Cryptographic Protocols

Chapter 1

Introduction

- Computing with encrypted data
- Authentication without giving away data
- Cryptographic voting protocol
- Blockchains that respect privacy
- Generate a random number that cannot be biased
- Sealed-bid auction without a trusted auctioneer

1.1 Examples

1.1.1 Generate a random bit (coin flip)

- Without cryptography, this is not possible among **two** parties
- Use a cryptographic hash-function $\mathbb{H}$
  - $\mathbb{H}$ is collision-free (& one way)
  - $\mathbb{H}$ could be SHA-2

  (special case of commitment)

Why secure?

- because c does not reveal anything about a, therefore Bob cannot bias b
- because Alice cannot find two distinct $(a, x) \neq (\tilde{a}, \tilde{x})$, s.t.:

$$\mathbb{H}(a\|x) = \mathbb{H}(\tilde{a}\|\tilde{x}) = c,$$

  she cannot change her bit (or will be caught)
- if one of the parties is honest (pick bit uniformly at random) the resultin $a \oplus b$ is uniform

1.1.2 Millionaire's Problem

A and B want to find out who is richer, but not leak more about their wealth

This is easy with a trusted (third) party $\mathbb{T}$

Examples

- auctions
- elections
- matchings

No easy solution here!

1.2 Why do computers see their data and programs?

$\rightsquigarrow$ Trusted computing base

1.3 Examples

1.3.1 Computing with encrypted data

$$y = Dec(sk, Eval(pk, f, Enc(pk, x))) = f(x)$$

$\rightsquigarrow$ homomorphic encryption

1.3.2 Secret vote among three

Parties $p_1,\ p_2,\ p_3$
Each has one binary vote v_i on a proposal
Goal is to compute privately $s = \sum_i v_i$ and not disclose more information about $v_1,\ v_2,\ v_3$ than follows from s

Protocol

- <u>primitive</u> $split(b) \rightarrow (x_1, x_2, x_3)$ to "share" or distribute bit b among 3 parties
- use prime p (e.g. 7)

  <u>b</u>

  $x_1 \leftarrow \mathbb{Z}_p$
  $x_2 \leftarrow \mathbb{Z}_p$
  $x_3 \leftarrow \mathbb{Z}_p$ s.t. $x_1 + x_2 + x_3 \equiv b (mod p)$
  return (x_1, x_2, x_3)

Parties are connected by secure channels (confidential & authenticated)

Protocol for p_i (v_i)

$(x_{i1}, x_{i2}, x_{i3}) \leftarrow \mathit{split}(v_i)$
<u>send</u> x_{ij} to p_j for $j = 1, 2, 3$
<u>receive</u> x_{ji} from p_j for $j = 1, 2, 3$
$y_i \leftarrow (x_{1i} + x_{1i} + x_{1i}) \bmod p$
<u>send</u> y_i to p_j for $j = 1, 2, 3$
<u>receiv</u> y_j to p_j for $j = 1, 2, 3$
<u>output</u> $(y_1 + y_2 + y_3) \bmod p$

Completeness

If every party follows the protocol, then every party outputs $s = v_1 + v_2 + v_3$

$$s \equiv \sum_j y_j$$

Security

1. $\mathit{split}(b) \rightarrow (x_1, x_2, x_3)$ hides b because any two values give no information on b ($\rightarrow$ one-time-pad)
2. given s and x_{ji} for $j = 1, 2, 3$ then party p_i has no more information about v_j for $j \neq i$ than what follows from s and v_i

1.4 Goals

1.4.1 Privacy

No party learns more information than the output
$\Rightarrow$ information contains its own input, output, protocol messages ... as if completed by a trusted party $\mathbb{T}$

1.4.2 Correctness

Every party receives the correct output
$\Rightarrow$ if inpit of a faulty party is not clear the protocol computes a consistent output for all correct parties

1.4.3 Input Independence

Inputs of faulty parties must not depend in any way on inputs of correct parties

1.4.4 Fairness

Faulty parties receive output if and only if correct parties received an output
$\rightsquigarrow$ fair contract signing

1.5 Types of Faults

- All faulty parties are controlled by an adversary $\mathbb{A}$
- Semi-Honest Behaviour:
  - Faulty parties execute protocol correctly, but leak all interval values to $\mathbb{A}$
  - "read-only" attack on a server
  - "passive-corruption"
- Malicious Behaviour:
  - Faulty parties behave arbitrarily, act against the correct parties
  - coordinated attack by $\mathbb{A}$

1.6 Types of Computations

- Any polynomial-time computable program
- Usually, without interaction with users, only with initial inputs
  $\rightarrow$ secure function evaluation
- Usually, with one common output (public)... but individual outputs would be possible