Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Visualização de *hashes* com imagens e animações

Animando hashes

Jorge Miguel Ribeiro

Monografia Final

mac 0499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. José Coelho de Pina

São Paulo
05 de dezembro de 2019

A todos que se importam e continuam tentando.

Agradecimentos

Agradeço a várias pessoas que me ajudaram a fazer esse trabalho. Muitas eu nem sei o nome, mas vou colocar todas que lembrar.

Agradeço especialmente ao Rodrigo, que foi quem me convenceu a mudar para o curso do BCC e está presente na minha vida há anos e quero que esteja por muitos mais; também por ter me ajudado a não desistir e todas as vezes em que entrei em crise.

Agradeço também a minha mãe pela ajuda quando eu estava sem energia. Temos nossos momentos difíceis mas o importante é nunca desistir.

Agradeço ao Professor José Coelho, pela orientação e entusiasmo que ele injetou nesse trabalho. Também agradeço à Professora Nina, que aguentou responder todas as dúvidas chatas que eu tinha!

Agradeço também às seguintes pessoas, que acabaram me ajudando: Fábio Arrebola, José Newton, Gabriel Wallace, Gabriel Kobayashi, Marilda, Carolina Davanzzo, Kilder, Felipe. Também agradeço às outras pessoas que responderam a pesquisa, que foi espalhada pela Carolina e Marilda.

Também agradeço às outras pessoas que me apoiaram nessa jornada, mesmo que não tenham me apoiado especificamente nesse trabalho.

Resumo

Jorge Miguel Ribeiro. **Visualização de *hashes* com imagens e animações: *Animando hashes*.** Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

A tarefa de comparar *hashes* é uma operação importante na segurança digital, porém tediosa e propensa a erros quando os *hashes* estão na forma de *strings* hexadecimais. Com base em tentativas de melhorar a qualidade de comparação de *hashes* por meio de imagens, esse trabalho desenvolveu um novo esquema de visualização de *hashes* com animações. O intuito do uso de animações é tornar possível a transmissão de mais informação através do eixo temporal, sem poluição visual. O esquema de visualização foi criado utilizando sequências formadas por *hashes* de *hashes* combinados com *salts*, permitindo adicionar mais elementos facilmente sem quebra de compatibilidade. Durante o desenvolvimento do esquema foi realizada uma pesquisa com uma amostra de usuários para verificar se algum elemento não tinha sua variação percebida, e a maioria dos elementos foi bem distinguida, com exceção de um único, que foi removido do resultado final. O trabalho produziu uma biblioteca Javascript que utiliza 4 letras e 4 ícones SVG para gerar animações de 2 segundos que transmitem por volta de 48 bits de uma sequência derivada do *hash* original, além de um protótipo de extensão do Firefox que utiliza essa biblioteca para mostrar a estampa do certificado `https` utilizado em uma página web.

Palavras-chave: Hash visual. Segurança digital. Usabilidade.

Abstract

Jorge Miguel Ribeiro. **Hash visualization using images and animations: *Animating hashes*.** Undergraduate Thesis (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2019.

The task of comparing hashes is important in digital security, yet tedious and prone to error when the hashes are in the form of hexadecimal strings. Based on works that try and improve hash comparison using images, this work developed a new hash visualization scheme using animations. The objective in using animations is to make possible the transmission of more information through the temporal axis, without visual pollution. The visualization scheme was created using sequences made up by hashes of hashes combined with salts, allowing easy extension without breaking compatibility. A survey was applied to some users during the development of the scheme to verify if any elements had its variation go unnoticed and the majority of them was noticed, except one, which was removed from the final result. This work produced a Javascript library that uses 4 letters and 4 SVG icons to generate 2 second animations that transmit around 48 bits of a sequence derived from the original hash, as well as a Firefox extension that uses this library to show the animated visual stamp of the `https` certificate used in a given webpage.

Keywords: Visual hash. Digital security. Usability.

Lista de Figuras

Sumário

Capítulo 1

Introdução

A comunicação segura e privada pode ser considerada um dos direitos fundamentais das civilizações ocidentais, e é um direito que deve se estender à internet. A comunicação segura através da internet exige conhecimento técnico que permita verificar alguns detalhes que interrompem a criptografia utilizada para estabelecer a mesma — e o uso apropriado das medidas de segurança pode evitar que governos espiem toda comunicação de seus usuários de maneira indiscriminada, como o governo do Cazaquistão tem tentado fazer desde 2015 [28].

A criptografia é usada na codificação, decodificação e troca segura de mensagens — e é utilizada de forma quase invisível no mundo digital, protegendo a privacidade e a possibilidade de comunicações pessoais confidencias dos usuários.

A forma mais visível da criptografia para usuários comuns é na transmissão segura de dados ao acessar *websites* com o protocolo `https`. Esse protocolo usa técnicas criptográficas para estabelecer não só a segurança dos dados trafegados como fornecer uma garantia da autenticidade do *website* sendo acessado. Infelizmente, poucos usuários sabem exatamente a importância do protocolo `https` e como a validade de um certificado é verificada - não sem motivo, pois o assunto é complicado.

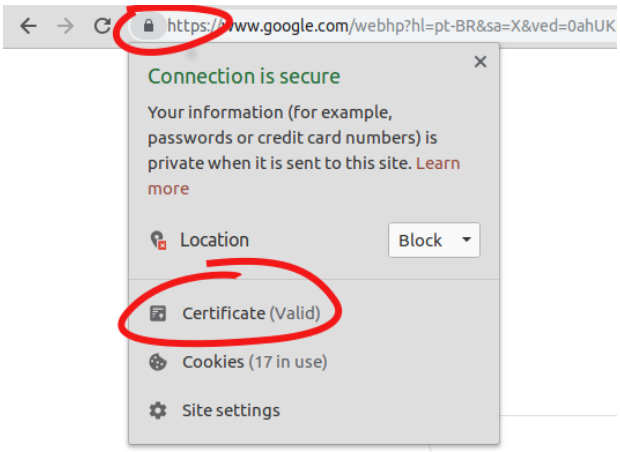


Figura 1.1: *Exemplo de uso do protocolo* `https` *no navegador Google Chrome 73*

O uso de `https` para estabelecimento da autenticidade da identidade de *websites* evita uma gama enorme de ataques de roubo de informações, especialmente ataques que roubam login e senha de um site legítimo, como *internet banking* ou *e-mail* e até mesmo informações de cartões de crédito.

Entretanto, toda a segurança do protocolo `https` depende de algumas suposições, por exemplo: o sistema operacional é confiável, nenhum certificado raiz indevido foi instalado, não foi vazado a chave de comunicação etc. A maior parte das pessoas supõe que um ataque de roubo dos dados não inclui um ataque físico — ou mesmo um ataque social, como roubo de informações por telefone que permitam o acesso indevido a alguma conta *online*.

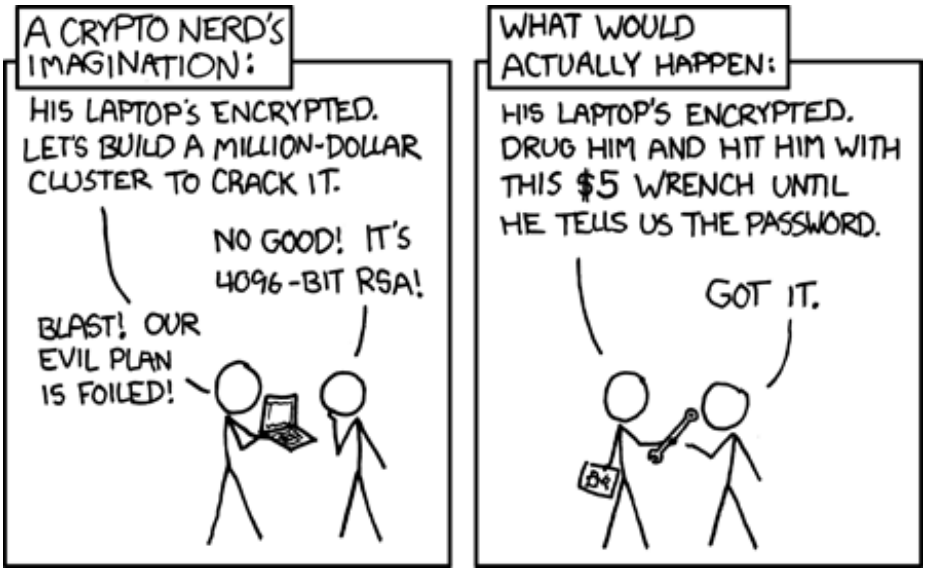


Figura 1.2: *Tirinha representando um ponto fraco na proteção digital de dados — o ser humano. Fonte: https://xkcd.com/538/.*

Esse trabalho teve como objetivo a criação de uma extensão de navegador que permite gerar uma "estampa visual" única a partir do certificado de um site — ou qualquer outro dado. O primeiro caso que permite validar de forma rápida e visual a ocorrência de ataques de substituição de site, como alguém fingindo ser um banco ou um governo substituindo o certificado original por outro, ao comparar a estampa gerada em um sistema conhecido e confiável com a estampa gerada no sistema em uso — ou mesmo com a estampa gerada em outro dia, para *websites* importantes. Também é possível utilizar a biblioteca desenvolvida para a comparação de chaves públicas, pareamento seguro de dispositivos ou mesmo verificação visual de uma senha digitada.

Junto a este texto está disponível o algoritmo e código fonte desenvolvidos, disponibilizados no domínio público com uma licença MIT [24], permitindo a reutilização em trabalhos e aplicações futuras. Além disso, estão disponíveis os dados e análises sobre a qualidade e quantidade de informação visual transmitida pela estampa.

