



Blockchain, Criptomoedas & Tecnologias Descentralizadas

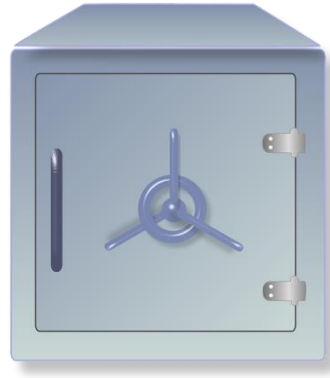
Criptografia simétrica

Geração de números aleatórios

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br
Escola Politécnica, Universidade de São Paulo

Objetivos

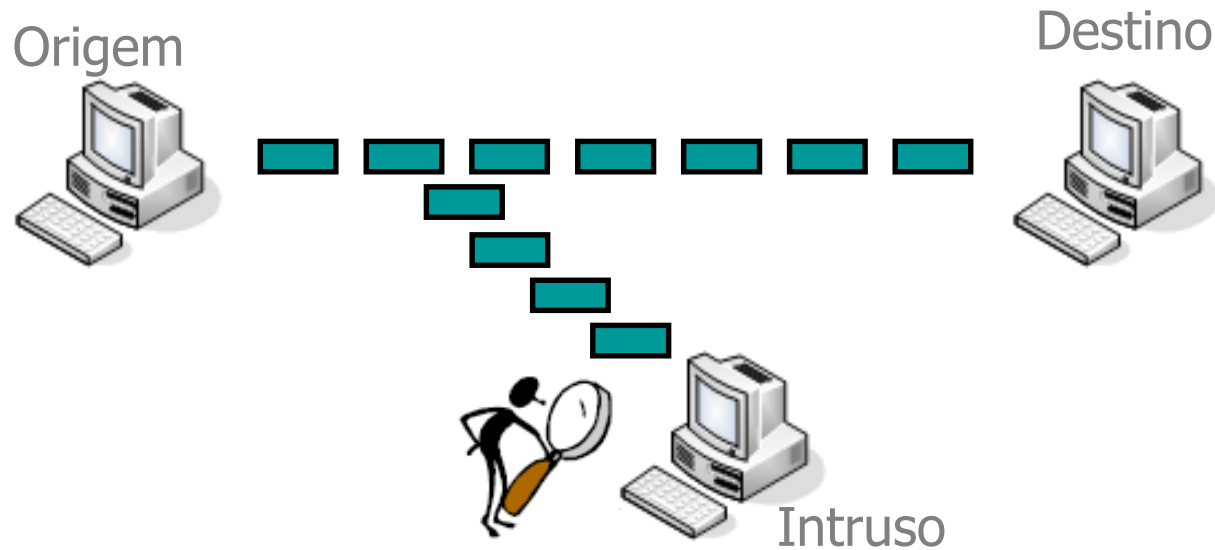
- Discutir os principais serviços fornecidos por algoritmos criptográficos simétricos
 - Disponibilidade
 - Confidencialidade: **cifras**
 - Integridade: **funções de hash**
 - Autenticidade: **códigos de autenticação de mensagens**
 - Irretratabilidade
- Discutir mecanismo auxiliar: geração de **números aleatórios**



Confidencialidade: Cifras

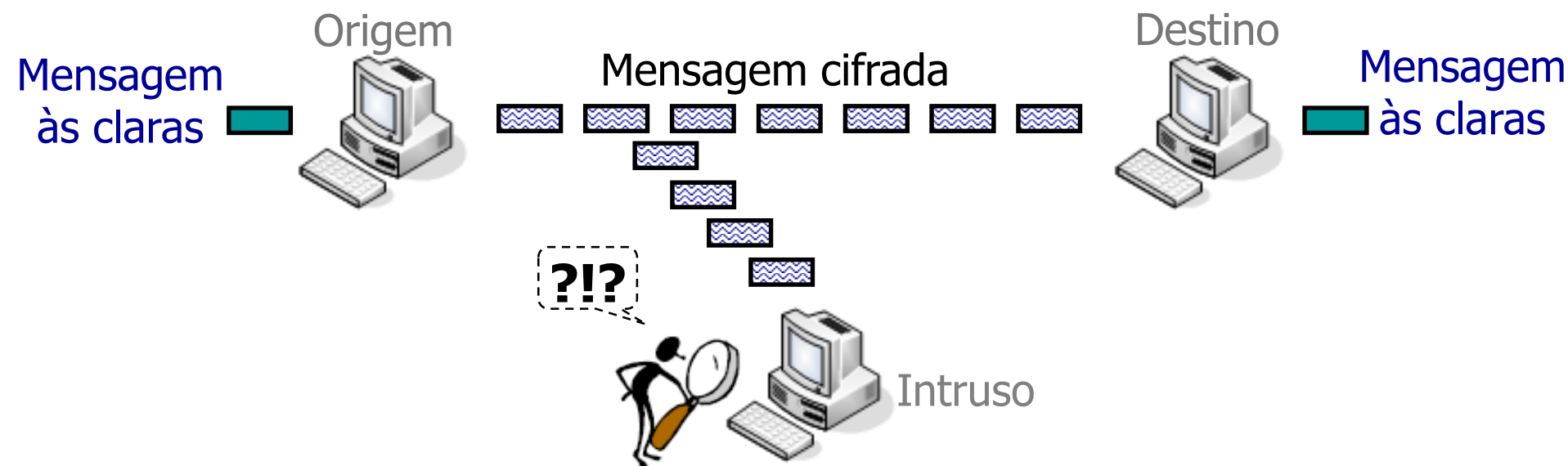
Confidencialidade

- Serviço necessário:
 - Prevenção do vazamento de informações



Confidencialidade: Cifras

“Embaralhamento” de dados: **cifras**

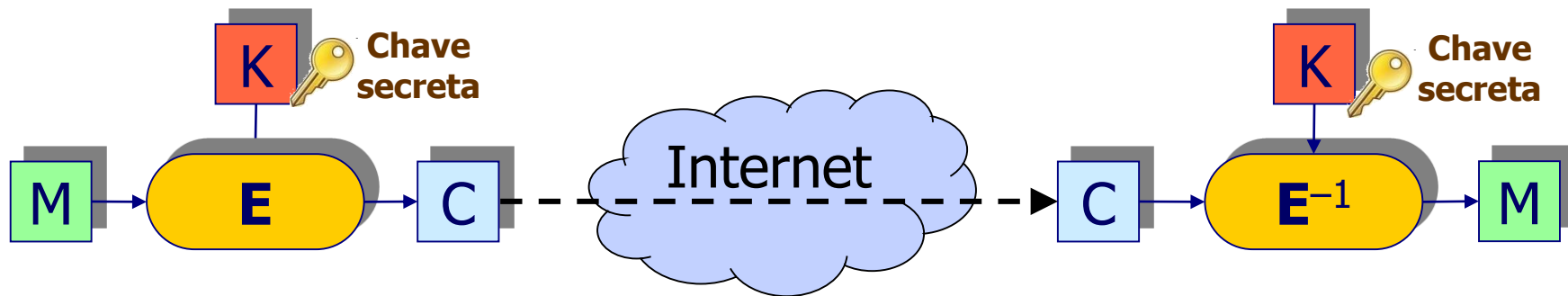


“Seu saldo é R\$10.000,00” ——— “Hlaafd7Y(@&fhF23%7” ———▶

“Seu saldo é R\$10.000,00”

Cifra Simétrica – Definição

- Transformação matemática inversível cujo cálculo depende, no sentido direto (**cifração**) e no sentido inverso (**decifração**), de uma *mesma* informação secreta: a chave K .
 - Se K for descoberta, a confidencialidade é perdida



Cifras: Algoritmos Principais



- DES (Data Encryption Standard):
 - Blocos de 64 bits; Chaves de 56 bits
 - **Obsoleto**: aposentado em 2004 (chaves muito curtas!)



