Segurança da Informação
Aula 2

Ivan Sendin

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

23 de maio de 2024

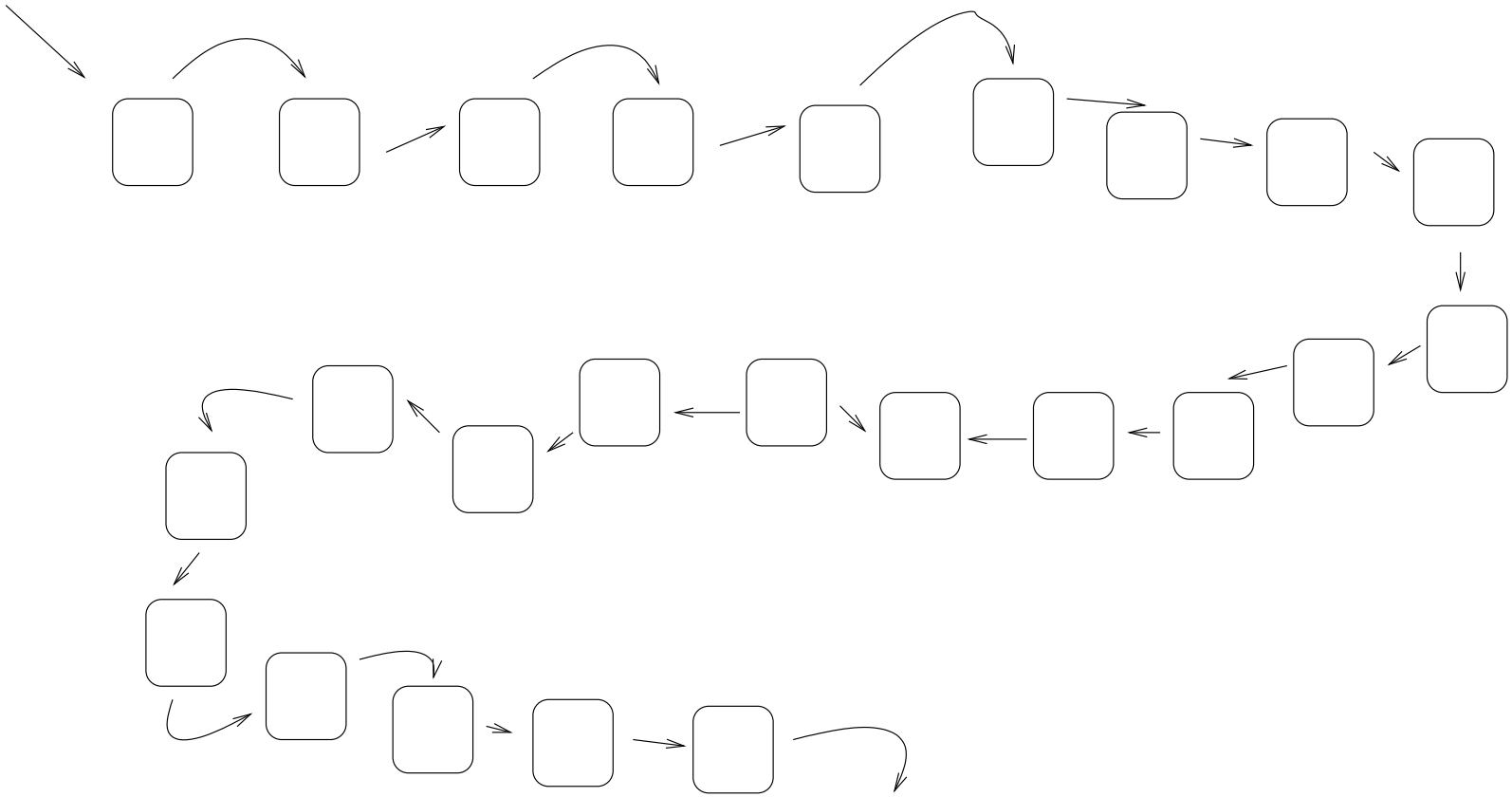
- Exponencial
- Executar 2^n operações é impossível para n de tamanho razoável (n=200, por exemplo)
- Um evento com probabilidade de $\frac{1}{2^n}$
- Para $n = 200$, nunca vai acontecer...nao importa o quanto voce se esforçe
- Para $n = 10$ ou 20 depende do seu esforço