Este trabalho apresenta no capítulo 2 uma revisão sobre *hashes* criptográficos. No capítulo 3 há uma revisão e discussão sobre assinaturas digitais e como elas se relacionam com `HTTPS` e *hashes*. A seguir, no capítulo 4 há uma discussão sobre como comparar *hashes* por meio de imagens para melhorar a segurança — e como animações podem permitir a transmissão de mais

informação visual sem aumentar a poluição visual. Por fim, no capítulo 5 há considerações sobre a biblioteca e protocolo desenvolvido, assim como possíveis melhorias.

Capítulo 2

Criptografia no mundo contemporâneo

A criptografia consiste em um conjunto de técnicas e algoritmos com o objetivo de proteger informação — seja garantindo a integridade ou autenticidade da mesma, protegendo seu transporte ou prevenindo acessos indevidos.

Dentre as diversas técnicas criptográficas, cada uma com suas vantagens e desvantagens, há duas categorias que se destacam por seus usos distintos e, ao mesmo tempo, complementares: a criptografia simétrica e a criptografia assimétrica.

2.1 Simétrica

O principal objetivo da criptografia simétrica é ofuscar uma mensagem, idealmente evitando o entendimento da mesma por pessoas não autorizadas.

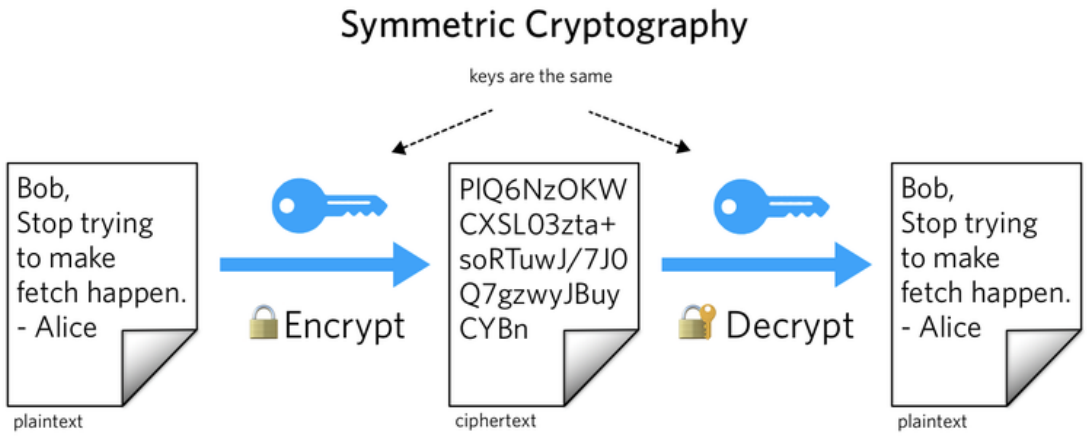


Figura 2.1: *Uso de criptografia simétrica para transmitir uma mensagem de forma segura. Imagem retirada de What is Public Key Cryptography? [20]*

O processo de criptografar uma mensagem com um algoritmo de criptografia simétrica utiliza duas entradas: a mensagem original chamada de *texto plano*, e uma sequência de *bytes* chamada de *chave*. O algoritmo produz uma sequência opaca, aparentemente sem sentido, chamada de *texto cifrado* – o qual pode ser transportado sem medo de acesso indevido pois é indecifrável sem a chave, que normalmente é um *segredo* conhecido apenas pelas entidades autorizadas.

Para decifrar o texto cifrado é necessário o uso da chave que cifrou a mensagem original.

Os algoritmos de criptografia simétrica (ou cifra, ou cifragem simétrica) usam a mesma chave (sequência de *bytes* ou letras) para criptografar o texto plano (transformá-lo em texto cifrado) e decifrar o texto cifrado (transformando-o em texto plano).

Alguns dos algoritmos de cifra simétrica são:

- Cifra de César, um método de criptografia utilizado por Júlio César em suas cartas [26]. Esse método remapeia as letras da mensagem de acordo com a chave utilizada;
- ROT13, um caso específico da Cifra de César [29], utilizado até hoje em alguns fóruns *online*;
- Máquina Enigma, um dispositivo mecânico utilizado para criptografar mensagens durante a segunda guerra mundial;
- Cifra XOR, um método que aplica a operação XOR entre os *bytes* do texto plano e os *bytes* da chave.

Cifragem com ROT13

A cifragem com ROT13 é uma operação interessante, por dois motivos: não existe nenhuma chave para cifrar ou decifrar as mensagem; e a operação de cifragem é idêntica à operação de decifragem — ou seja, basta aplicar a operação novamente para se obter a mensagem original.

A cifragem com ROT13 consiste, basicamente, em "avançar" cada letra em 13 posições no alfabeto, retornando ao início caso não haja mais letras. Por exemplo, a primeira letra do alfabeto — A — vira a letra N, a décima-quarta letra do alfabeto.

As mensagens podem ser cifradas em ROT13 no Linux com um comando *shell*:

```
$ echo "Bacharelado em Ciencia da Computacao do IME USP" |
       tr "A-Za-z" "N-ZA-Mn-za-m"
Onpunerynqb rz Pvrapvn qn Pbzchgnpnb qb VZR HFC
$ echo "Onpunerynqb rz Pvrapvn qn Pbzchgnpnb qb VZR HFC" |
       tr "A-Za-z" "N-ZA-Mn-za-m"
Bacharelado em Ciencia da Computacao do IME USP
```

Pode-se notar que a operação é revertida ao aplicar duas vezes a cifragem — o que torna raros usos práticos em operações que requerem segurança real. Pode-se dizer, então, que a cifragem com ROT13 falha em criar uma cifra real, pois o método é bem conhecido e pode ser detectado e revertido usando um dos métodos mais simples de criptoanálise: análise de frequência de letras [3].

Cifra XOR

A cifra XOR, ao contrário de ROT13, necessita de uma chave. Para cifrar a mensagem, a mesma é transformada em *bits* e em cada *bit* é aplicada a operação lógica *eXclusive OR* com um bit da chave. O processo de decifragem, é idêntico ao processo de cifragem — a mensagem original é recuperada desde que seja aplicada a cifra XOR com a mesma chave nos dois processos.

Segue um exemplo de uso da cifra XOR [baseado na wikipedia]. Aqui ciframos o texto `WIKI` (codificado em ASCII com 8 *bytes*) com a chave 11110011:

Processo de cifragem:

$$\begin{array}{rcccc} & \overbrace{01010111}^{\text{W}} & \overbrace{01101001}^{\text{I}} & \overbrace{01101011}^{\text{K}} & \overbrace{01101001}^{\text{I}} \\ \oplus & 11110011 & 11110011 & 11110011 & 11110011 \\ \hline = & 10100100 & 10011010 & 10011000 & 10011010 \end{array}$$

Processo de decifragem:

$$\begin{array}{rcccc} & 10100100 & 10011010 & 10011000 & 10011010 \\ \oplus & 11110011 & 11110011 & 11110011 & 11110011 \\ \hline = & \underbrace{01010111}_{\text{W}} & \underbrace{01101001}_{\text{I}} & \underbrace{01101011}_{\text{K}} & \underbrace{01101001}_{\text{I}} \end{array}$$

2.2 Assimétrica

O principal objetivo da criptografia assimétrica também é ofuscar uma mensagem — entretanto, o processo é realizado de forma que a mensagem cifrada pode ser lida por agentes que possuem uma chave que permite apenas decifrar a mensagem.

Assim, a criptografia assimétrica utiliza duas chaves: uma chave privada, que em geral é utilizada para cifrar a mensagem, e uma chave pública, em geral utilizada para decifrar uma mensagem cifrada pela chave privada. Entretanto, como visto na figura 2.2 o processo inverso também é utilizado, para enviar mensagens que apenas quem possui a chave privada consegue decifrar.

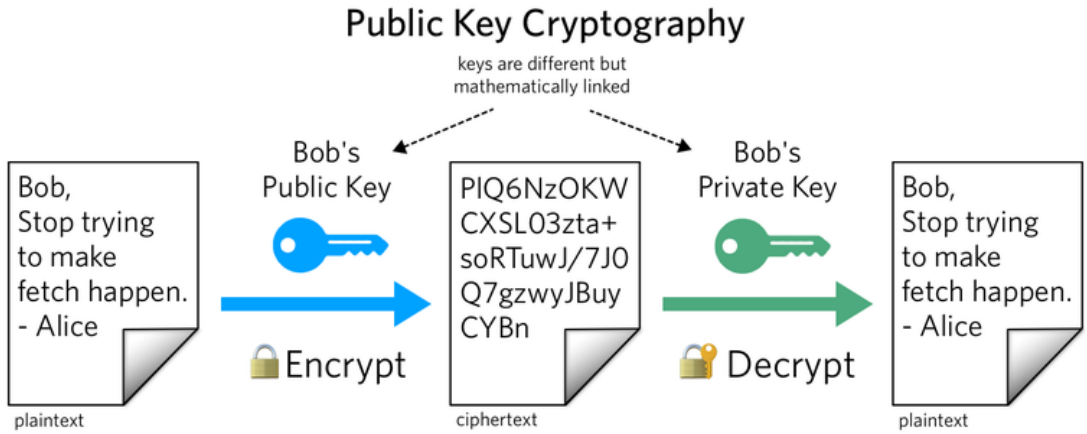


Figura 2.2: *Uso de criptografia assimétrica para transmitir uma mensagem cuja autoria pode ser verificada. Imagem retirada de What is Public Key Cryptography? [20]*

Alguns dos algoritmos de criptografia assimétrica mais conhecidos incluem: o protocolo de troca de chave Diffie-Hellman, o sistema de Ron Rivest, Adi Shamir e Leonard Adleman (RSA), o sistema ElGamal e técnicas de curvas elípticas.

Uma combinação de algoritmos assimétricos e simétricos permite estabelecer canais de comunicação segura mesmo na *internet*, onde todos *bytes* transmitidos podem ser monitorados. O protocolo `https`, essencialmente, utiliza as chaves públicas para estabelecer um segredo simétrico comum entre o servidor e o navegador — isso é possível pois uma informação cifrada com uma chave pública só pode ser decifrada com a chave privada correspondente.