- RC4 (ArcFour):
 - Chave: tamanho variável (múltiplo de 8 bits, até 2048 bits).
 - **Legado**: antigo padrão do SSL/TLS, **aposentado** em 2015 (RFC 7465)



- 3DES (DES triplo):
 - **Legado**: tripla aplicação do DES, aproveitando implementações existentes → **desaconselhado** em 2017; **aposentado** em 2023
 - Chaves: $3 \times 56 = 168$ bits (mas segurança é de ~112 bits)

- AES (Advanced Encryption Standard):
 - **Padrão atual** (desde 2001): vencedor de concurso público iniciado em 1997 (nome original: Rijndael)
 - Blocos de 128 bits; Chaves de 128/192/256 bits.



Exemplo prático: HTTP vs. HTTPS

- HTTP: dados passam em aberto na rede
 - Site de testes: <http://testphp.vulnweb.com/login.php>

The image shows a Wireshark packet capture of an HTTP login attempt. The packet list shows a POST request to /userinfo.php. The packet details pane shows the form data: username=test and password=test. The packet bytes pane shows the raw data, which is URL-encoded. Two blue arrows point to the form items in the details pane.

No.	Time	Source	Destination	Protocol	Length	Info
13	0.741547	176.28.50.165	172.20.5.237	HTTP	1156	HTTP/1.1 200 OK (text/css)
14	0.759762	176.28.50.165	172.20.5.237	TCP	1514	80 → 38873 [ACK] Seq=3160 Ack=394 Win=54 Len=1460 [TCP segment of a reassembled PDU]
15	0.759907	176.28.50.165	172.20.5.237	TCP	1514	80 → 38873 [ACK] Seq=4620 Ack=394 Win=54 Len=1460 [TCP segment of a reassembled PDU]
16	0.759993	176.28.50.165	172.20.5.237	HTTP	874	HTTP/1.1 200 OK (GIF89a)
17	5.939076	172.20.5.237	176.28.50.165	HTTP	711	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
18	6.166120	176.28.50.165	172.20.5.237	TCP	1514	80 → 38873 [ACK] Seq=6900 Ack=1051 Win=65 Len=1460 [TCP segment of a reassembled PDU]
19	6.166293	176.28.50.165	172.20.5.237	HTTP	1459	HTTP/1.1 200 OK (text/html)

Frame 17: 711 bytes on wire (5688 bits), 711 bytes captured (5688 bits)

Ethernet II, Src: Pegatron_1a:8f:4b (38:60:77:1a:8f:4b), Dst: JuniperN_86:14:db (00:14:f6:86:14:db)

Internet Protocol Version 4, Src: 172.20.5.237, Dst: 176.28.50.165

Transmission Control Protocol, Src Port: 38873, Dst Port: 80, Seq: 394, Ack: 6900, Len: 657

Hypertext Transfer Protocol

HTML Form URL Encoded: application/x-www-form-urlencoded

- > Form item: "uname" = "test"
- > Form item: "pass" = "test"

0250 2f 74 65 73 74 70 68 70 2e 76 75 6c 6e 77 65 62 /testphp.vulnweb

0260 2e 63 6f 6d 2f 6c 6f 67 69 6e 2e 70 68 70 0d 0a .com/login.php..

0270 41 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a Accept-Encoding:

0280 20 67 7a 69 70 2c 20 64 65 66 6c 61 74 65 0d 0a gzip, deflate..

0290 41 63 63 65 70 74 2d 4c 61 6e 67 75 61 67 65 3a Accept-Language:

02a0 20 65 6e 2d 55 53 2c 65 6e 3b 71 3d 30 2e 39 0d en-US,en;q=0.9..

02b0 0a 0d 0a 75 6e 61 6d 65 3d 74 65 73 74 26 70 61 ..uname=test&pa

02c0 73 73 3d 74 65 73 74 ss=test

Text item (text), 9 bytes

Packets: 19 · Displayed: 19 (100.0%)

Profile: Default

Usuário e senha
enviados às claras

Exemplo prático: HTTP vs. HTTPS

- HTTPS: dados cifrados (túnel SSL/TLS)
 - Login em <https://uspdigital.usp.br/>

The image shows a Wireshark packet capture of an HTTPS login session. The top pane displays a list of packets, with packet 15 (Application Data) selected. The middle pane shows the details of this packet, including the TLSv1.2 Record Layer and the encrypted application data. A blue arrow points to the encrypted data field. The bottom pane shows the raw packet data in hexadecimal and ASCII. A dashed box highlights the encrypted data in the ASCII view, with the text 'Usuário e senha cifrados' (User and password encrypted) next to it.

No.	Time	Source	Destination	Protocol	Length	Info
9	0.002546	200.144.248.43	172.20.5.237	TCP	1384	443 → 38935 [PSH, ACK] Seq=2661 Ack=518 Win=34654 Len=1330 [TCP segment of a reassembled PDU]
10	0.002567	200.144.248.43	172.20.5.237	TLSv1.2	516	Server Hello, Certificate
11	0.009214	200.144.248.43	172.20.5.237	TLSv1.2	396	Server Key Exchange, Server Hello Done
12	0.009628	200.144.248.43	172.20.5.237	TLSv1.2	396	Server Key Exchange, Server Hello Done
13	0.027200	172.20.5.237	200.144.248.43	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
14	0.028969	172.20.5.237	200.144.248.43	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
15	0.029272	172.20.5.237	200.144.248.43	TLSv1.2	147	Application Data
16	0.029802	172.20.5.237	200.144.248.43	TLSv1.2	415	Application Data

Internet Protocol Version 4, Src: 172.20.5.237, Dst: 200.144.248.43

Transmission Control Protocol, Src Port: 38934, Dst Port: 443, Seq: 644, Ack: 4795, Len: 93

Transport Layer Security

▼ TLSv1.2 Record Layer: Application Data Protocol: http2

Content Type: Application Data (23)

Version: TLS 1.2 (0x0303)

Length: 88

Encrypted Application Data: 1f9660187307ab58c89397b9ca76c1533c986ed1884ebc56a59b4b3b366b8f9dbe2ccd4c... ←

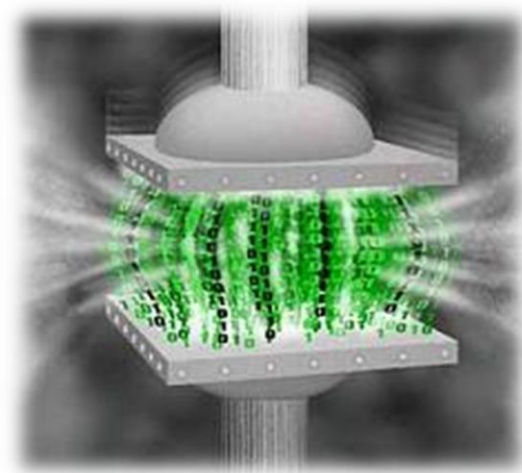
[Application Data Protocol: http2]

