SMC

Ivan Sendin

FACOM - Universidade Federal de Uberlândia
ivansendin@yahoo.com,sendin@ufu.br

16 de outubro de 2024

SMC

- Problema do Milionario
  Avaliar se $m_1 < m_2$
- Problema do Casamento
  Avaliar $a \wedge b$
- Sem revelar $a, b, m_1, m_2, \ldots$ para os participantes, observadores,etc...
- (Alguma informação é obtida)

- Computação Segura Multiparte
- SMC
- É um modelo de computação onde os dados dos participantes ficam "seguros"

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- Pode virar um circuito
- um algoritmo
- Uma função
- E Vice-Versa

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Uma tabela verdade é/define uma **computação**

Pode ser mais complexa!! Por exemplo $>$ para qualquer tamanho de número

A	B	$A \wedge B$
1	0	0
0	1	0
1	1	1
0	0	0

- $\mathcal{A}$ escolhe um valor (2 linhas)
- $\mathcal{B}$ escolhe um valor (2 linhas)
- Intersecção $\rightarrow$ computação feita!!
- Não importa a ordem das linhas (veja que eu ja mudei)

$\mathcal{A}$ faz o commit da tabela permutada, escondendo a **sua** entrada e a saída.

A	B	$A \wedge B$
$\mathcal{C}_{\mathcal{A}}(1)$	0	$\mathcal{C}_{\mathcal{A}}(0)$
$\mathcal{C}_{\mathcal{A}}(0)$	1	$\mathcal{C}_{\mathcal{A}}(0)$
$\mathcal{C}_{\mathcal{A}}(1)$	1	$\mathcal{C}_{\mathcal{A}}(1)$
$\mathcal{C}_{\mathcal{A}}(0)$	0	$\mathcal{C}_{\mathcal{A}}(0)$

B faz outra permutação (omitida) e o commit:

A	B	$A \wedge B$
$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(1))$	$\mathcal{C}_{\mathcal{B}}(0)$	$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(0))$
$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(0))$	$\mathcal{C}_{\mathcal{B}}(1)$	$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(0))$
$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(1))$	$\mathcal{C}_{\mathcal{B}}(1)$	$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(1))$
$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(0))$	$\mathcal{C}_{\mathcal{B}}(0)$	$\mathcal{C}_{\mathcal{B}}(\mathcal{C}_{\mathcal{A}}(0))$

- Agora, os participantes escolhem (com commit) as linhas de acordo com a entrada
- $\mathcal{A}$ (1,3) ou (2,4); $\mathcal{B}$ (1,4) ou (2,3)
  (Neste exemplo, não houve permutação por $\mathcal{B}$...mais fácil para entender...)
- A linha em comum é o resultado da computação...
- O commit é revelado
  Somente o commit do resultado

- Se os participantes fizerem isso em publico e o resultado for 0
- Tanto A quanto B podem "dizer" que não queriam casar!!
- Se for 1...

- E se alguem trapacear??
- (exemplo: A pode gerar uma saída so com 1s....)
- O protocolo tem dois caminhos:
  1. Computação (visto)
  2. Verificação de Honestidade
     "Duvido"
- O comportamento desonesto pode ser punido com multas
- (juíz ou SC)
- Smart Contracts como uma plataforma para computação segura
  Bianca Cristina da Silva

3 criptografos estão jantando. Ao final da noite, o garçom informa que a conta já foi paga.
Se foi um dos 3 participantes que pagou a conta...ok.
Mas se foi um agente da ABIN que pagou a conta, pode haver um constrangimento.
E é claro que eles não revelar facinho...
(Este problema não é similar ao jantar do filosofos...meio nojento)

Preparação

Cada par de filosofos produz um bit aleatório compartilhado.
(joga um moeda...escondido)

$$b_{AB}, b_{BC}, b_{AC}$$

Execução

Cada participante calcula o $\oplus$ dos seus bits.
Por, exemplo o criptografo A calcula $b_A = b_{AB} \oplus b_{AC}$.
Cada participante publica:

- b_n se não pagou a conta;
- $\bar{b_n}$ se pagou a conta.

Fim...