Além do estabelecimento de canais de comunicação seguros, a assimetria existente nas operações com chaves assimétricas permite a criação de sistemas de assinatura digital — protocolos que permitem validar a autoria e integridade de uma mensagem, como o protocolo `PKCS7` [12] e o padrão *Cryptographic Message Syntax* (CMS) [10].

2.3 Hashes criptográficos

Além dos algoritmos de cifragem, existe outra operação essencial ao uso da criptografia, especialmente na verificação de integridade de dados e verificação de assinaturas digitais: funções de hash criptográficos.

Funções de hash são funções que mapeiam dados ou entradas de tamanhos variáveis para valores com tamanhos fixos. Ou seja, sempre é produzido um valor padronizado que depende unicamente da entrada.

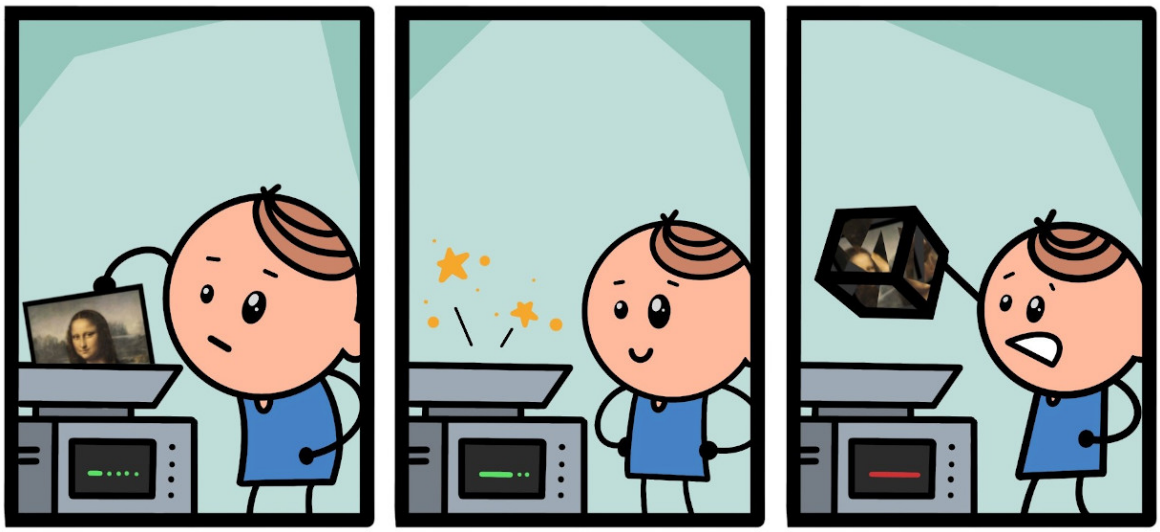


Figura 2.3: *Ilustração da transformação de uma entrada em um valor padronizado através de uma função de hash. Tirinha desenhada por kdrawtoons através do site https://www.fiverr.com*

Funções de hash criptográficas, por outro lado, pertencem a uma categoria especial — são funções de hash com alguns requisitos extras [27]:

- *resistência pré-imagem*: dado um hash de valor h, deve ser difícil encontrar qualquer entrada m tal que $hash(m) = h$. Essencialmente, é desejado que a função de *hash* seja uma função *one-way* ou irreversível — dada uma saída, é extremamente difícil encontrar a entrada que produziu a mesma;
- *resistência de segunda pré-imagem*: dada uma entrada m_1, deve ser difícil encontrar uma entrada m_2 tal que $hash(m_1) = hash(m_2)$. Ou seja, dada uma entrada, deve ser difícil encontrar uma entrada diferente que produza a mesma saída;
- *resistência de colisão*: deve ser difícil encontrar duas mensagens diferentes m_1 e m_2 tal que o $hash(m_1) = hash(m_2)$. Ou seja, é desejável que entradas 'aleatórias' do mundo real tenham *hashes* diferentes.

Em geral os *hashes* criptográficos são representados em hexadecimal expandido, para maior facilidade de comparação: por exemplo, o *hash* com a sequência de *bytes* `100011010010110111111111111011101101001000100110010000111100101101011110111101011001000110100011011001010001000101111101001001101111010111000100001011000110110` é representado geralmente como `11a5bffbb48990f2d7bd6468d9445f49bd711636`.

As funções de hash são essenciais em vários processos criptográficos, especialmente no estabelecimento de integridade e verificação de assinaturas digitais, sendo parte integral do protocolo `https`, que usa certificados assinados por outros certificados — essas assinaturas precisam ser validadas para estabelecer confiança de um *website*.

Várias das funções de *hash* criptográficos mais utilizadas tiveram seus algoritmos e implementações de referência publicadas pelo governo dos Estados Unidos, sendo os mais utilizados e

conhecidos: SHA-1, que produz saída de 20 *bytes*, SHA-256, que produz saída de 32 *bytes* e SHA-512, que produz *hashes* de 64 *bytes*.

O algoritmo SHA-1

O algoritmo SHA-1 foi publicado pela primeira vez em 1995 [30] e consiste em 80 rodadas de expansão e "mistura" dos dados de entrada, produzindo uma saída com 20 *bytes*.

Durante vários anos foi considerado o algoritmo padrão de assinaturas digitais, entretanto, após a construção de ataques de colisão de *hashes* gerados com SHA-1 em laboratório [21] e ataques de arquivos PDF reais com o mesmo *hash* SHA-1 [22], especialistas recomendaram a mudança para algoritmos mais fortes, como SHA-256 e SHA-512 — tanto que os principais navegadores deixaram de aceitar certificados SSL com assinaturas usando SHA-1 em 2017 [11][19][9].

Abaixo seguem alguns exemplos das saídas produzidas pelo algoritmo SHA-1:

```
$ echo "IME USP" | sha1sum
11a5bffbb48990f2d7bd6468d9445f49bd711636  -
$ echo "IME-USP" | sha1sum
357d1b9f3fa88804b14b642af25f3304eb42dfb4  -
```

Um trecho da implementação de referência do SHA-1 pode ser visto no trecho 2.1.

```
1  ...
2    for (t = 16; t < 80; t++)
3      W[t] = SHA1_ROTL(1, W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16]);
4
5    A = context->Intermediate_Hash[0];
6    B = context->Intermediate_Hash[1];
7    C = context->Intermediate_Hash[2];
8    D = context->Intermediate_Hash[3];
9    E = context->Intermediate_Hash[4];
10
11   for (t = 0; t < 20; t++) {
12     temp = SHA1_ROTL(5,A) + SHA_Ch(B, C, D) + E + W[t] + K[0];
13     E = D;
14     D = C;
15     C = SHA1_ROTL(30,B);
16     B = A;
17     A = temp;
18   }
19
20   for (t = 20; t < 40; t++) {
21     temp = SHA1_ROTL(5,A) + SHA_Parity(B, C, D) + E + W[t] + K[1];
22     E = D;
23     D = C;
24     C = SHA1_ROTL(30,B);
25     B = A;
26     A = temp;
27   }
28
29   for (t = 40; t < 60; t++) {
30     temp = SHA1_ROTL(5,A) + SHA_Maj(B, C, D) + E + W[t] + K[2];
```

2.3

```
31        E = D;
32        D = C;
33        C = SHA1_ROTL(30,B);
34        B = A;
35    }
36  ...
```

Listing 2.1: *Trecho do código de referência do algoritmo SHA-1 [8]*

A representação visual de uma rodada do algoritmo SHA-1 pode ser encontrada na figura 2.4; nela podemos ver como todas as partes do valor original estão interligadas e afetam a próxima rodada — gerando um efeito cascata, pelo qual a mudança de um único *byte* da entrada original leva a uma saída completamente diferente.

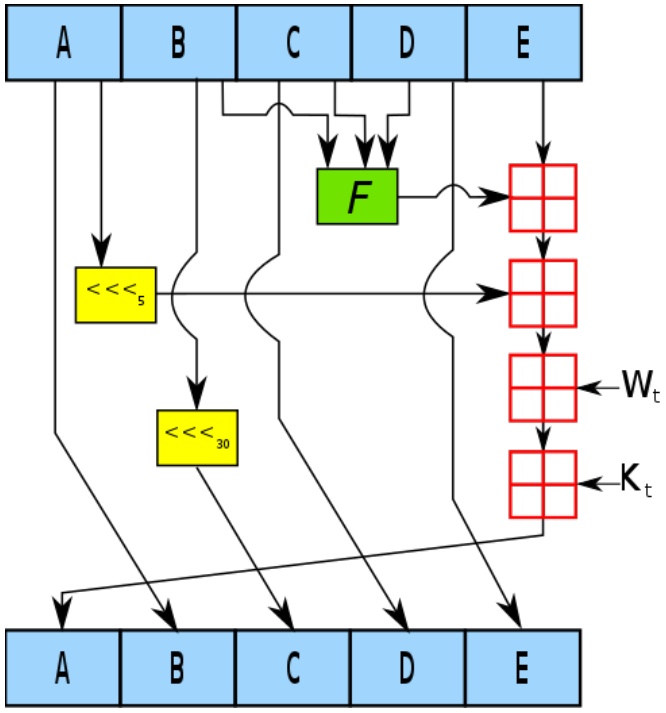


Figura 2.4: *Representação visual de uma rodada de mistura de dados do algoritmo SHA-1. A mistura envolve operações de* shift *de bytes, soma módulo* 2^{32} *e operações de expansão. Fonte: Commons [6]*

O algoritmo SHA-256

O algoritmo SHA-256 faz parte da família de algoritmos SHA2. Ele foi publicado pela primeira vez em 2001 [31] e consiste em 64 rodadas de expansão e "mistura" dos dados de entrada similares à do SHA-1, porém produzindo uma saída maior, com 32 *bytes*.

