Principles of Cryptocurrency Design

Appunti ed Esercizi

Francesco Pasquale

21 marzo 2024

1 Funzioni Hash Crittografiche

Con il programma seguente (o con qualunque altra implementazione della funzione hash crittografica sha256) potete verificare che l'hash espresso in esadecimale della stringa *Francesco 16114492071* inizia con una sequenza di 9 zeri.

```
from hashlib import sha256

nonce = '16114492071'
text = 'Francesco ' + nonce

print(sha256(text.encode('utf8')).hexdigest())
```

Esercizio 1. Scrivere un programma che trovi una stringa `<nonce>` tale che l'hash della stringa `<nome> <nonce>`, per il vostro `<nome>` inizi con una sequenza di 9 zeri. Stimare il *running time* del programma ed eseguirlo.

Le due linee seguenti sono un estratto di un file `/etc/shadow` contenente gli hash delle password degli utenti di un sistema GNU/Linux

```
satoshi:$y$j9T$2wTxPAjXwsoqXJc2eiYbb1$iERNSCCzND12k9zqBms.CqAC7CtzaGQhJnao/53Jjh2:
nakamoto:$y$j9T$RrPRoX82jNdhTLDHuVmhY1$UIzs5jXCaEiB8lgXyBbQ8uwiOYoulfguzeTl2mPUx41:
```

Dalla pagina di manuale shadow vediamo che in ogni riga i campi sono separati da ":" (nel caso in questione sono omessi tutti i campi esclusi i primi due). Il primo campo è lo *username* dell'utente. Il secondo campo è a sua volta diviso in tre parti separate dal simbolo $: la prima indica lo schema utilizzato per ottenere l'hash della password, in questo caso `y` indica che lo schema utilizzato è yescrypt. Le altre tre parti indicano rispettivamente i parametri passati allo schema, il *salt* e l'hash ottenuto.

Nonostante questo sistema di memorizzazione degli hash delle password sia molto sicuro, gli utenti possono sempre scegliere password "deboli". Nell'esempio in questione infatti uno dei due utenti ha scelto una password adeguatamente complessa, l'altro ha scelto una password molto debole.

Esercizio 2. Usando un opportuno software per il *password cracking* (per esempio, John the Ripper) determinare quale dei due utenti ha una password debole.

2 Byzantine Broadcast e State Machine Replication

Si consideri un sistema distribuito con n nodi in cui ogni nodo può eseguire computazioni locali arbitrarie e inviare messaggi di lunghezza arbitraria a ogni altro nodo. Degli n nodi, f sono *corrotti*

e $n - f$ nodi sono *onesti*. Assumiamo di essere in un sistema *sincrono* in cui c'è un *global clock* noto a tutti i nodi che scandisce il tempo in *round* discreti e ogni messaggio inviato in un round t arriva a destinazione prima dell'inizio del round $t + 1$.

Consideriamo il problema seguente, che chiamiamo *State Machine Replication*: ad ogni nodo $i \in [n]$, in ogni round $r \in \mathbb{N}$, può essere affidata una o più *transazioni* `tx` (stringhe binarie). Ogni nodo i mantiene un *log* che consiste in una concatenazione di transazioni, indichiamo con LOG_i^r il log del nodo i al round r. Vogliamo progettare un protocollo che faccia in modo che l'evoluzione dei log dei nodi nel tempo soddifi le due proprietà seguenti

- **Consistency**: Per ogni $i, j \in [n]$ e per ogni coppia di round $r, s \in \mathbb{N}$, se i e j sono nodi onesti allora $\mathrm{LOG}_i^r \preccurlyeq \mathrm{LOG}_j^s$ oppure $\mathrm{LOG}_j^s \preccurlyeq \mathrm{LOG}_i^r$, dove con la notazione $\mathrm{LOG} \preccurlyeq \mathrm{LOG}'$ intendiamo che LOG è un prefisso di LOG'.
- **Liveness**: Esiste un *confirmation time* $T_{\mathrm{conf}} \in \mathbb{N}$ tale che, se una transazione `tx` viene affidata a un nodo onesto in un round $r \in \mathbb{N}$, allora per ogni nodo onesto $i \in [n]$, $\texttt{tx} \in \mathrm{LOG}_i^{r+T_{\mathrm{conf}}}$.

Esercizio 3. Mostrare che se abbiamo un protocollo Π_{BB} che risolve *Byzantine Broadcast* in R round, allora possiamo progettare un protocollo Π_{SMR} per *State Machine Replication* con confirmation time $T_{\mathrm{conf}} = \mathcal{O}(nR)$.