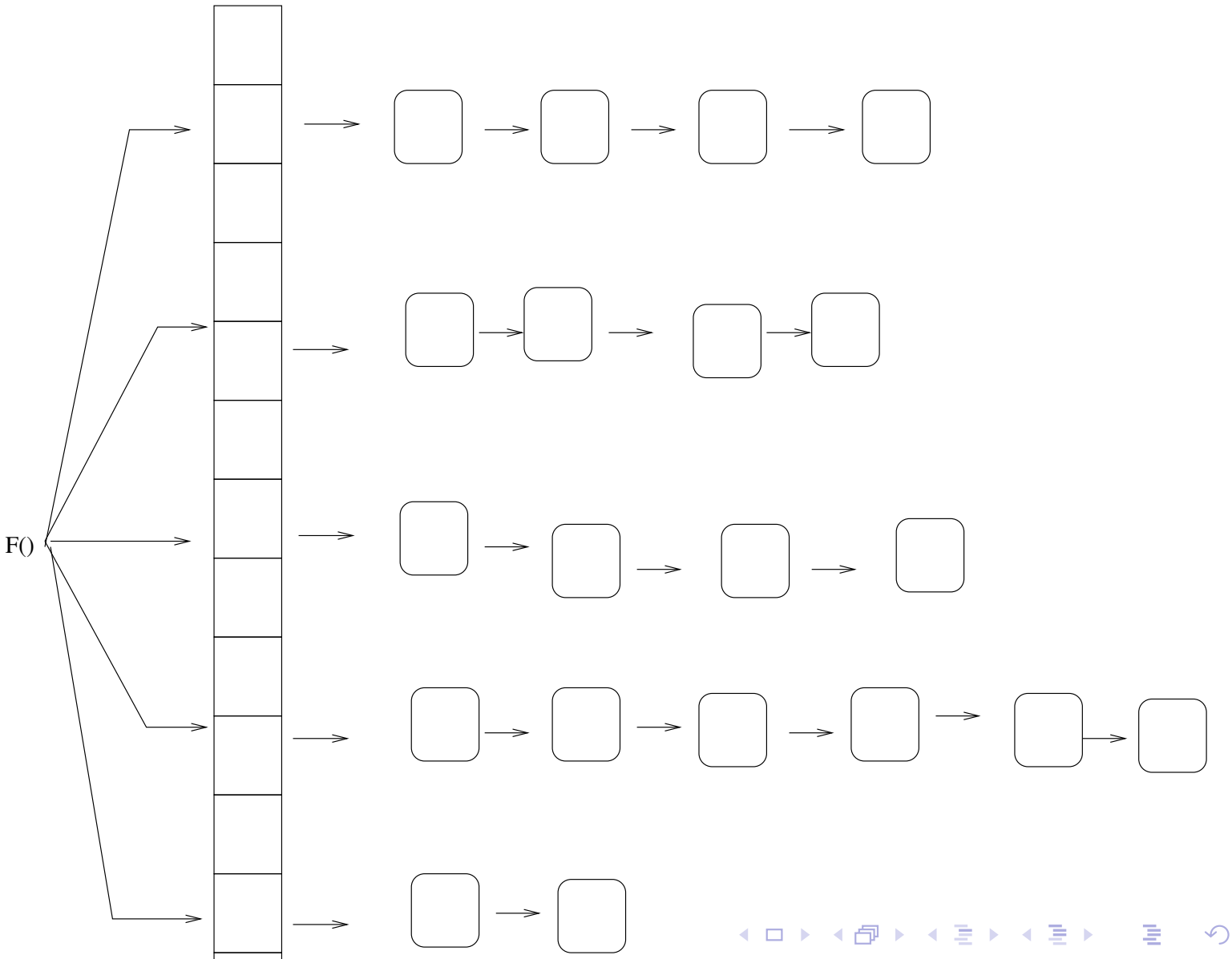
- Um banco “digital” é fácil de ser criado...
- Já temos e faz tempo
- Um servidor e esta tudo resolvido!
- O Bitcoin é uma **moeda**
  Eu tenho,eu gasto
- Para o Bitcoin funcionar é necessário **Consenso Distribuído**

- Consenso Distribuído ou Consenso de Sakamoto
- Pessoas com interesses conflitantes
  Dinheiro!
- Objeto em consenso: "planilha" com o "saldo" de cada participante
- Comunicação: rede P2P "não permissionada"
- Essa é a novidade do Bitcoin....

- IMPORTANTE
- Confiar nos protocolos...nos algoritmos
- Nunca nos outros
- (tenha sempre isso em mente!!)