Abaixo estão alguns exemplos de saídas produzidas pelo algoritmo SHA-256:

```
$ echo "IME USP" | sha256sum
30c260022e5b2527b4093c4cafca25ff27fcc10862a7f31d4258801be2f67f1d  -
$ echo "IME-USP" | sha256sum
36f1add0814768289892144a6336f4317efc0c302371fb0556d3413c7e62df35  -
```

Parte da implementação do código de referência das rodadas do SHA-256 pode ser visto no trecho 2.2:

```
1  ...
2    uint32_t W[64]; /* Word sequence */
3    uint32_t A, B, C, D, E, F, G, H; /* Word buffers */
4
5    /*
6     * Initialize the first 16 words in the array W
7     */
8    for (t = t4 = 0; t < 16; t++, t4 += 4)
9      W[t] = (((uint32_t)context->Message_Block[t4]) << 24) |
10            (((uint32_t)context->Message_Block[t4 + 1]) << 16) |
11            (((uint32_t)context->Message_Block[t4 + 2]) << 8) |
12            (((uint32_t)context->Message_Block[t4 + 3]));
13
14   for (t = 16; t < 64; t++)
15     W[t] = SHA256_sigma1(W[t-2]) + W[t-7] +
16          SHA256_sigma0(W[t-15]) + W[t-16];
17
18   A = context->Intermediate_Hash[0];
19   B = context->Intermediate_Hash[1];
20   C = context->Intermediate_Hash[2];
21   D = context->Intermediate_Hash[3];
22   E = context->Intermediate_Hash[4];
23   F = context->Intermediate_Hash[5];
24   G = context->Intermediate_Hash[6];
25   H = context->Intermediate_Hash[7];
26
27   for (t = 0; t < 64; t++) {
28     temp1 = H + SHA256_SIGMA1(E) + SHA_Ch(E,F,G) + K[t] + W[t];
29     temp2 = SHA256_SIGMA0(A) + SHA_Maj(A,B,C);
30     H = G;
31     G = F;
32     F = E;
33     E = D + temp1;
34     D = C;
35     C = B;
36     B = A;
37     A = temp1 + temp2;
38   }
39 ...
```

Listing 2.2: *Trecho do código de referência do algoritmo SHA-256 [8]*

A representação visual de uma rodada do algoritmo SHA-256 pode ser vista na figura 2.5. Da mesma maneira que as rodadas do SHA-1, as operações estão interligadas de forma a gerar um efeito cascata na saída final.

2.3

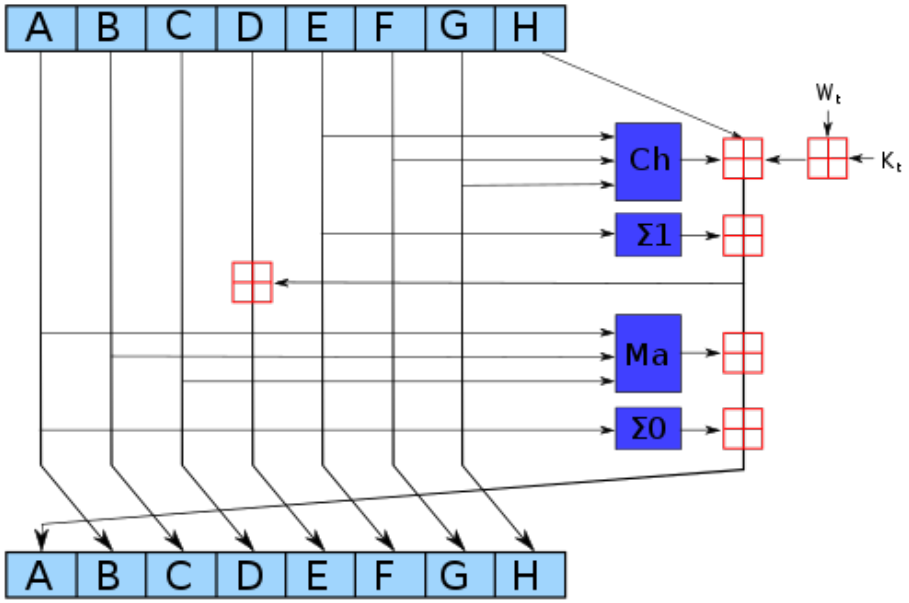


Figura 2.5: *Representação visual de uma rodada de mistura de dados do algoritmo* SHA-256. *A mistura envolve operações de* shift *de bit, soma módulo* 2^{32} *e operações de expansão. Fonte: Commons [7]*

Por ser o padrão atual dentre os algoritmos de *hash* criptográficos, a extensão de navegador desenvolvida neste trabalho utiliza primariamente o algoritmo SHA-256 para a geração dos parâmetros a partir do valor inicial.

Capítulo 3

Segurança digital

A segurança das comunicações no mundo digital depende da chamada Infraestrutura de Chave Pública (PKI). A PKI consiste de várias entidades, protocolos e elementos de *software* e *hardware* que, utilizados de maneira orquestrada, permite estabelecer uma cadeia de confiança e verificação de identidade através de certificados e assinaturas digitais.

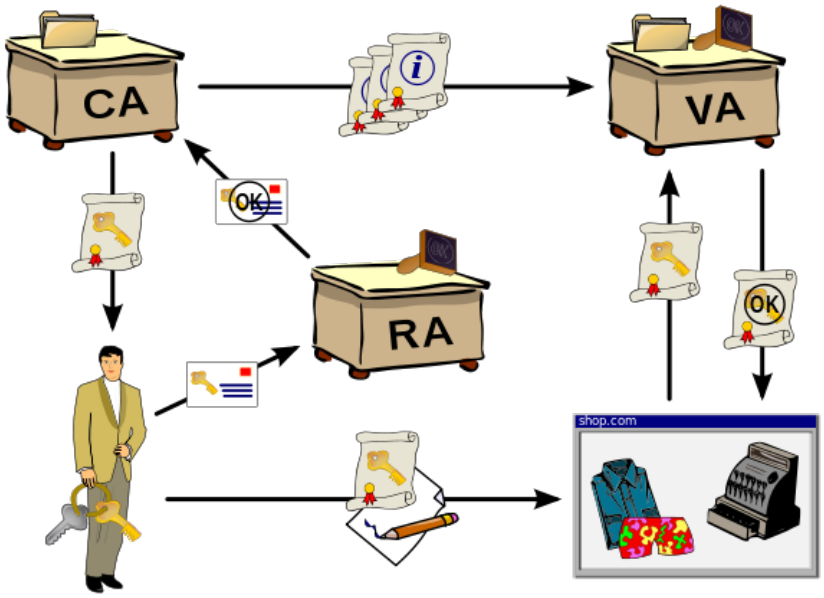


Figura 3.1: *Ilustração dos vários elementos que compões a PKI. Fonte: Commons [5]*

O amplo uso do protocolo `https`, por exemplo, depende da PKI — pois se um certificado de um site não for emitido por uma entidade na qual, de alguma forma, o usuário ou o sistema confia, não há confiança na confidencialidade da transmissão das informações.

Essencialmente, a PKI é uma maneira de tornar a maior parte do processo de estabelecimento de confiança na comunicação quase totalmente algorítmica, permitindo definir protocolos bem estabelecidos e conhecidos para a verificação de identidade no meio digital.

3.1 Criptografia e hashes no cotidiano

A criptografia e hashes aparecem diariamente na internet — especialmente no acesso de *websites* `https` que utilizam assinaturas digitais e certificados. Os certificados usados em `https` essencialmente são assinaturas em cima de outras assinaturas, que acabam formando uma espécie de cadeia de hashes — pois normalmente certificados são identificados por seus *hashes*.

3.2 Formatos de assinaturas e certificados digitais

Certificados e assinaturas digitais são utilizados para o estabelecimento de comunicação `https`, assim como estabelecimento de identidade, autoria ou ciência sobre um documento. Os principais formatos utilizados no protocolo `https` são:

- CMS — utilizado na distribuição de assinaturas
- PEM — utilizado na distribuição de certificados digitais, que por sua vez contém uma assinatura digital no formato CMS

Ambos formatos — CMS e PEM — utilizam o protocolo ASN.1 [25] para definir o formato dos dados. O protocolo ASN.1 é bastante usado para definir estruturas de dados relacionadas a assinaturas e certificados de maneira descritiva e, ainda mais importante, bem definida.

No trecho 3.1 podemos visualizar parte da definição do formato CMS, especificamente como salvar informações sobre os assinantes na estrutura da uma assinatura digital. Um exemplo real da estrutura ASN.1 de um certificado utilizado para comunicação `https` pode ser vista na figura 3.2.

```
SignedData ::= SEQUENCE {
   version CMSVersion,
   digestAlgorithms DigestAlgorithmIdentifiers,
   encapContentInfo EncapsulatedContentInfo,
   certificates [0] IMPLICIT CertificateSet OPTIONAL,
   crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
   signerInfos SignerInfos }
...
SignerInfos ::= SET OF SignerInfo
...
SignerInfo ::= SEQUENCE {
   version CMSVersion,
   sid SignerIdentifier,
   digestAlgorithm DigestAlgorithmIdentifier,
   signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
   signatureAlgorithm SignatureAlgorithmIdentifier,
   signature SignatureValue,
   unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }

SignerIdentifier ::= CHOICE {
   issuerAndSerialNumber IssuerAndSerialNumber,
   subjectKeyIdentifier [0] SubjectKeyIdentifier }
```

