Commit

Ivan Sendin

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

8 de outubro de 2024

Dois jogadores de xadrez mental ficaram entediados.

"Vamos jogar poker mental", disse o primeiro.

"Sim. Eu dou as cartas!", respondeu o outro.

Shamir, Rivest, Adleman; Mental Poker

Compromissos - Commitments

- *Commitment Schemas*
- Eu preciso me **comprometer** com uma informação
- Eu não posso/quero revelar a informação **no momento**
- Manuel Blum em Coin Flipping by Telephone
- Shamir, Rivest, Adleman em Mental Poker

Compromissos - Commitments

- Um *Commitment Schemas* tem duas fases:
  1. Compromisso
  2. Abertura ou revelação

Compromissos - Commitments

- (Compromisso) Alice conhece M e gera $h = \mathcal{H}(M)$
- Alice envia h para Bob
- (Abertura) Em algum momento do futuro Alice envia M' para Bob
- Bob verifica se $M' == M$
- Fim!!

Compromissos - Commitments

- Bob tem condições de saber se Alice teve um comportamento honesto??
- $M \stackrel{?}{=} M'$
- $h \stackrel{?}{=} \mathcal{H}(M')$

Compromissos - Commitments

- **Bob consegue determinar M?**
- Não, propriedade de unidirecionalidade de $\mathcal{H}$
- **Bob tem condições de saber se $M == M'$??**
- Sim, propriedade de resistencia a colisão de $\mathcal{H}$
- **Bob tem condições de gerar um M^{Bob} e trapaçear??**
- Não, resistência a 2a pré-imagem

Compromissos - Commitments

- Então se Alice e Bob combinam de usar "cara" e "coroa" não fica difícil para Bob "inverter" o valor h...
- basta tentar calcular o hash de "cara" e comparar...
- ... e depois (se for necessario) repetir com "coroa"
- Quando o espaço de entrada das funções de hashing for pequeno, um uso *ingênuo* das funções de hashing permite a sua inversão
- Para resolver esse problem, vamos **aumentar o espaço de entrada**

Compromissos - Commitments

- Um *nonce* é uma sequencia de bits aleatórios usado uma única vez em um protocolo
- O commitment de um valor v usando um *nonce* n pode ser feito por

$$\mathcal{H}(n|v)$$

- No passo de revelação n e v são enviados
- Se n tiver um tamanho grande (160 bits?) impede a força bruta

Compromissos - Commitments / Aplicações

- Leilão de Envelope fechado
- Os candidatos enviam envelopes fechados e lacrados ate uma data limite
- No dia do leilão, o leiloeiro abre os envelopes e determina o melhor lance
- (comum em licitações/privatizações)

Compromissos - Commitments / Aplicações

- Leilão de Envelope fechado
- O candidato i envia $h_i = \mathcal{H}(N_i|M_i)$
- Cada h_i é publicado
- Após a fase de compromisso, os candidatos revelam M_i e N_i
- O leiloeiro e os demais candidatos podem verificar a legitimidade de cada M_i
- (Estritamente falando, o candidato pode desistir de revelar M_i....mas uma **caução** pode obrigar a revelação!)

Compromissos - Commitments / Aplicações

- LUPA
- Lance aberto "atrapalha" o jogo...

Par-Impar

- Par-Impar...cara-coroa
- aposta $\frac{1}{2}$

```
pragma solidity >=0.4.25 <0.6.0;

library SimpleCommit {

  enum CommitStatesType {Waiting,Revealed}

  struct CommitType {
    bytes32 commited;
    byte value;
    bool verified;
    CommitStatesType myState;

  }

  function commit(CommitType storage c,bytes32 h) public {
        c.commited = h;
        c.verified = false;
        c.myState = CommitStatesType.Waiting;
  }


  function reveal(CommitType storage c, bytes32 nonce, byte v) public {
    require (c.myState == CommitStatesType.Waiting);
    bytes32 ver = sha256(abi.encodePacked(nonce,v));
    c.myState = CommitStatesType.Revealed;
    if (ver==c.commited) {
      c.verified = true;
      c.value =v;
    }
  }
```

```
  function isCorrect(CommitType storage c) public returns (bool) {
    require (c.myState == CommitStatesType.Revealed);
    return c.verified;
  }

  function getValue(CommitType storage c) public returns(byte) {
    require (c.myState == CommitStatesType.Revealed);
    require (c.verified==true);
    return c.value;
  }


}
```

```
    pragma solidity >=0.4.25 <0.6.0;

import "./SimpleCommit.sol";

contract CoinFlipping {

   using SimpleCommit for SimpleCommit.CommitType;


   SimpleCommit.CommitType firstPlayer;
   SimpleCommit.CommitType secondPlayer;
   address payable firstPlayerAddress;
   address payable secondPlayerAddress;
   uint value;
   address winner;

   constructor(bytes32 c) public payable {
     firstPlayer.commit(c);
     value = msg.value;
     firstPlayerAddress = msg.sender;
   }

   function joinBet(bytes32 c) public payable {
      secondPlayer.commit(c);
      secondPlayerAddress = msg.sender;
   }
```

```
    function reveal(byte v,bytes32 nonce) public {
      if (msg.sender == firstPlayerAddress) {
        firstPlayer.reveal(nonce,v);
      }
      if (msg.sender == secondPlayerAddress) {
        secondPlayer.reveal(nonce,v);
      }
    }

    function pay() public {
      if (firstPlayer.isCorrect() && secondPlayer.isCorrect()) {
        byte v1 = firstPlayer.getValue();
        byte v2 = firstPlayer.getValue();
        if (v1 == v2) {
          firstPlayerAddress.transfer(2*value);
        } else {
          secondPlayerAddress.transfer(2*value);
        }

      }
    }
}
```

Pelo menos 3 bugs:

1. Não é verificado se o jogador 2 fez o pagamento
2. O jogo só faz o pagamento se ambos se comportarem
3. O codigo assume que o valor escolhido será 0 ou 1