- Hashing
  Espalhamento, resumo ou dispersão
- Estrutura de Dados (Banco de Dados)
- Transforma uma lista muito grande em muitas listas pequenas/unitárias
- $O(1)$
- O mundo real não funciona assim...muito, muito caro!
  Leetcode X mundo real




F()

- Na figura anterior existem muitos “encadeamentos”
- Espera-se que eles ocorram com probabilidade baixa!

- As funções de hashing cirptográficas são o **canivete suiço** da criptografia
- 3 propriedades criptográficas
- Centenas de usos...
- Eram o *primo pobre* da criptografia até o surgimento das criptomoedas.

Propriedades

$$h = \mathcal{H}(x)$$

- **Compressão** $|h| << |x|$ (tamanho é fixo)
- **Eficiência**
- (**Algoritmo Público**)

Propriedades Criptográficas

$$h = \mathcal{H}(x)$$

- Dificil ! = impossivel (teorico)
- Dificil = computacionalmente inviável
- Difícil = altamente improvável, $p() \approx 0$

Propriedades Criptográficas

$$h = \mathcal{H}(x)$$

- **Unidirecionais**
  - Dado x, calcular $h = \mathcal{H}(x)$ é fácil
  - dado h deve ser dificil determinar x' tal que $h = \mathcal{H}(x')$
- x' pode ser diferente de x
- Sim, existe um algoritmo simples para *inverter* o hash
- **Resistência a pré-imagem**

Propriedades Criptográficas

$$h = \mathcal{H}(x)$$

- **Resistente a segunda pré imagem**:
  - Dado x_1
  - Deve ser dificil determinar x_2, $x_2 \neq x_1$, com $\mathcal{H}(x_1) = \mathcal{H}(x_2)$
- Alguma informação de x_1 pode ser usada...

Propriedades Criptográficas

$$h = \mathcal{H}(x)$$

- **Resistente a colisoes**
  - deve ser dificil encontrar x_1 e x_2 com $\mathcal{H}(x_1) = \mathcal{H}(x_2)$
  - Liberdade em x_1 e x_2

Propriedades Criptográficas

$$h = \mathcal{H}(x)$$

- As propriedades são muito parecidas
- Por enquanto são "sem sentido"
- As coisas ficam mais claras conforme o uso

Algumas Funções

- MD5 (*message digest*)
  - Ron Rivest
  - MD2,MD4,...
  - hash de 128 bits
  - Atualmente não é considerado seguro (colisões)

Algumas Funções

- Família SHA
  - Secure Hash Algorithm
  - 0,1,2,3
  - ate 512 bits
  - SHA-0 é insegura (criptoanálise)
- RIPEMD
  - 160 bits

SHA3 e RIPEMD são as mais usadas no mundo das criptomoedas.

```
luke@xwing:$ echo teste | sha1sum
9dc628289966d144c1a5fa20dd60b1ca1b9de6ed
luke@xwing:$ echo Teste | sha1sum
5a0d15fb8760e783cdf5b36495144dce113a05c8
```

Em binario:

```
...11100101000011011100111011110011011101101
...11100111000010001001110100000010111001000
```

```
luke@xwing:$ echo Teste | sha512sum
0337b702779b146a461929b62ddaae71766b4d0739b6a2
facbb655dd8aa0eb91168a1f33d5ccb7ff7b895c54fc32
40b62391ae666e1590a2cab845109acb8064  -
```

Hashing on line

Resumo

$$h = \mathcal{H}(x)$$

- Dado h ninguem consegue saber nada sobre x...grande? pequeno? é PDF? tema muito bit 1?
- h é um identificador único de x...
- Se eu conheço h eu nao "aceito" $x'(\neq x)$
- (Usamos esses fatos para criar o Bitcoin!)

Onde tudo começou

- How to Time-Stamp a Digital Document
- Stuart Haber e W.S.Stornetta - 1991
- Impedir a modificação de documentos digitais
- "Timestamp"
  Impede o gasto duplo de dinheiro (digital)
- Varias soluções...

- *Time Stamp Service* (cartório?)
- (*Trusted Third Party* == NUNCA FAÇA ISSO!!)
- Usuário envia x para o TSS
  "Eu vendi a minha casa para o João"
- TSS diz "O documento x existe desde d"
- Impede o gasto duplo
  (ok...nao esta completo)
- Privacidade, Banda/Armazenamento,
  Incompetência e **confiança**

- Usuário envia $h = \mathcal{H}(x)$ para o TSS
- TSS **assina** "O hash h existe desde d"
- Propriedades do hash
- Resolve banda, armazenamento e privacidade
- **Confiança**
  O TSS ainda pode inventar coisas....