Listing 3.1: *Trecho da definição do formato CMS em ASN.1 [10]*

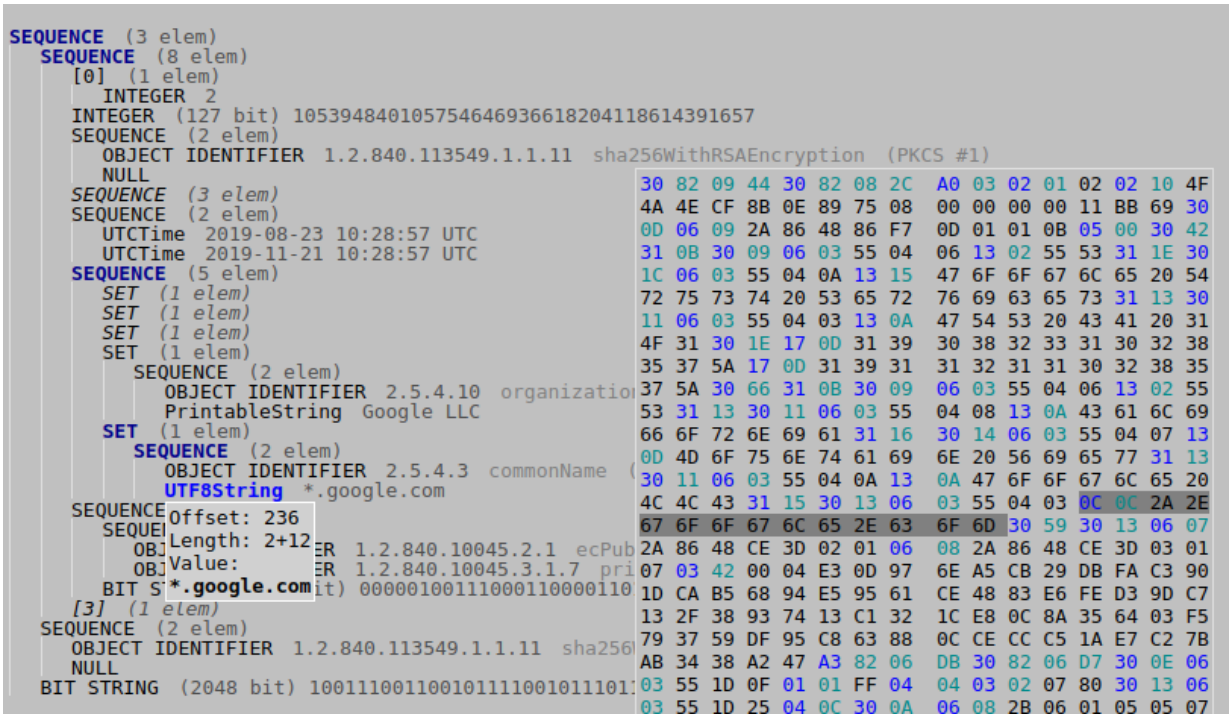


Figura 3.2: *Certificado SSL dos principais sites da Google, visualizado com o decodificador online https://lapo.it/asn1js/. O item destacado em hexadecimal e com a* popup *é o endereço do site principal da entidade.*

3.3 Assinaturas

As assinaturas são geradas em uma estrutura de dados específica (por exemplo, CMS [10]) — essa estrutura define onde as informações assinadas devem ser guardadas, como elas devem ser codificadas, informações sobre o assinante e quais informações devem ser utilizadas no cálculo do conteúdo assinado.

De maneira resumida, o conteúdo a ser assinado é utilizado para gerar um hash criptográfico. Esse hash, por sua vez, é colocado em uma estrutura mínima de distribuição de assinatura — parte do formato CMS — que contém apenas os dados ditos "informações assinadas", como hash do conteúdo assinado, informações sobre o momento da assinatura, comprovantes de validade da chave no momento da assinatura etc.

Essa estrutura mínima, finalmente, é utilizada para gerar outro hash criptográfico — chamado de *message-digest* — que é então criptografado com a chave privada do assinante.

Finalmente, a estrutura final da assinatura é gerada, contendo: a estrutura mínima dos dados assinados, o hash dos dados assinados criptografado com a chave privada, informações sobre o assinante como certificado e chave pública do mesmo, conteúdo assinado e quaisquer outras informações relevantes à assinatura.

A estrutura CMS pode ser então verificada por qualquer agente que receba a mesma: uti-

lizando os dados presentes, o agente pode reproduzir o passo-a-passo da geração do hash da estrutura mínima. Tendo chegado a esse hash o agente pode decodificar o hash contido na assinatura, através da chave pública do assinantes.

Tendo os dois valores basta o agente comparar os mesmos para descobrir se o conteúdo ou assinatura não sofreu modificações. Caso os valores sejam iguais, a assinatura é válida.

O fato de uma assinatura ser válida pode ser interpretado de algumas maneiras diferentes, a depender do contexto: quem assinou pode ser o autor do conteúdo ou ter lido e aprovado um contrato contido no mesmo, por exemplo.

3.4 Certificados

Um certificado digital é uma estrutura de dados, geralmente no formato PEM, que é essencialmente equivalente a um RG digital.

Um certificado digital é uma estrutura de dados contendo informações sobre alguma entidade e uma chave pública que corresponde a uma chave privada em propriedade da dita entidade. As informações identificadoras contidas no certificado são assinadas por uma Autoridade Certificadora (que tem seu próprio certificado) que é dita ter emitido o certificado que ela assinou.

Um exemplo das informações contidas em um certificado pode ser visto na figura 3.3.

Todos certificados são assinados por algum outro certificado, com exceção de Certificados Raiz, que assinam a si mesmos e podem definir uma cadeia de confiança — caso um sistema confie em um Certificado Raiz, ele confia nas informações de certificados emitidos pelo mesmo, e certificados “netos”, “bisnetos” etc do certificado raiz.

Um certificado é, essencialmente, um RG digital, havendo várias similaridades, detalhadas na Tabela 3.1: a emissão por entidades políticas, o processo de verificação de identidade e o estabelecimento da identidade do portador.

Uma questão interessante e importantíssima sobra após toda verificação algorítmica da validade da assinatura de um certificado: como decidir se um Certificado Raiz é confiável?

3.4

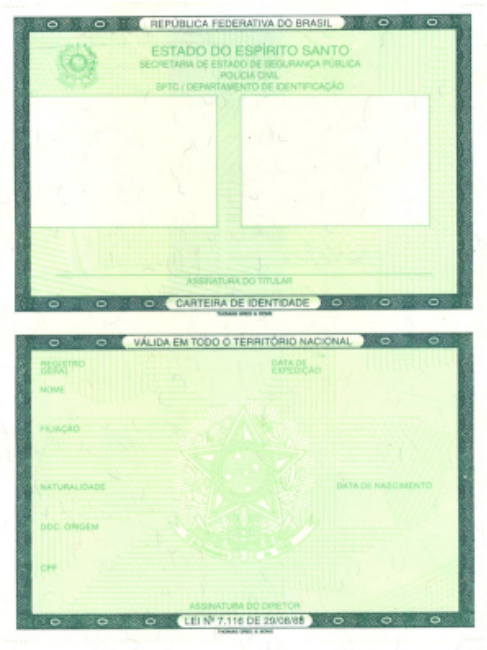
	Item	
	Certificado	RG
		
	Representação textual do certificado `https` do site google.com.br em 11 de setembro de 2019	Modelo de um RG (imagem extraída de Commons [4])
Emitido por	Autoridades certificadoras (ACs)	Institutos de identidade ou delegacias
Identidade	A identidade é verificada comparando documentos, assinatura, endereço de e-mail. No caso de identidade de *websites*, há verificação de desafio	A identidade é verificada por comparação de documentos, assinatura e presença física
Relacionam	Uma chave pública com a identidade do portador da chave privada	Foto e assinatura são relacionados ao nome da pessoa
Verificação	Verificação algorítmica da assinatura do certificado e de seu *status* de revogação	Foto e assinatura podem ser comparados
Roubo	Verificação por *status* de revogação por listas de certificados revogados (CRLs) ou OCSP	Busca de boletins de ocorrência (BOs) mencionando o documento

Tabela 3.1: *Similaridades entre um certificado digital e um RG*

Figura 3.3: *Detalhes de um certificado, visualizados com a ferramenta* `openssl`*. Inspirado por https://smallstep.com/blog/everything-pki.html*

3.5 Elemento humano e confiança

Mesmo contando com as estruturas da PKI, certificados e validação de assinaturas digitais, em algum momento acaba ocorrendo a validação da confiança por alguma parte humana.

Essencialmente, quando se fala em confiança em certificados e na PKI, queremos dizer que o usuário — ou alguma outra entidade na qual este confia — acredita que há baixa probabilidade de um certificado ter sido emitido de forma errônea ou maliciosa pelas Autoridades Certificadoras (ACs) consideradas confiáveis. Os mantenedores desses repositórios de certificados confiáveis utilizam vários métodos para decidir sobre a confiança deles. Por exemplo, existem entidades dedicadas ao rastreio de certificados, entidades que realizam auditorias sobre os certificados emitidos por ACs e os processos de verificação de identidade antes da emissão de um certificado.

Os sistemas de criptografia atuais acabam dependendo, sempre, em algum momento, do estabelecimento inicial de confiança (de ao menos um elemento) por algum ser humano.