0000 00 14 f6 86 14 db 38 60 77 1a 8f 4b 08 00 45 008`w..K..E.
0010 00 85 30 02 40 00 80 06 57 b3 ac 14 05 ed c8 90 ..0.@...W.....
0020 f8 2b 98 16 01 bb ac 9d c8 05 a3 71 7f bd 50 18 +.....q..P..
0030 fd 3c 4b eb 00 00 17 03 03 00 58 1f 96 60 18 73 ..<K.....X...s
0040 07 ab 58 c8 93 97 b9 ca 76 c1 53 3c 98 6e d1 88 ..X.....v.S<.n..
0050 4e bc 56 a5 9b 4b 3b 36 6b 8f 9d be 2c cd 4c 31 N.V..K;6 k...,.L1
0060 d5 f6 c3 da bd 31 2e 4b c4 3c 1e be f7 72 11 f81.K.<....r..
0070 20 df c0 6a fd 85 cc 6f f1 95 42 e0 a7 40 13 26 ..j...o..B...@.&
0080 5c 91 66 59 6a 45 d0 8f 8e 36 bd 92 97 e2 da e8 \.fYjE...6.....
0090 b4 ce 44 ...D

Payload is encrypted application data (tls.app_data), 88 bytes

Packets: 550 · Displayed: 550 (100.0%)

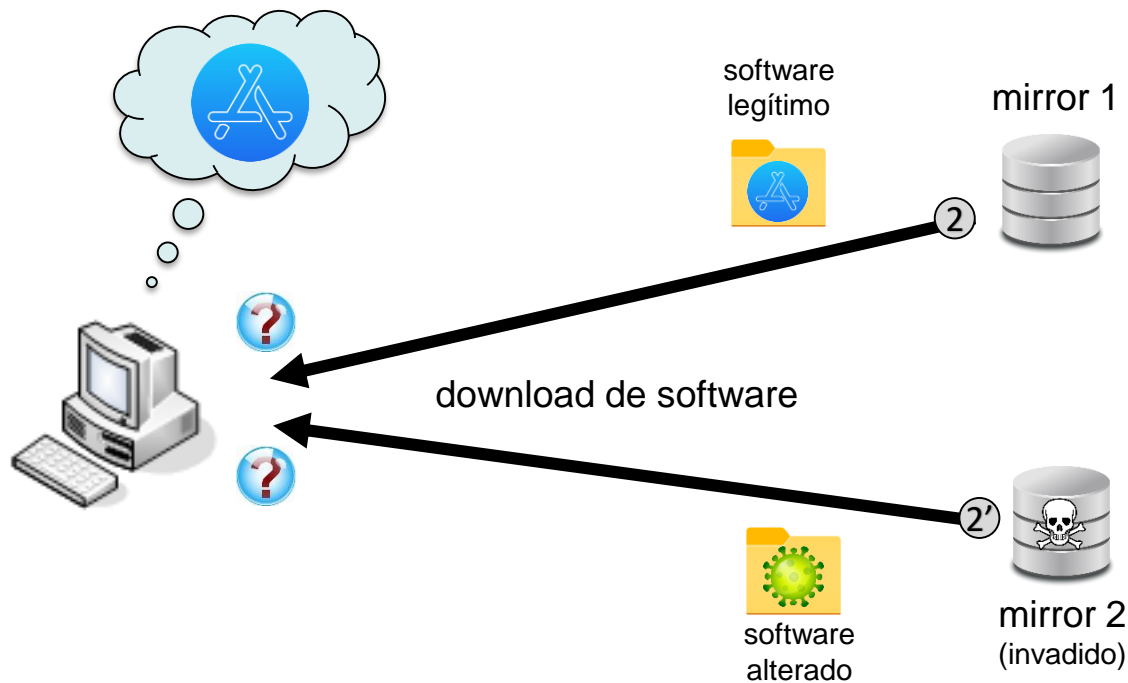
Profile: Default



Integridade: Funções de Hash

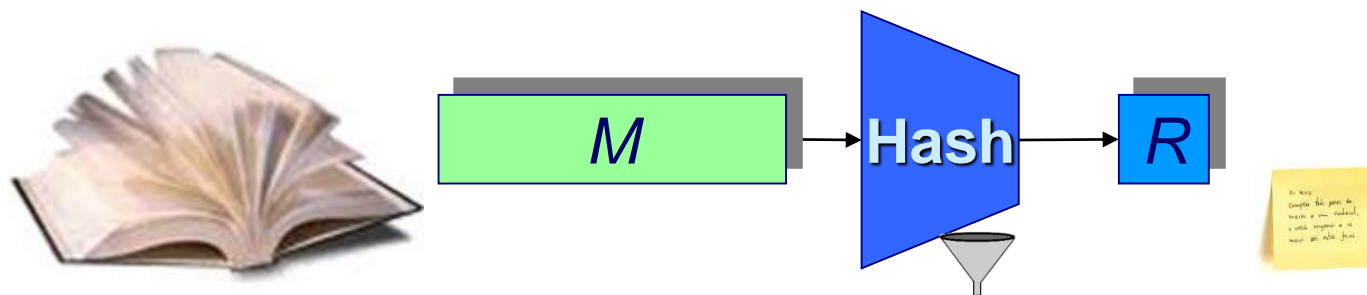
Integridade

- Serviço necessário
 - Capacidade de verificar se informação foi alterada



Integridade: Funções de Hash

- Geram um “**resumo criptográfico**” da entrada
 - **Alterações** nos dados de entrada são **detectadas** porque elas **alteram o resumo**
 - O resumo também é chamado de “**hash**”
 - O hash tem **tamanho fixo**, e seu valor depende exclusivamente da mensagem (**não envolve** o uso de uma **chave secreta**)

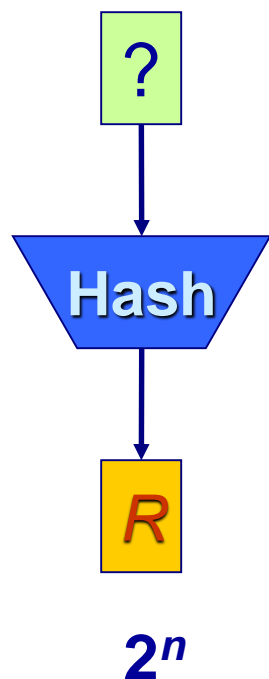


Integridade: Funções de Hash

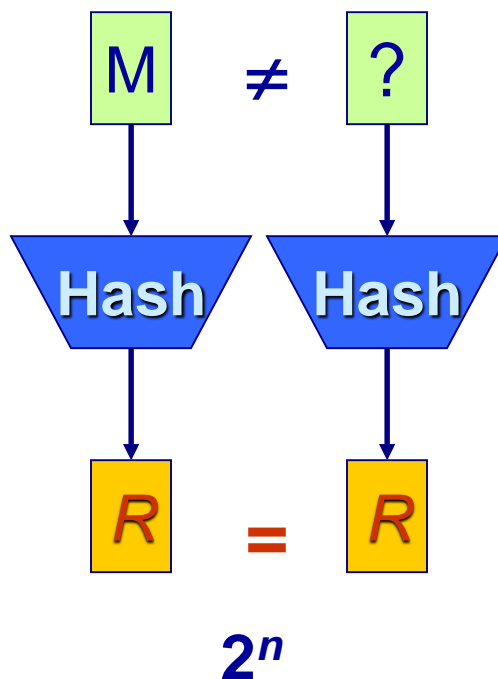
- **Requisitos criptográficos fundamentais**
- (Resistência a primeira inversão) Dado um resumo R , é inviável encontrar uma mensagem M tal que $R = H(M)$.
- (Resistência a segunda inversão) Dado um resumo R e uma mensagem M_1 tal que $R = H(M_1)$, é inviável encontrar outra mensagem $M_2 \neq M_1$ tal que $R = H(M_2)$.
- (Resistência a colisões) É inviável encontrar duas mensagens M_1 e M_2 tais que $H(M_1) = H(M_2)$.