- Usuário envia $h = \mathcal{H}(x)$ para o TSS
- TSS **cria**:

$$C_n = (n, t_n, h_n, \mathcal{H}(C_{n-1}))$$

- Confiança **distribuída**
- Se alguem me envia um C_n eu posso confiar ??

- C_n é verdadeiro ??

$$C_n = (n, t_n, h_n, \mathcal{H}(C_{n-1}))$$

- C_{n+1} atesta/testemunha que C_n é verdadeiro

$$C_{n+1} = (n+1, t_{n+1}, h_{n+1}, \mathcal{H}(C_n))$$

- C_{n+2} atesta/testemunha que C_{n+1} e C_n são verdadeiros
- C_{n+3} atesta/testemunha que...
- Quanto mais "antigo" for C_n, mais difícil falsificar....

- Uma Blockchain é uma cadeia de blocos
- Cada bloco contém informações do **negócio**
  No nosso exemplo *h*
- Cada bloco "aponta" para o anterior
- Este aponta **não** é um aponta de *localização*!
- É um aponta de **confiança**

- Se alguem for falsificar o B_n...
- preciserá falsificar os blocos $B_{n+1}, B_{n+2} \ldots B_u$
- No modelo do *timestamp* cada bloco esta sob os cuidados de um usuário...essa falsificação exigiria um complô
- A Blockchain/Bitcoin estão distribuídos em uma rede P2P....
- Falsificar alguns milhares de blocos é trivial...pelo menos por enquanto

Prova de trabalho

- O protocolo *Hashcash* propõe o uso de **colisões parciais** para evitar *spam*
- Serviços gratuitos podem *cobrar* colisões/inversões parciais, evitando abusos
- parciais = numero de bits reduzido
- Obtidos por força bruta: gastando tempo e CPU

Prova de trabalho

- O Procotolo de email poderia exigir que os emails enviados tivessem a seguinte característica:

$$\mathcal{H}(PoW|subject|Recipient|Msg|timestamp...) < K$$

$$\mathcal{H}(PoW|subject|Recipient|Msg|timestamp...) < K$$

- PoW (=Prova de trabalho)
- PoW deve ser gerado por busca exaustiva
- Para $K = 1$, temos prob. $\frac{1}{2^n}$ de achar
- Para $n = 10$ e $K = 102$, temos prob $\approx 1/10$ em cada tentativa...
- O servidor verifica em apenas um passo!
- Email legitimo: ok, voce pode esperar um minuto para enviar!
- Spam com milhares de emails: NÃO!!!

```python
import hashlib

pre = "Informacoes do negocio, hash do bloco anterior,etc..."
x = 1

mined = False

while not mined:
    temp = pre +str(x)
    h = hashlib.sha256(temp.encode()).hexdigest()
    if h[:6] == '000000':
        mined = True
    x+=1
print(h)
```

```
yoda@dagobah:~$ python miner.py
0000000399c6aea5ad0c709a9bc331a3ed6494702bd1d129d8c817a0257a1462
```

- O Bloco é composto por informações como: transações, timestamp, altura, **hash do bloco anterior**...
- e uma área de *nonce*
- O hash do bloco deve ter algumas características específicas para ser aceito pelo protocolo
- Prova de trabalho (PoW)
- Falsificar blocos ficou mais difícil...quanto mais velho mais difícil

- Falsificar blocos ficou mais difícil...

$$C_n = (n, t_n, h_n, \mathcal{H}(C_{n-1}), NONCE_n)$$

- Achar o $NONCE_n$ de todos os que estao acima dele
- Quanto mais velho o bloco, mais difícil

- E a famosa remuneração dos mineradores???
- esta no campo transações!
- A primeira transação (coinbase) é uma transação de criação de moedas, então cada minerador coloca os seus endereços com destinatorios desta transação
- Além disso cada transação “normal” também paga uma taxa(tip)..o minerador coleta estes valores e também redireciona para um endereço dele

- No canal BlockchainTXT temos um micro exemplo de blockchain
- Voces devem fazer a prova de trabalho e ir aumentando a blockchain conforme descrito na plataforma
- Cuidado com caracteres “invisíveis” e use a função de hashing correta
- Existem detalhes que voce perceberão somente na execução do exercicio
- Prova de trabalho: o Hash deve iniciar com ’0000000’ (7 zeros = 28 bits)