Como exemplos da confiança em certificados no cotidiano, temos:

- Os usuários do sistema operacional Windows confiam automaticamente nos certificados raiz do repositório do sistema operacional

- Os usuários do navegador Firefox confiam automaticamente nos certificados confiados pelo navegador
- Os usuários do navegador Chrome também confiam nos certificados presentes na lista de certificados confiáveis do navegador

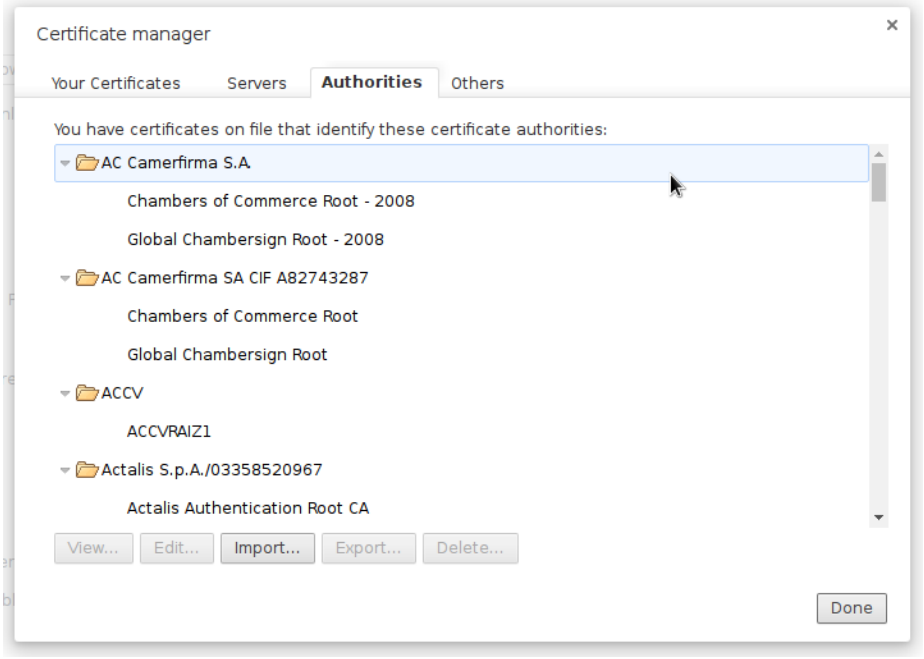


Figura 3.4: *Lista de certificados confiáveis do Chrome 57*

Deve-se notar que as listas dos certificados confiáveis não são criadas automaticamente - elas são mantidas por alguém - um mantenedor - que cura os certificados raiz que podem ser considerados confiáveis pela maioria dos usuários.

Esse processo de manutenção da lista de certificados confiáveis é um processo de absoluta importância - o mantenedor deve verificar todas as informações possíveis sobre o certificado e quem cuida do mesmo, incluindo logs de auditoria de certificados emitidos pela Autoridade Certificadora responsável.

Um dos pontos mais frágeis nesse processo de curagem é o momento da inclusão do certificado no sistema/navegador: deve ser verificada a integridade do mesmo. A verificação da integridade se dá pela validação das informações presentes no certificado, somadas à validação do hash do certificado (SHA-1, geralmente).

Essa validação necessita da comparação de uma ou mais sequências de caracteres “sem sentido”, e conforme Perrig e Song [17] os seres humanos tem dificuldade em validar essas sequências, geralmente verificando apenas o começo e o fim das mesmas.

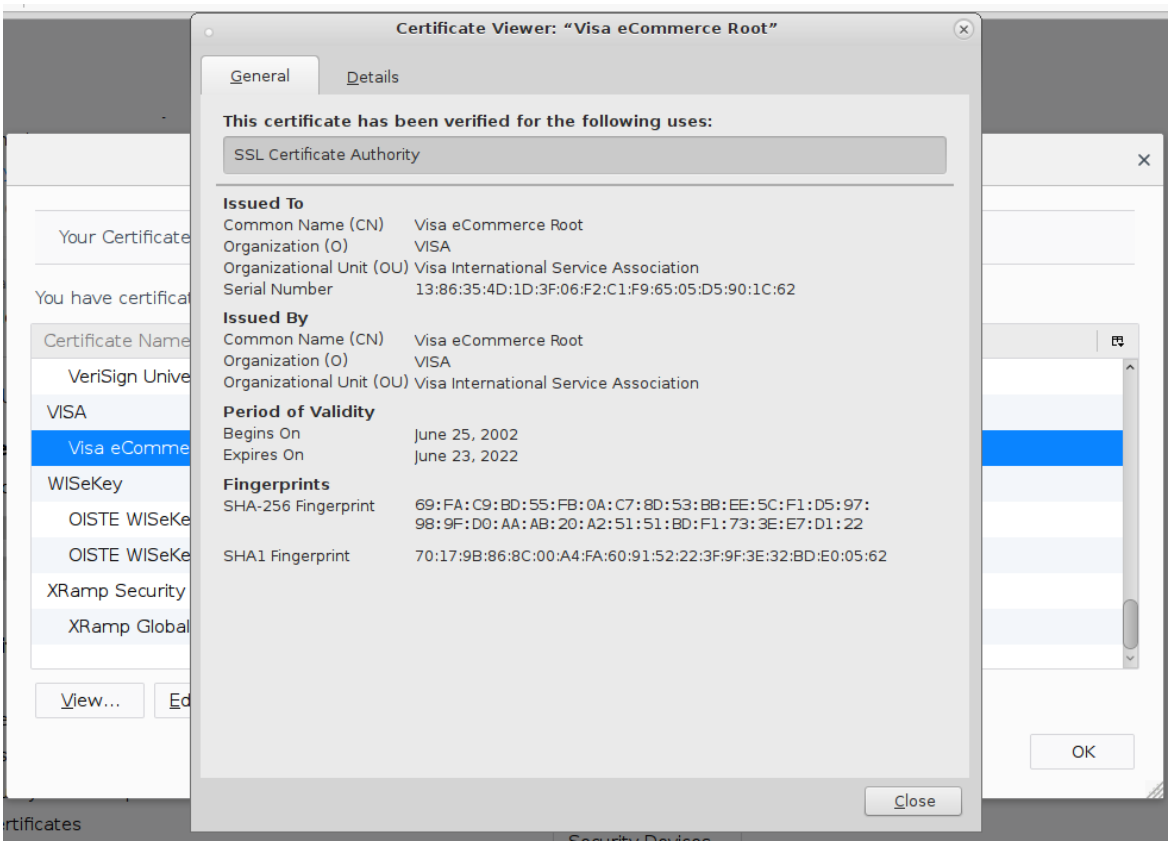


Figura 3.5: *Informações do certificado raiz de SSL da Visa, no Firefox 60. Note os campos **SHA1 Fingerprint** e **SHA-256 Fingerprint** — conforme Perrig e Song [16], os dados necessitam de uma comparação difícil, tediosa e propensa a erros.*

De maneira resumida: sempre é necessário confiar em alguém, mesmo que por *proxy* dos certificados confiáveis do sistema operacional ou do navegador ou de algum outro sistema. Nesses casos acabamos confiando que os mantenedores dos repositórios ou sistemas não tenham sido comprometidos e tenham o bem dos usuários em mente — além disso, há a confiança implícita que todas emissões somente ocorreram após um processo de validação de identidade por parte das ACs.

Também há outra suposição: acabamos confiando que, além de tudo isso, os mantenedores verificaram a integridade dos certificados — e essa validação ocorre por meio de conferências de hash e informações presentes nos certificados.

Entretanto, sempre vale lembrar que toda segurança digital tem limitações e suposições por trás — mesmo o protocolo `https`, super bem difundido, utilizado e desenvolvido, tem várias limitações, conforme o blog *Limitations of HTTPS [13]*.

Por outro lado, as entidades que cuidam dos navegadores tem interesse em manter a segurança e confiança em seus repositórios, e por isso sempre verificam não só as informações dos certificados raiz — também verificam todos certificados encontrados na internet para verificar se não ocorrem emissões errôneas ou mal intencionadas. Isso pode levar à perda de confiança nas Autoridades Certificadoras responsáveis e a consequente remoção dos certificados raiz das listas de certificados confiáveis. Isso aconteceu com a CNNIC, Autoridade Certificadora da China

3.5

Armasu [1] em 2015 e, em 2018, com a Autoridade Certificadora SSL da Symantec [2].

Capítulo 4

Verificação de certificados por imagens

A verificação de assinaturas e certificados ocorre por meio da comparação de *hashes* — strings ENORMES e sem sentido, que são difíceis de serem comparadas por seres humanos. Por exemplo, encontre a diferença entre os hashes:

`e0d31d7599daebee65a15a639736d65dae6963d2`

`e0d31d7599daebee65a15a636736d65bae6963d2`

A principal ideia do trabalho é tornar a verificação de conexões HTTPS uma tarefa mais agradável, para que mais usuários verifiquem suas conexões. Isso pode evitar que indivíduos e governos executem ataques MITM[1] em massa, como o governo do Cazaquistão tem tentado

Figura 4.1: *Hash sendo animado por um programador. Tirinha desenhada por kdrawtoons através do site https://www.fiverr.com*

[1] *Man in the middle* — uma forma de ataque que intercepta a informação original e substitui ou extrai partes da mesma

desde 2015 ([28]).

Entretanto, a comparação de hashes é complicada e vulnerável a alguns ataques. Conforme Perrig e Song [16] “It is a known fact in psychology that people are slow and unreliable at processing or memorizing meaningless strings.” Além disso, conforme Tan et al. [23], após certo tempo comparando hashes sem a ocorrência de ataques, as pessoas tendem a tomar atalhos nas comparações, por exemplo, comparando apenas o início e final dos hashes. Entretanto, isso permite que atacantes gerem certificados ou conteúdos com hashes com apenas alguns bits similares — o que é factível com hardwares atuais.

Uma das ideia propostas para evitar esse problema de fatiga de comparação é transformar hashes em imagens — ideia essa que tem sido desenvolvida há alguns anos [16][14][15][23] e ainda não é *mainstream*.

Conforme Perrig e Song [16] “Humans are good at identifying geometric objects (such as circles, rectangles, triangles, and lines), and shapes in general. We call images, which contain mostly recognizable shapes, regular images. If an image is not regular, i.e. does not contain identifiable objects or patterns, or is too chaotic (such as white noise), it is difficult for humans to compare or recall it.” Ou seja, imagens podem ser mais fáceis de comparar do que strings sem sentido, se as imagens forem bem comportadas.

4.1 Visualizações de *hashes*

Uma das primeiras tentativas bem sucedidas de transformar hashes em imagens ocorreu com o desenvolvimento de Random Art em Perrig e Song [16], que gera imagens através da geração de árvores de expressão derivadas a partir dos bits do hash. Essa árvore de expressões então é avaliada, gerando um valor RGB para cada pixel da imagem.