Integridade: Funções de Hash

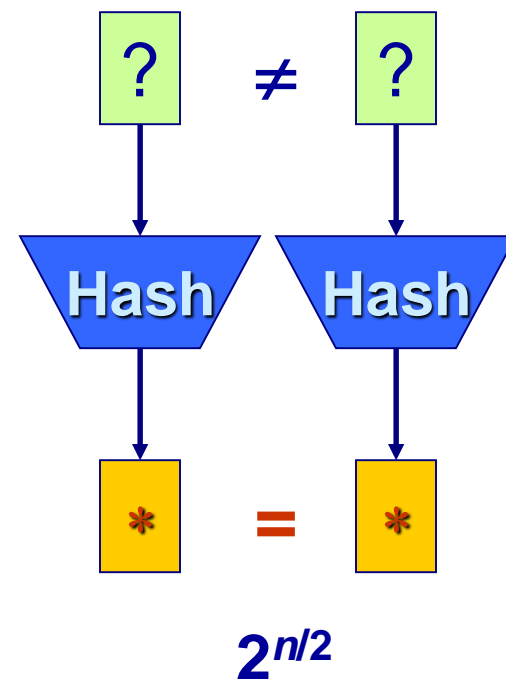
1ª inversão



2ª inversão



colisão

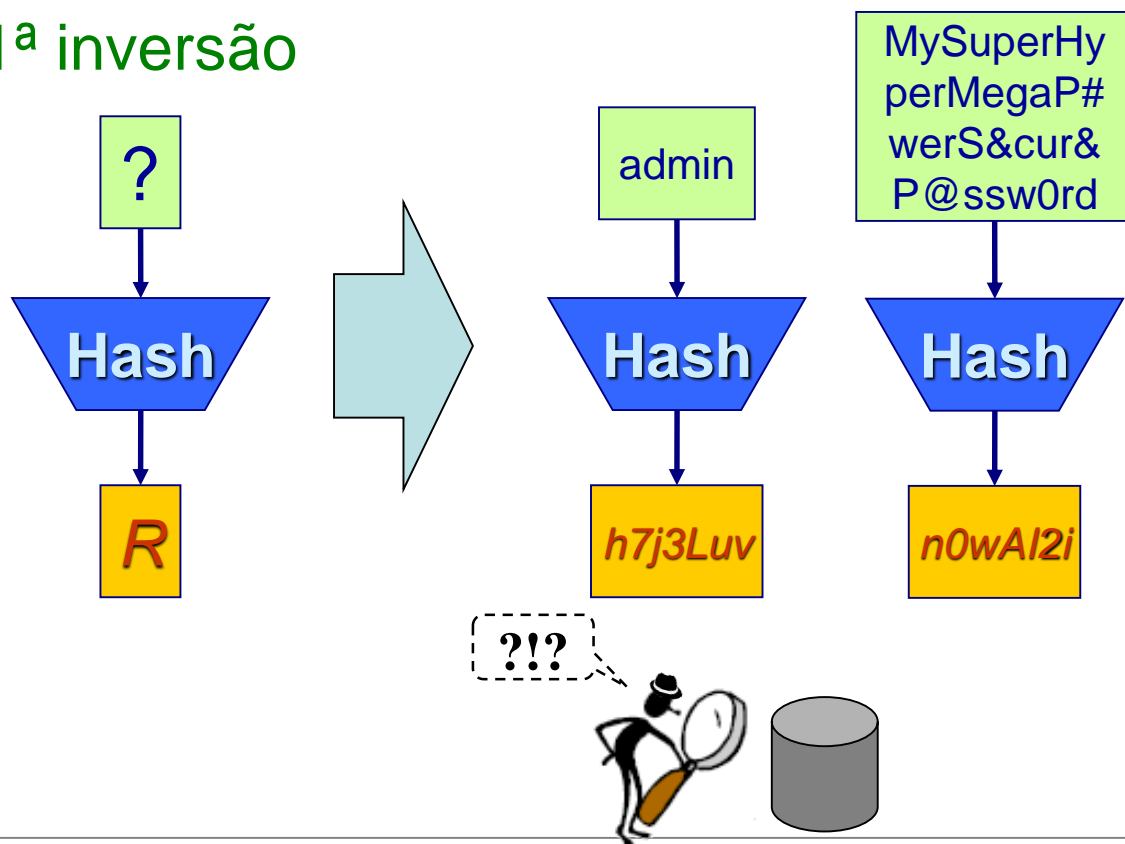


Custo do ataque: hash de n bits

Funções de Hash: usos

Proteção de senhas em bancos de dados

1ª inversão

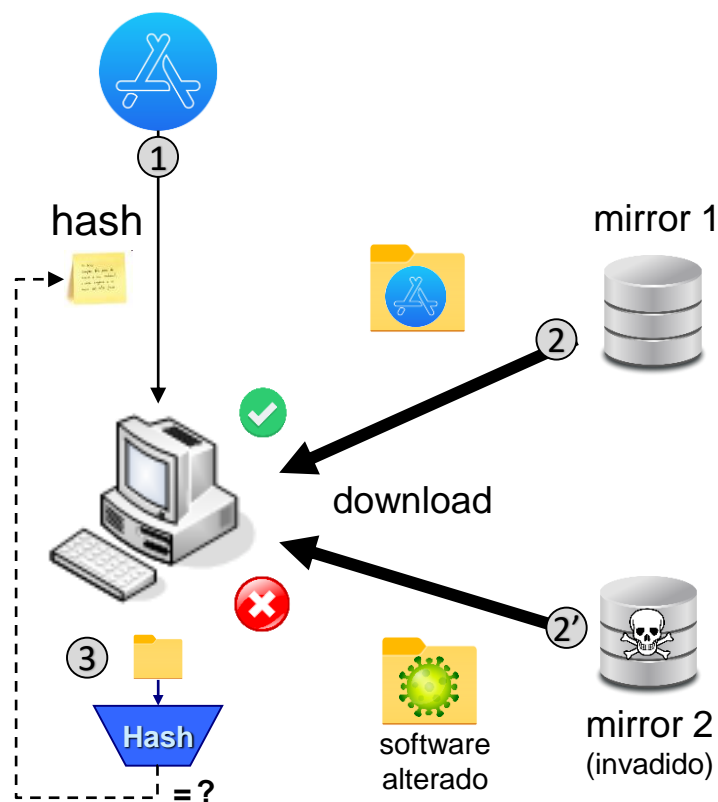
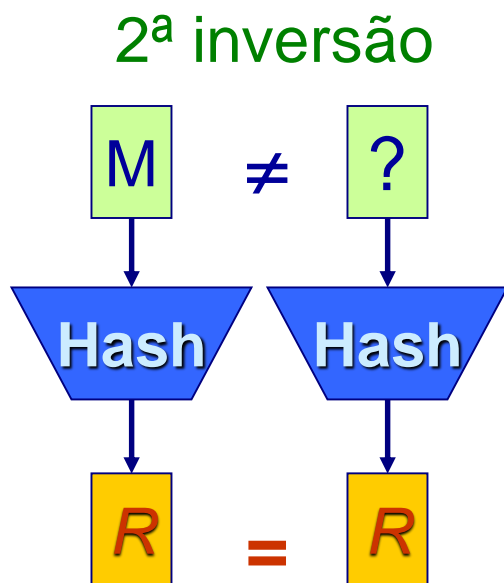


* Na prática, são usados algoritmos derivados de funções de hash: password hashing schemes (PHS)

† Não confundir com cifração, que é usada quando se deseja que alguém autorizado (i.e., de posse de chave secreta) consiga obter entrada a partir da saída

