Primitive Avanzate

Riccardo Longo

Protocolli Crittografici

- Sistemi complessi basati su **primitive**
- Hanno l'obiettivo di mettere in sicurezza una serie di operazioni
  - **Comunicazione** / scambio di dati
  - **Gestione** di dati
  - **Manipolazione** di dati
- Oltre alla **sicurezza** sono importanti la **correttezza** e l'**efficienza**

Primitive Crittografiche

- Scambio di chiavi
- Autenticazione
- Cifratura
- Non-ripudio
- Secure Multi-Party Computation
- Secret Sharing
- Zero-Knowledge Proof

Primitive e Protocolli

- Un protocollo può essere composto da altri protocolli
- Allo stesso modo alcune primitive sono in realtà basate su altre primitive
  - Hash
  - Generatori pseudorandom
  - One-way function
- Nelle costruzioni composte a volte si può dimostrare la sicurezza se i blocchi di base sono sicuri

Secure Multi-Party Computation

- Vari partecipanti $1 \leq i \leq n$, alcuni possono essere malevoli
- Ognuno conosce un certo input segreto x_i
- L'obiettivo è calcolare il risultato di una funzione su questi input:

$$y = F(x_1, \ldots, x_n)$$

  senza però rivelare i segreti x_i
- I requisiti sono:
  - **Privacy**: non deve essere svelata nessuna informazione sugli x_i (a parte ciò che consegue da y)
  - **Correttezza**: anche in presenza di una frazione di attaccanti che partecipano attivamente al calcolo y dev'essere corretto (o l'operazione deve fallire)

Secret Sharing

- Un segreto viene **condiviso** tra più utenti
- Ognuno riceve una quota o **share**
- Il segreto può essere ricostruito solo se una certa **soglia** di quote viene combinata (t su n)
- **Secure sharing**
  - Se il numero di quote è inferiore alla soglia non è possibile ottenere **alcuna informazione** sul segreto
  - Se la soglia è raggiunta il segreto è completamente ricostruibile (**tutto o niente**)

Applicazioni del Secret Sharing

- **Resilienza**: il segreto è salvato in più posti e quindi si può recuperare anche se qualcuno di questi non è più accessibile
  - Chiavi di backup
- **Sicurezza**: nessun utente singolo detiene il segreto completo, quindi non può agire da solo.
  - Codici di lancio

Trivial sharing

- $\mathbf{t = 1}$: il segreto viene semplicemente distribuito tra tutti gli utenti
- $\mathbf{t = n}$: supponendo che il segreto sia la stringa binaria $s \in \{0,1\}^\ell$
  - Ad ogni utente $1 \leq i < n$ viene affidato lo share

    $$x_i = r_i \in \{0,1\}^\ell$$

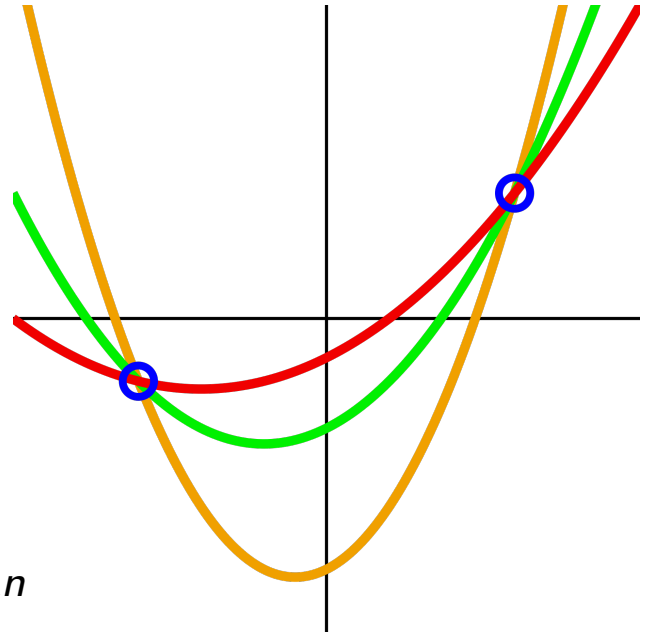
    con r_i scelto casualmente
  - All'utente n viene affidato lo share

    $$x_n = s \oplus r_1 \oplus \ldots \oplus r_{n-1}$$

Interpolazione polinomiale

- Un polinomio di grado n è completamente definito dal passaggio per $n+1$ punti
- Dati n punti qualsiasi passano infiniti polinomi di grado n
- Un polinomio è definito dai suoi coefficienti:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$



Shamir's Secret Sharing

- Basato sull'interpolazione di polinomi
- I conti vengono fatti modulo p, un primo grande
- Il segreto è un numero $0 \leq s < p$
- Per dividere un segreto in modo che servano t share per ricostruire il segreto viene usato un polinomio di grado $t-1$:

$$P(x) = s + \sum_{i=1}^{t-1} a_i x^i$$

- I coefficienti a_i sono scelti random in $\mathbb{Z}_p$, per $1 \leq i \leq t-1$
- Gli share sono semplicemente la valutazione del polinomio negli interi:

$$x_i = P(i)$$

Proprietà SSS

- Per ricostruire il segreto dati t share x_i si fa una interpolazione: si calcola il polinomio passante per i punti

$$(i, x_i) \in P \Rightarrow P(i) = x_i \Rightarrow s = P(0)$$

- Si possono creare quante share si vogliono
- Si possono aggiornare le share (invalidando quelle vecchie) senza cambiare il segreto, usando un polinomio tale che $P(0) = 0$
- Gli share possono essere essenzialmente della stessa dimensione del segreto

Zero Knowledge Proof

- Dimostrare la **conoscenza** di un segreto senza rivelare nulla del segreto stesso
- Interazione tra Peggy (**prover**) e Victor (**verifier**)
- I partecipanti e **solo loro** sono convinti della dimostrazione
- È possibile **simulare** l'interazione tra P e V senza conoscere il segreto

Applicazioni

- **Autenticazione**: verificare la conoscenza di un segreto (password) senza comprometterne la confidenzialità
- **Assicurazione etica**:
  - I partecipanti al protocollo dimostrano di seguire correttamente le istruzioni
  - I segreti non sono rivelati
  - Si ha la prova che ognuno si comporta onestamente

Proprietà ZKP

- **Completezza**: se l'affermazione è vera, un dimostratore onesto potrà convincere del fatto un verificatore onesto
- **Correttezza**: se l'affermazione è falsa, la probabilità di riuscire a convincere un verificatore onesto può essere resa bassa a piacere
- **Zero-Knowledge**: dall'interazione il verificatore non impara nulla se non la veridicità dell'informazione. Cioè è sempre possibile simulare una interazione, indistinguibile da una autentica, sapendo solo l'affermazione da dimostrare.

Esempio: ZKP del DLOG

- Dato y, Peggy vuole dimostrare a Victor la conoscenza di $x : y = g^x$
- Peggy sceglie r random, calcola $C = g^r$ e manda C a Victor
- Victor sceglie casualmente se chiedere a Peggy il valore r o $x + r$
- Peggy risponde col valore appropriato z, Victor verifica:
  - Calcolando $g^z \stackrel{?}{=} C$ se ha richiesto r
  - Calcolando $g^z \stackrel{?}{=} C \cdot y$ se ha richiesto $x + r$
- Ripetendo l'esperimento la probabilità di imbrogliare va a 0
- Se Peggy risucisse a prevedere la richiesta di Victor potrebbe imbrogliare: $C' = g^{r'} \cdot y^{-1} = g^r \Rightarrow r' = x + r$