Com todos os b_n's publicos podem constatar que:

$$(b_{AB} \oplus b_{AC}) \oplus (b_{AB} \oplus b_{BC}) \oplus (b_{AC} \oplus b_{BC}) == 0$$

Logo, se um dos termos estiver negado....tenho um **OU-Exclusivo** para a frase "Eu paguei a conta..."

- DC-net
- Participantes honestos
- Restaurante honesto

- Sorteio de uma moeda privativa aos pares...
- O resto é trivial

```
pragma solidity >=0.4.25 <0.6.0;

/// @author Ivan Sendin
/// @title Simple Diffie Hellman Implementation

library DH {

///
struct DHPair {
        uint p;
        uint g;
        address a;
        uint ga;
        address b;
        uint gb;
}



/*modifier onlyParticipants(string memory n) {
          DHPair memory dhp = dhKeys[n];
          require ( msg.sender == dhp.a ||  msg.sender == dhp.b);
          _;
}
*/
```

```
/// Creates a struct to handle a DH info
/// @param _p the prime used in modulos
/// @param _g the generato
function createsDHPair( DHPair storage dhp, uint _p, uint _g) public {
        dhp.p= _p;
        dhp.g = _g;
        dhp.a = msg.sender;
        //dhKeys[n] = dhp;
}

/*
/// The information os the second participant
/// @param n the name to access the data
/// @param _gb g powered to b
function completeDHPair(string memory n, uint _gb) public {
        DHPair memory dhp = dhKeys[n];
        dhp.gb = _gb;
        dhp.b = msg.sender;
        dhKeys[n] = dhp;
}
```

```
/// Gets the prime number used in modulos operations
/// @param n the name to access the data
/// @return stored value
function getPrime(string memory n) public returns (uint) {
        DHPair memory dhp = dhKeys[n];
        return dhp.p;
}

/// Gets the generator
/// @param n the name to access the data
/// @return stored value
function getGenerator(string memory n) public returns (uint) {
        DHPair memory dhp = dhKeys[n];
        return dhp.g;
}
```

```
/// Gets the information about the other user
/// @param n the name to access the data
/// @return g powered to a secret value
function getOther(string memory n) public onlyParticipants(n) returns (uint)  {
        DHPair memory dhp = dhKeys[n];
        if (msg.sender == dhp.a)
           return dhp.gb;
        if (msg.sender == dhp.b)
           return dhp.ga;
}
*/

}
```

```
pragma solidity >=0.4.25 <0.6.0;


import {DH} from "./DH.sol";

contract DC {

  using DH for DH.DHPair;

  address crypto1;
  address crypto2;
  address crypto3;

  bool b1;
  bool b2;
  bool b3;


  mapping (string => DH.DHPair) dh;
```

```
constructor(address c1, address c2, address c3, uint256 g, uint256 p) public {
   crypto1 = c1;
   crypto2 = c2;
   crypto3 = c3;
   dh["12"].createsDHPair(p,g);
   dh["13"].createsDHPair(p,g);
   dh["23"].createsDHPair(p,g);
 }
```

```
function initDH(string memory id, uint256 ga) public  {
   require (dh[id].g >0 ,"This DH doest not exist!");

   if (dh[id].ga==0) {
     dh[id].a = msg.sender;
     dh[id].ga  = ga;
     return;
   }

   if (dh[id].gb==0) {
     dh[id].b = msg.sender;
     dh[id].gb  = ga;
     return;
   }

   require(false, "This shouldnt happened...");
   return;
 }
```

```
function getDHValue(string memory id) public returns (uint256) {
    require (dh[id].g >0 ,"This DH doest not exist!");
    if (dh[id].a == msg.sender) {
      return dh[id].gb;
    }
    if (dh[id].b == msg.sender) {
      return dh[id].ga;
    }
    require(false, "This shouldnt happened...");
    return 0;

  }
```

```
function revealBit(bool v) public {
    if (msg.sender == crypto1) {
      b1=v;
      return;
    }
    if (msg.sender == crypto2) {
      b2=v;
      return;
    }
    if (msg.sender == crypto3) {
      b3=v;
      return;
    }
    require(false, "This shouldnt happened...");
    return;

  }
```

```
function wasPaid() public view returns (bool) {
   return (b1 !=b2) != b3;
 }
```