Funções de Hash: usos

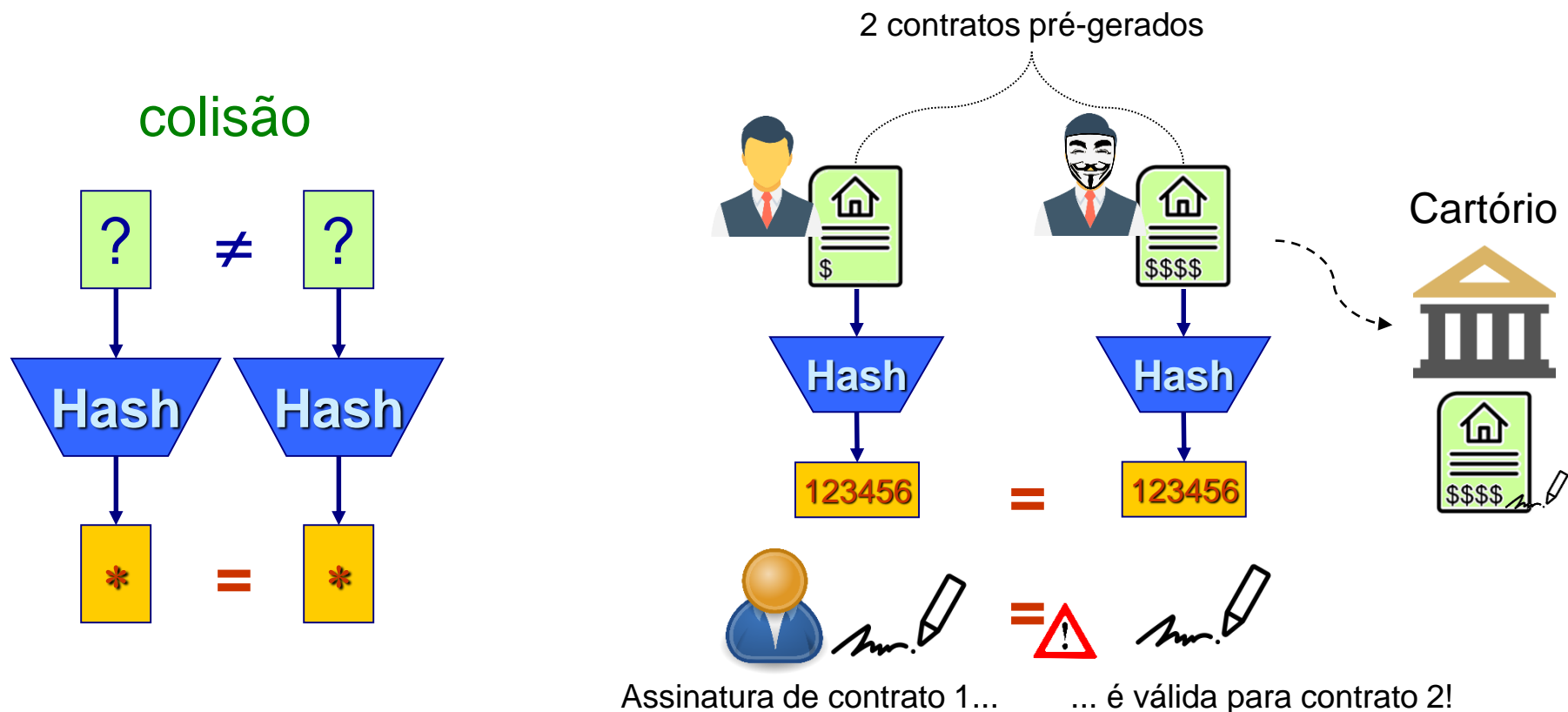
Verificação de downloads



Funções de Hash: usos

Integridade de assinaturas digitais

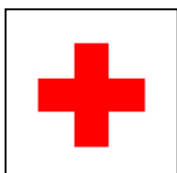
(são feitas sobre hashes dos dados)



Integridade: Funções de Hash



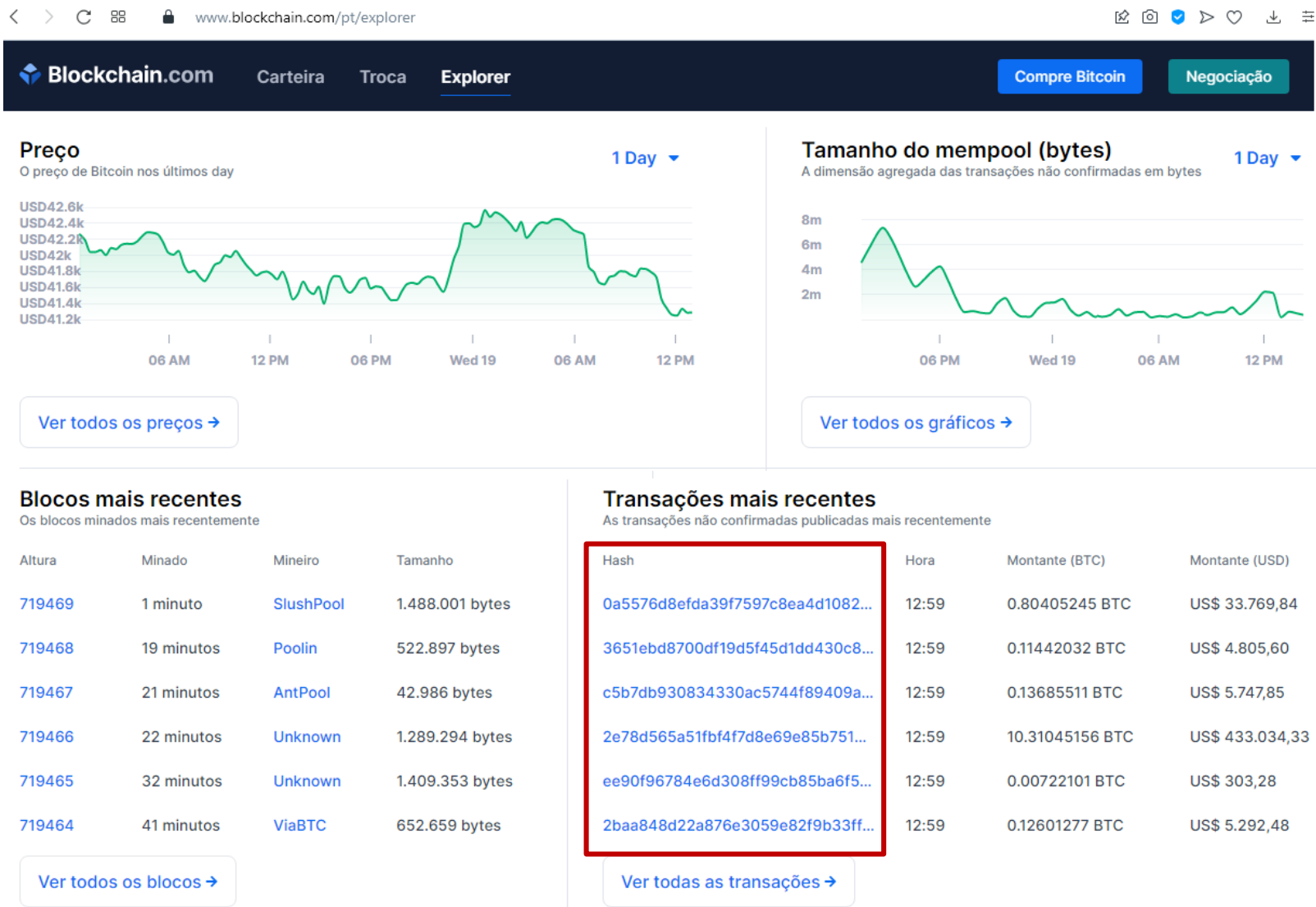
- Família MD:
 - MD2, MD4 e MD5: hashes de 128 bits
 - Completamente quebrada (Wang et al., 2004)