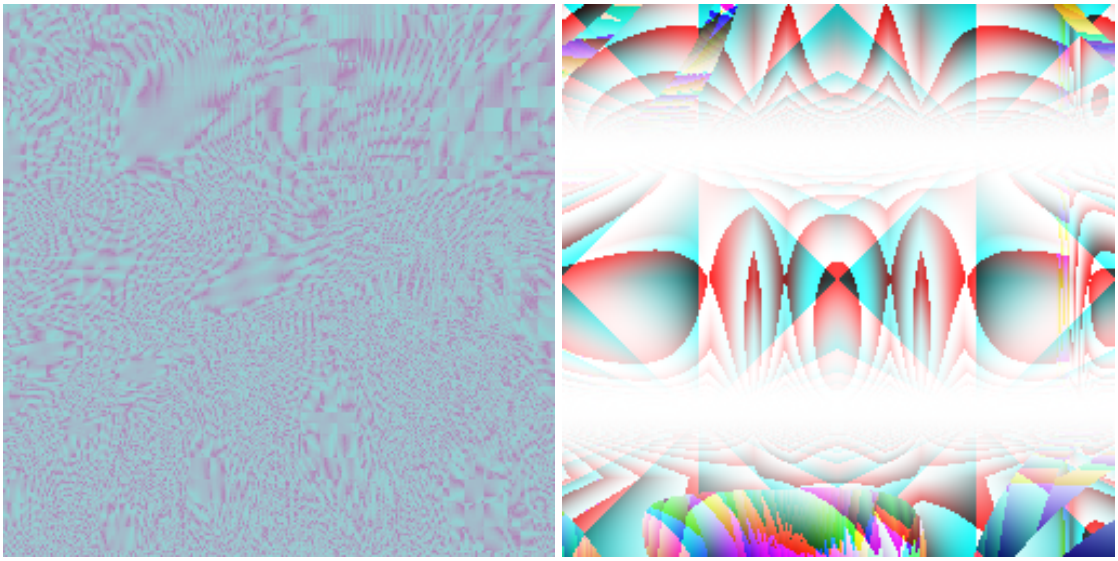


Figura 4.2: *Exemplos de imagens geradas com Random art. À esquerda, imagem gerada pelo site http://www.random-art.org/online/ com a frase "IME USP". À direita, Random art gerado a partir de uma adaptação da implementação disponibilizada em http://math.andrej.com/2010/04/21/random-art-in-python/, usando como semente o hash* `e0d31d7599daebee65a15a636736d65dae6963d2` *e hashes de hashes como o gerador de números aleatórios.*

Outros esquemas tomam uma abordagem diferente e interessante: a montagem de imagens a partir de pedaços menores que se encaixam, como pode ser visto na figura 4.3, que mostra duas figuras geradas pelo site Robohash. Esse esquema é bem divertido e acaba permitindo bastante liberdade artística à implementação, mas permite ataques de similaridade por utilizar diretamente os bits do hash na escolha dos elementos.

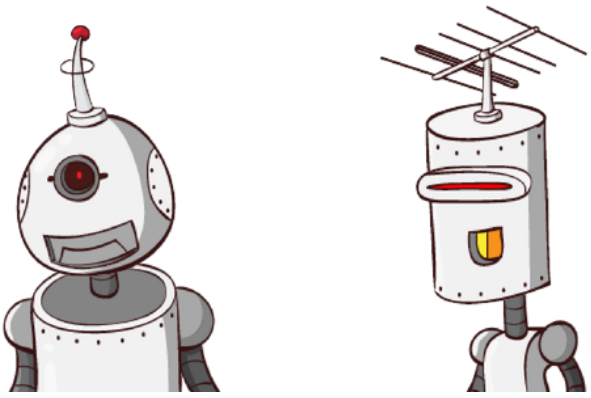


Figura 4.3: *Exemplos de robôs gerados pelo site Robohash. À esquerda, robô gerado a partir da frase "IME USP" (acessível em https://robohash.org/IME%20USP), à direita, robô gerado a partir da frase "IME-USP" (acessível em https://robohash.org/IME-USP)*

Ainda outra abordagem foi utilizada num esquema de visualização de hashes especificamente projetado para uso no pareamento de dispositivos: a visualização com T-Flags [14], que utiliza blocos de cores e um T para definir a orientação. Essa abordagem tem fácil cálculo da quantidade de informação transmitida e todas cores utilizadas foram escolhidas para minimizar erros devidos a deficiências visuais como daltonismo.

Figura 4.4: *Exemplo de comparação de T-Flags (imagem extraída de Lin et al. [14], artigo que apresentou o conceito) para pareamento de dispositivos.*

Aluns outros esquemas desenvolvidos incluem a arte do OpenSSH, Vash e Unicorn [23].

A visualização de arte ASCII do OpenSSH é uma das mais utilizadas e tem implementação rápida, porém gera estruturas desordenadas.

Quanto a Vash e Unicorn, ambas utilizam sequências derivadas dos hashes para definir mais bits transmitidos, evitando ataques de similaridade de bits parciais do hash; porém Unicorn não é tão efetivo no uso real e Vash precisa do uso de GPU para geração de imagens em tempo real, por ser similar a Random Art.

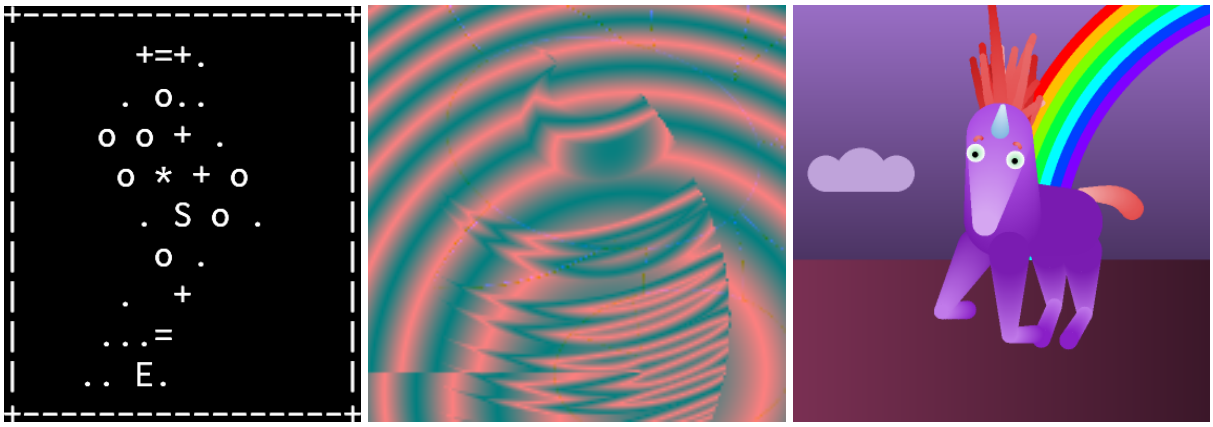


Figura 4.5: *Outras representações visuais de hashes (extraídas de Tan et al. [23]). À esquerda, OpenSSH; ao meio, Vash; à direita, Unicorn.*

O esquema de visualização de hashes no qual o trabalho atual se baseia, por outro lado, foi desenvolvido por Maina Olembo et al. [15] — o chamado protocolo SCoP, visível na figura 4.6. Ele mistura letras e ícones coloridos, preenchidos com listras verticais ou horizontais, por exemplo.

O protocolo SCoP teoricamente transmite 60 bits de informação, mas alguns dos elementos, como a direção das listras preenchendo os ícones podem ser difíceis de distinguir. Além disso, não foi feita a remoção de letras parecidas da fonte selecionada, aumentando a chance de confusão — vide a figura 4.6 que possui uma diferença razoavelmente difícil de perceber entre Z (subfigura esquerda superior) e 2 (subfigura esquerda inferior).

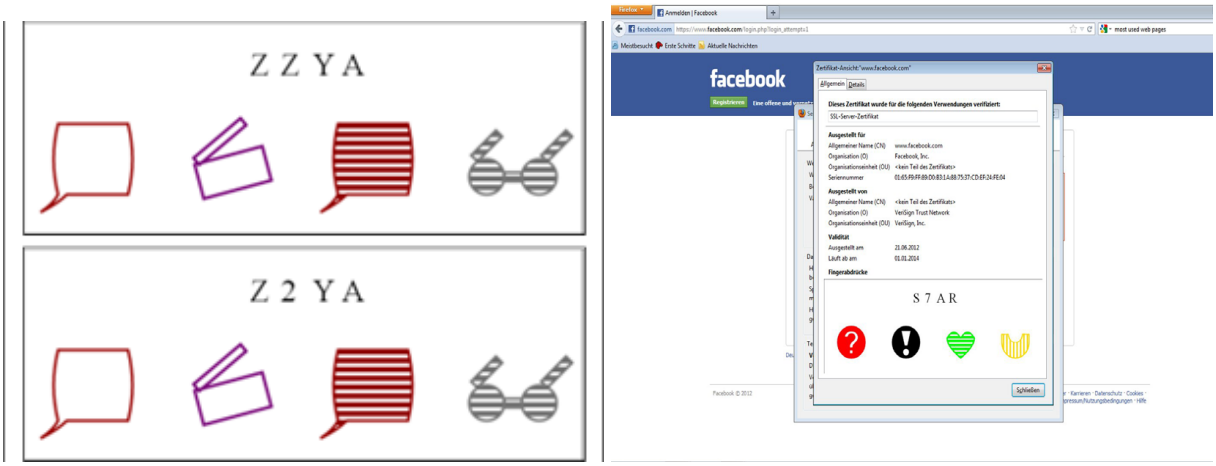


Figura 4.6: *Exemplo de SCoP, desenvolvido em Maina Olembo et al. [15]*

4.2 Animações de *hashes*

Para tornar as comparações de hashes mais animadas, foi desenvolvido um protocolo de geração de imagens com animações únicas derivadas a partir de hashes encadeados.

