nel Report 1, qualsiasi SoC che implementi RV32I è in grado, tramite software, di garantire il corretto funzionamento delle primitive crittografiche elencate nella Sezione 2. Inoltre, anche per MB-Lite e OpenRISC, due core non basati su Risc-V, pur non condividendo lo stesso insieme di istruzioni di base di RV32I, risulta fattibile effettuare le operazioni necessarie al corretto funzionamento delle primitive utili alle architetture blockchain.

Osserviamo che la fattibilità di poter effettuare le operazioni necessarie alle primitive crittografiche individuate è per il momento relegata a livello software, tuttavia esistono coprocessori specifici a riguardo, che implementano a livello hardware quelle funzionalità che prima, seppur presenti, dovevano essere implementate a livello software. È il caso di Halley e Sir10us, due coprocessori specifici per l'algoritmo di forma digitale ECDSA, Fulmine, specifico invece per AES-128 e Keccak, così come alcuni SKY130 SHA3 Miner Caravel, pensato per calcolare gli hash di SHA-3, Crypto Accelerator Chip, pensato per accelerare il calcolo di AES-128 e AES-256, e di Skywater 130 Decred Miner, pensato per lavorare con la funzione di firma BLAKE.

5 Riferimenti bibliografici

[1] Quynh H DANG. Secure hash standard. 2015.

[2] Hans DOBBERTIN, Antoon BOSSELAERS, and Bart PRENEEL. Ripemd-160: A strengthened version of ripemd. *International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, p. 71-82*, 1996.

[3] Nist. secure hash standard. *https://csrc.nist.gov/csrc/media/publications/fips/180/2/archive/2002-08-01/documents/fips180-2withchangenotice.pdf.*, 2002.

[4] David W KRAVITZ. Digital signature algorithm. *US Patent, 5.231: 668*, 1993.

[5] Don JOHNSON, Alfred MENEZES, and Scott VANSTONE. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security, 1.1: 36-63*, 2001.

[6] Claus P. SCHNORR. Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system. *U.S. Patent No 4,995,082.*, 1991.

[7] Daniel J. BERNSTEIN et al. High-speed high-security signatures. *Journal of cryptographic engineering, 2.2: 77-89*, 2012.

[8] Lawrence C. WASHINGTON. *Elliptic curves: number theory and cryptography.* 2008.

[9] Rudolf LIDL and Harald NIEDERREITER. *Introduction to finite fields and their applications.* 1994.

[10] Markku-Juhani O. SAARINEN, G. Richard NEWELL, and Ben MARSHALL. Building a modern trng: An entropy source interface for risc-v. *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security. p. 93-102*, 2020.

[11] John VIEGA, Matt MESSIER, and Pravir CHANDRA. Network security with openssl: cryptography for secure communications. *O'Reilly Media, Inc.*, 2002.

[12] Ronald L. RIVEST, Adi SHAMIR, and Yael TAUMAN. How to leak a secret. *International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, p. 552-565*, 2001.

[13] Torben Pryds PEDERSEN. Non-interactive and information-theoretic secure verifiable secret sharing. *Annual international cryptology conference. Springer, Berlin, Heidelberg, p. 129-140.*, 1991.

[14] David DERLER, Christian HANSER, and Daniel SLAMANIG. Revisiting cryptographic accumulators, additional properties and relations to other primitives. *Cryptographers' track at the rsa conference. Springer, Cham, p. 127-144.*, 2015.

[15] Licheng WANG et al. Cryptographic primitives in blockchains. *Journal of Network and Computer Applications, 127: 43-58*, 2019.

[16] Shafi GOLDWASSER, Silvio MICALI, and Charles RACKOFF. The knowledge complexity of interactive proof systems. *SIAM Journal on computing, 18.1: 186-208.*, 1989.

[17] Georg FUCHSBAUER. Subversion-zero-knowledge snarks. *IACR International Workshop on Public Key Cryptography. Springer, Cham, p. 315-347.*, 2018.

[18] Benedikt BÜNZ et al. Bulletproofs: Short proofs for confidential transactions and more. *2018 IEEE Symposium on Security and Privacy (SP). IEEE, p. 315-334*, 2018.

[19] Amit K. CHOPRA and Munindar P. SINGH. Contextualizing commitment protocol. *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems p. 1345-1352*, 2006.

[20] Josh BENALOH and Michael DE MARE. One-way accumulators: A decentralized alternative to digital signatures. *Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, p. 274-285*, 1993.

[21] Michael GAUTSCHI et al. Sir10us: A tightly coupled elliptic-curve cryptography co-processor for the openrisc. *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors. IEEE, p. 25-29*, 2014.

[22] Simon HERON. Advanced encryption standard (aes). *Network Security, 2009.12: 8-12*, 2009.

[23] Guido BERTONI et al. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 2009.

[24] Christina EPSON. Dtb001: Decred technical brief. *Available at https://coss.io/documents/white-papers/decred. pdf Additional information available at https://www.decred.org*, 2015.