- Família SHA
 - SHA-0: hashes de 160 bits
 - Não recomendado: colisão em 2^{39} passos x 2^{80} projetado
 - SHA-1: hashes de 160 bits
 - Não recomendado: desde 2010, para assinaturas
 - Segurança: colisões em 2^{60} passos x 2^{80} projetado
 - **SHA-2**: Hash de X bits, para X=224, 256, 384 ou 512
 - Paliativo atual: baseados no SHA-1, mas hash grande dificulta ataques
 - **SHA-3**: hashes de 224, 256, 384 e 512 bits
 - Concurso público finalizado em 2012: **Keccak**



Exemplo prático: Bitcoin

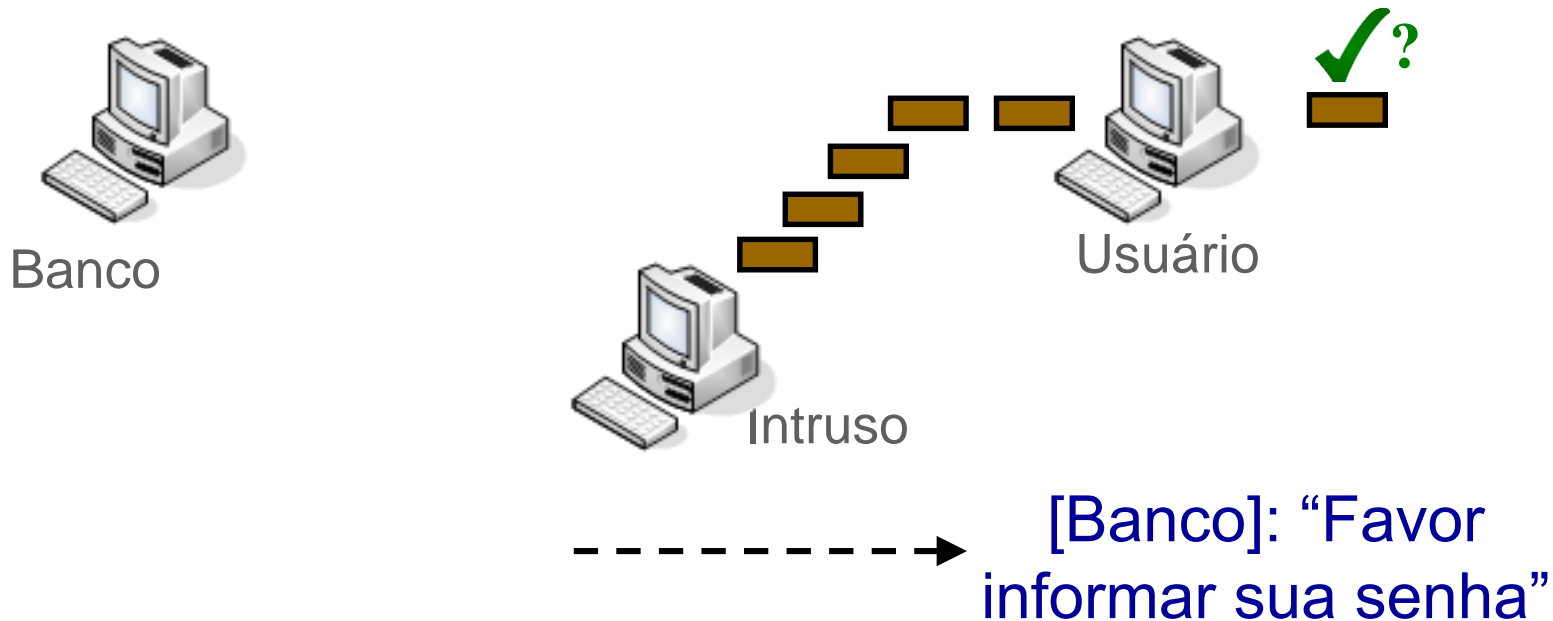




Autenticidade: Códigos de Autenticação de Mensagens (MAC)

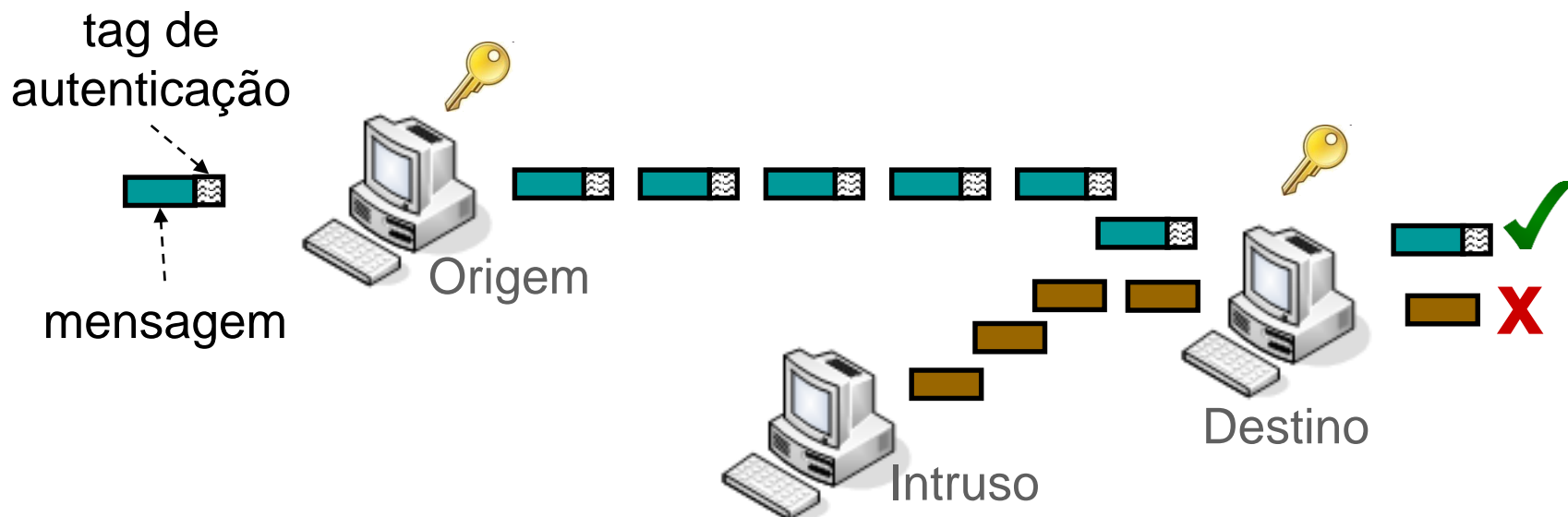
Autenticidade

- Serviço necessário:
 - Capacidade do receptor em verificar quem é o emissor da mensagem



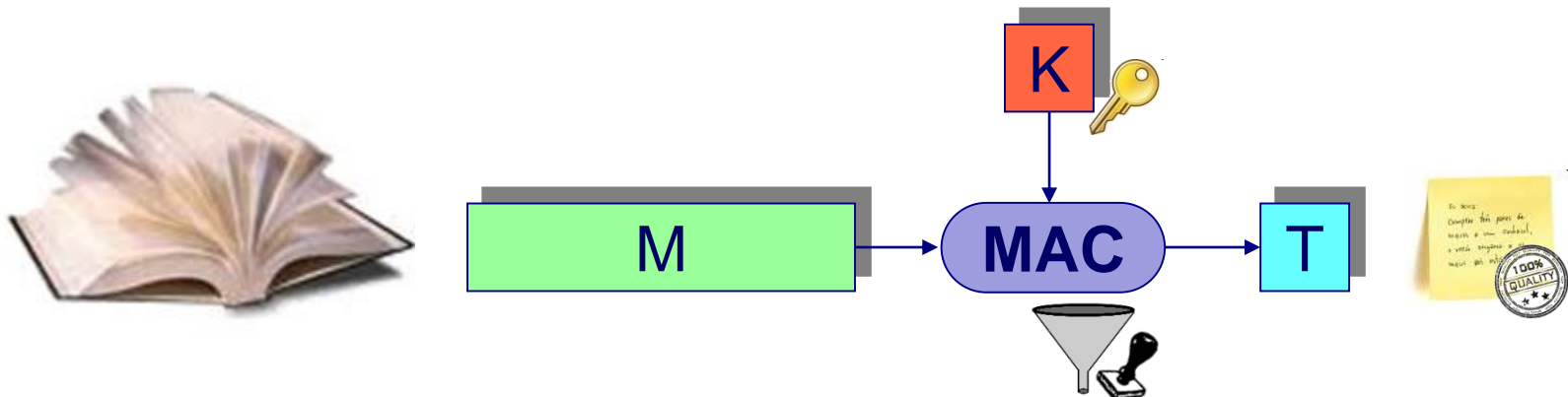
Autenticidade: estratégia básica

- Usar resumo criptográfico dependente de chave
 - Apenas origem e destino conhecem a chave e conseguem calcular resumo corretamente
 - Também garante integridade (alteração na mensagem detectada, como no caso das funções de hash)



Códigos de Autenticação

- Message Authentication Code (**MAC**):
 - Cria resumo que é anexado à mensagem, permitindo detectar alterações (**integridade**) e garantir a **autenticidade** do remetente.
- Resumo: “tag (etiqueta) de autenticação”
 - Depende da **mensagem** e também de uma **chave secreta** conhecida por remetente e destinatário.

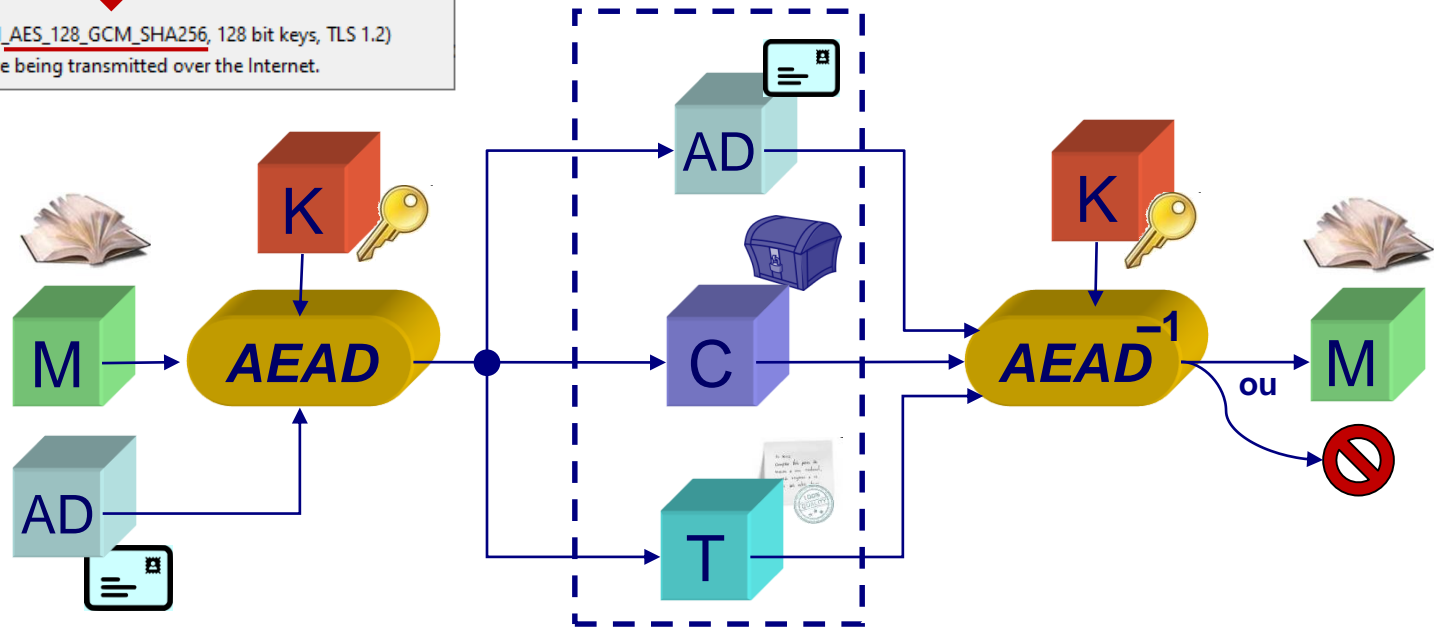
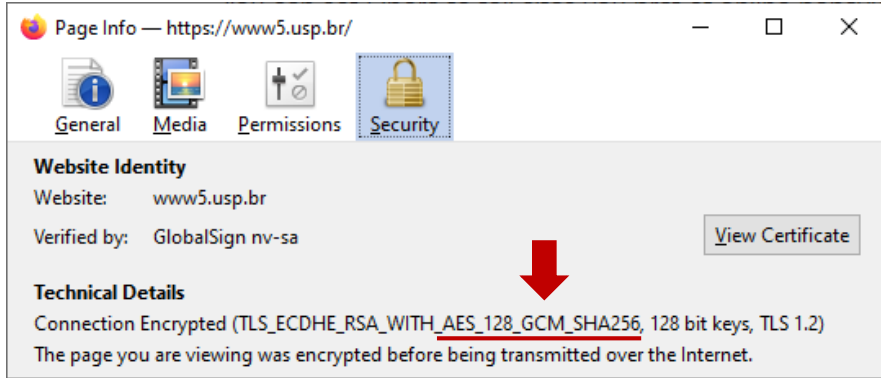




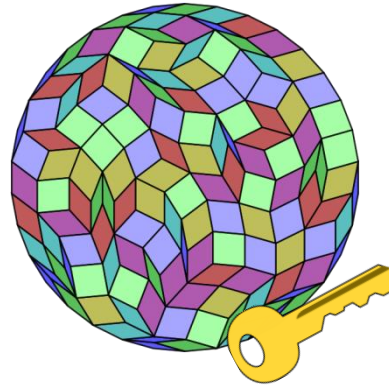
Autenticidade: Algoritmos

- Baseados em cifras de bloco:
 - **CMAC** (NIST SP 800-38B).
 - Pró: tamanho de código (reusam cifras de bloco).
- Baseados em funções de *hash*:
 - **HMAC** (FIPS 198).
 - Pró: desempenho (funções de *hash* puras).
- Combinados com cifras
 - **AEAD**: *Authenticated Encryption with Associated Data* (confidencialidade de parte dos dados)
 - Exemplos tradicionais: **GCM, CCM, EAX**
 - **Concurso** finalizado em 2018 (Caesar):
(<http://competitions.cr.yp.to/caesar-submissions.html>)
 - Ascon (mais leve), AEGIS-128 & OCB (alto desempenho), Deoxys-II (defesa em profundidade: e.g., não requer nonces)

Exemplo prático: TLS



- Dado **confidencial** (C) e **autenticado** (T)
 - AD : dados associados (enviados às claras, autenticados)
 - Serviços: confidencialidade, integridade e autenticidade (cifra simétrica e algoritmo de MAC internos a AEAD)



Geração de chaves: números aleatórios

Estudo de caso: Netscape



- Netscape 1.x (1995).
- Dois estudantes de Berkeley descrevem como quebrar a segurança do navegador, recuperando chaves usadas em sessões seguras (HTTPS) em **25 s**.
- Chaves pequenas?
 - Não, chaves de 128 bits (tamanho atual !!!)
- Pergunta: como isso é possível?



Análise de (in)segurança

- Baixa **aleatoriedade** das chaves de sessão!
 - Chaves geradas a partir do relógio do sistema (precisão de μs), sem acúmulo entre ativações
 - Conhecendo minuto da criação da sessão HTTPS: menos de 60 milhões de chaves possíveis
 - Segurança de cerca de 2^{26} , não 2^{128}



- Geração de chaves segura: fontes de **entropia**
 - Ex. (**físicas**): relógio, ruído térmico
 - Ex. (**comportamentais**): estatísticas de rede, pastas temporárias (Firefox 3.5), posição do mouse (VeraCrypt)
 - Soluções de **sistema**: “SecureRandom” (Java), “/dev/random” (Unix)

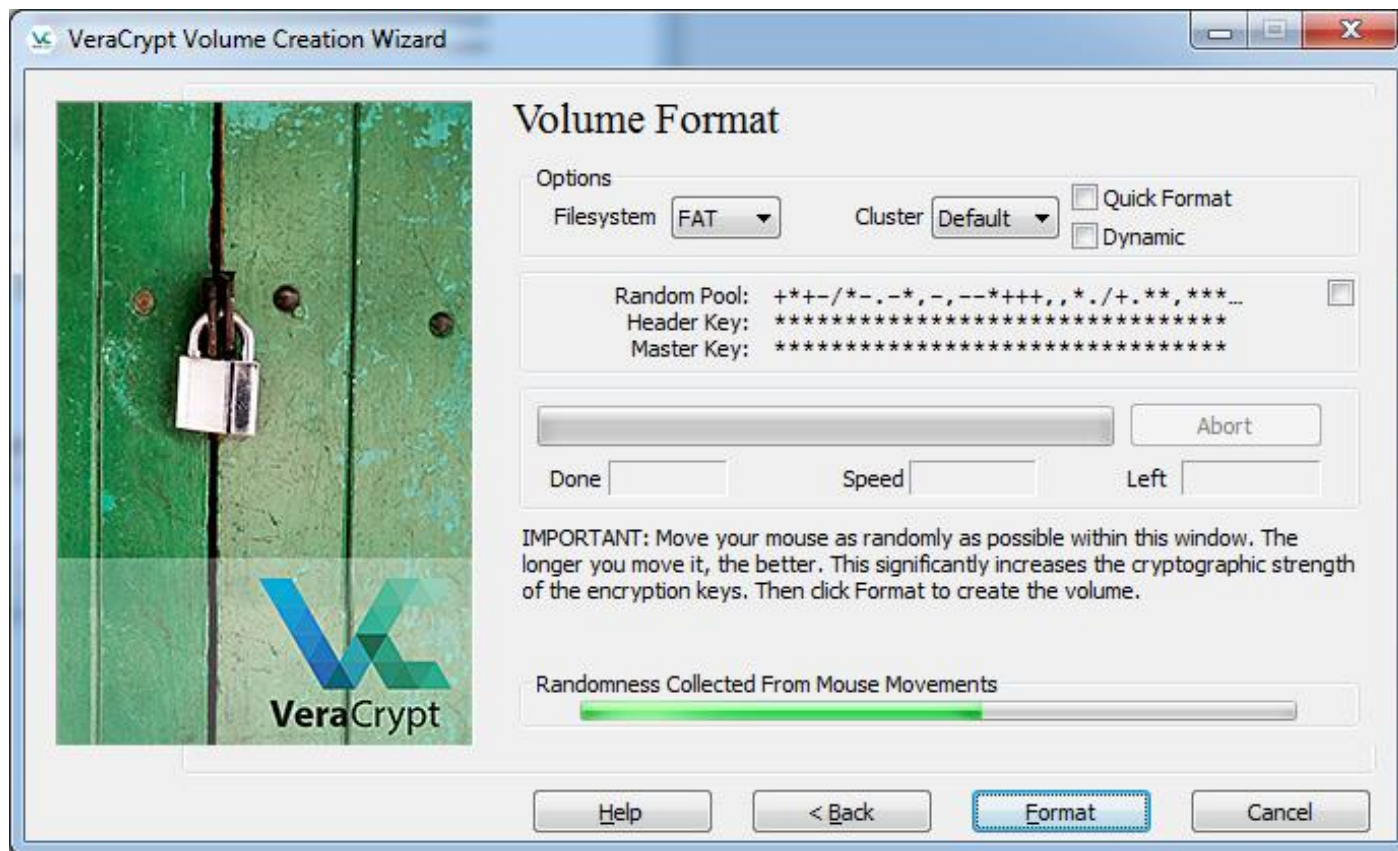
Geradores pseudo-aleatórios

- Evitam necessidade de capturar entropia “bruta” repetidamente
 - Na prática: ganhos de desempenho
- Basicamente:
 - Coleta-se entropia bruta (de várias fontes!) para criar/atualizar **semente** de tamanho adequado.
 - Semente deve ser **mantida secreta**
 - Usa-se algoritmo determinístico para gerar uma sequência “**indistinguível**” de **bits aleatórios**.
- Algoritmos padrão: NIST-SP800-90A-Rev 1
 - Revisão removeu Dual_EC_DRBG (backdoor da NSA)



Entropia: exemplo prático

- Veracrypt: acúmulo de entropia fornecida pelo usuário (movimento do mouse)



Entropia: contra-exemplos

- Baixa entropia é recorrente na literatura... ☹
 - Debian OpenSSL (2008): chaves dependentes apenas de process-id e arquitetura de hardware
 - Urna eletrônica brasileira (2012): recuperação da ordem dos votos registrados na urna (RDV)
 - Chaves RSA geradas por dispositivos de rede (2012): repetição de números aleatórios (fator primo compartilhado), permitindo recuperação da chave privada.
 - Problema observado também em estudos em 2013 (smart cards), 2015 (servidores HTTPS) e 2017 (chaves Tor)
 - “Brain-wallet” (2015+): chaves privadas geradas a partir de senhas, facilitando roubo de criptomoedas





Blockchain, Criptomoedas & Tecnologias Descentralizadas

Criptografia simétrica

Geração de números aleatórios

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br
Escola Politécnica, Universidade de São Paulo

Referências

- W. Stallings, L. Brown “Computer Security Principles and Practice – 2nd/3rd/4th edition”. Prentice-Hall, ISBN: 0-13-277506-9. 2011/2015/2018.
 - Em português: W. Stallings, L. Brown. “Segurança de Computadores - Princípios e Práticas” (2ª Ed), Elsevier, 2014
- W. Stallings: “Cryptography and Network Security” (6th/7th Ed.), Prentice-Hall 2013/2016.
 - Em português: W. Stallings: “Criptografia e Segurança de Redes” (6ª Ed.), Pearson-Prentice-Hall (2014).
- S. Wykes. Criptografia Essencial: A Jornada do Criptógrafo, 1a ed. Elsevier, 2016.
- A. Narayanan, J. Bonneau, E. Felten. "Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction". Princeton University Press, 2016. ISBN: 0691171696. Available: https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf?a=1