4.2.1 Random Art animado

A ideia de animar imagens geradas a partir de hashes já foi executada pelo menos uma vez, pelo usuário vshymanskyy do GitHub.

Uma implementação do Random Art animado pode ser encontrada em https://github.com/vshymanskyy/randomart, porém essa implementação depende da GPU; além disso a animação é basicamente incontrolável e não possui loops curtos, tornando mais difícil a comparação se a mesma estiver sendo exibida em dispositivos diferentes — seria necessário sincronizar os dispositivos para uma comparação efetiva, pois, como pode ser visto na figura 4.7, em momentos diferentes a mesma animação pode perder quase toda a similaridade.

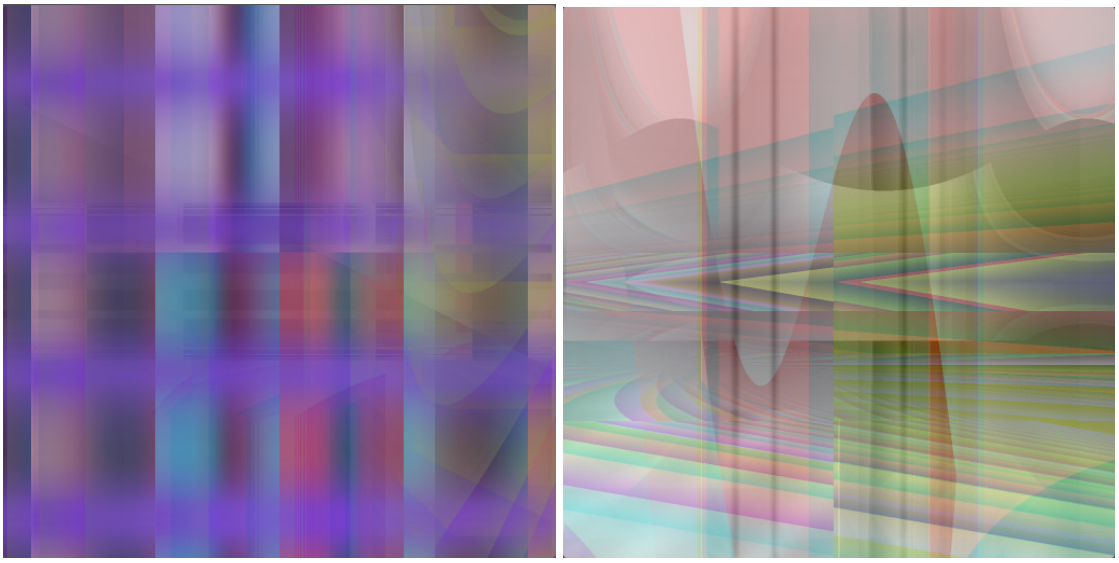


Figura 4.7: *Dois quadros do Random art animado gerado por https://github.com/vshymanskyy/randomart, usando como entrada a frase "IME USP" (imagem gerada em Dezembro de 2019)*

Segue no trecho abaixo a expressão que gerou a animação da qual foram extraídos os quadros da figura 4.7:

```
Mix(Level(-0.7366, Sum(Mod(Level(0.0588, Sum(Sin(1.0227, 1.9091, Const(
-0.2600, -0.7765, -0.3175)), Mix(Const(-0.1016, 0.5625, 0.3203), Const(
0.7344, 0.7578, 0.4219), BW(-0.4813))), Mod(Level(0.2069, Const(-0.2576
, 0.4684, 0.2232), Const(-0.3960, 0.5140, 0.2957), Const(0.1150, 0.1513
, -0.4389)), Sum(posX, Const(-0.1476, 0.0609, -0.8649))), Well(Level(-0
.3778, posX, BW(-0.7210), posX))), Closest(Mod(Not(posX), Mix(BW(-0.016
7), BW(-0.0157), posX)), Mul(Level(-0.7058, BW(0.0691), posY, posY), Te
nt(posY)), Not(Sin(1.2254, 1.8589, Const(-0.5863, 0.4930, -0.8951))))),
Mix(Tent(Mul(Closest(BW(0.5376), Const(0.7135, -0.9110, -0.1820), frmT)
, Level(0.6900, posX, Const(0.7578, 0.4375, -0.2031), posX))),Closest(M
ix(Mix(posY, frmT, Const(-0.9418, -0.0698, -0.5791)), Mix(BW(-0.5403),
frmT, Const(0.4766, -0.3359, -0.6094)), Sin(2.0874, 5.8667, posY)), Sin
(2.4723, 3.4490, Tent(Const(0.8292, -0.8331, -0.6601))), Mul(Tent(Const
(-0.5703, -0.5391, -0.4688)), Closest(Const(-0.1016, 0.5625, 0.3203), f
rmT, frmT))), Mul(Mod(Level(-0.7117, posX, posX, Const(-0.4800, 0.6332,
-0.2334)), Tent(posX)), Mul(Sum(Const(-0.0294, 0.0219, 0.3706), Const(-
0.8402, 0.9539, -0.2203)), Mix(frmT, Const(-0.3655, 0.7366, -0.3981), p
osX))))), Tent(Well(Not(Level(-0.1119, Level(0.1985, posY, posX, frmT),
Closest(Const(0.1364, -0.3844, 0.1227), Const(0.7344, 0.7578, 0.4219),
frmT), Well(Const(0.7578, 0.4375, -0.2031)))))), Sin(2.7871, 5.3920, Mo
d(Sum(Mix(Level(0.4525, posX, BW(0.7507), posY), Tent(posX), Well(posX)
), Sin(2.3219, 1.0451, Tent(posX))), Well(Tent(Closest(BW(0.6063), Cons
t(0.4766, -0.3359, -0.6094), BW(-0.6327)))))))), Mix(Well(Mix(Not(Sum(No
t(Const(0.4284, -0.1360, 0.7727)), Sum(BW(0.0228), Const(-0.5703, -0.53
91, -0.4688)))), Not(Mix(Well(posY), Mul(Const(0.7114, 0.7854, 0.4533),
posX), Sin(2.7816, 4.3291, Const(-0.1016, 0.5625, 0.3203))))), Not(Mod(M
od(Const(-0.7425, -0.9216, 0.8753), posY), Mod(Const(-0.5152, -0.1713,
```

```
0.7606), posX))))), Mul(Not(Well(Tent(Tent(frmT)))), Tent(Level(0.4224,
Level(0.3740, Mul(frmT, posX), Sum(posX, Const(0.7344, 0.7578, 0.4219))
, Tent(BW(0.0548))), Not(Level(0.2204, BW(0.7587), posY, frmT)), Not(Te
nt(Const(-0.1266, -0.8998, -0.4882)))))), Mod(Not(Not(Closest(Not(BW(-0
.4389)), Tent(frmT), Not(Const(0.7578, 0.4375, -0.2031))))), Mix(Closes
t(Well(Sin(2.6736, 4.7034, BW(-0.7841))), Closest(Tent(frmT), Mul(Const
(0.4766, -0.3359, -0.6094), Const(-0.5703, -0.5391, -0.4688)), Mod(Cons
t(-0.1016, 0.5625, 0.3203), posX)), Closest(Mul(posY, BW(0.4773)), Sin(
0.5667, 2.9181, frmT), Tent(posY))), Well(Sin(1.0503, 2.3797, Well(Cons
t(0.7344, 0.7578, 0.4219)))), Level(-0.0442, Mod(Tent(posY), Well(posY)
), Well(Mod(BW(-0.6968), posX)), Closest(Not(Const(0.7578, 0.4375, -0.2
031)), Tent(posX), Mod(posX, Const(0.3361, 0.9843, -0.1760)))))))), Mix(
Tent(Mix(Mod(Mix(Mul(frmT, Const(0.8919, -0.0965, -0.2778)), Closest(fr
mT, BW(0.3167), Const(-0.2742, -0.1723, 0.2771)), Level(0.4213, frmT, C
onst(0.4766, -0.3359, -0.6094), BW(-0.2305))), Mix(Sum(Const(-0.5703, -
0.5391, -0.4688), Const(-0.1016, 0.5625, 0.3203)), Mul(posY, BW(0.7826)
), Sin(0.6069, 4.6295, frmT))), Not(Tent(Sin(1.1627, 5.9632, posY))), T
ent(Not(Mul(Const(0.9150, -0.3670, 0.0650), frmT))))), Well(Well(Sum(Te
nt(Mul(Const(-0.3660, 0.1041, 0.7443), frmT)), Level(-0.7141, Not(Const
(-0.6623, -0.0711, -0.9320)), Level(0.6283, BW(0.0801), frmT, Const(0.9
740, 0.6503, 0.9409)), Mod(BW(-0.2646), frmT))))), Well(Mix(Mod(Tent(No
t(Const(0.7344, 0.7578, 0.4219))), Level(0.5596, Well(Const(0.7578, 0.4
375, -0.2031)), Tent(Const(0.4766, -0.3359, -0.6094)), Well(frmT))), Mi
x(Level(-0.3562, Tent(posX), Mul(Const(-0.5703, -0.5391, -0.4688), frmT
), Mod(posX, Const(-0.1016, 0.5625, 0.3203))), Well(Mod(posX, BW(0.4497
))), Mix(Mix(BW(-0.1459), frmT, Const(0.7344, 0.7578, 0.4219)), Sin(1.1
386, 4.9153, Const(0.7578, 0.4375, -0.2031)), Closest(posY, posX, frmT)
)), Mul(Sin(1.1792, 2.0699, Well(posX)), Level(0.1059, Not(posY), Sin(0
.0249, 5.3286, Const(-0.8006, -0.5086, -0.0936)), Sin(0.2734, 3.6236, f
rmT)))))))
```

4.2.2 Protocolo proposto

Esse trabalho expande o esquema desenvolvido por Maina Olembo et al. [15] adicionando informações no eixo temporal, permitindo menor poluição visual sem que ocorra a perda de muitos bits transmitidos.

Essencialmente, o objetivo é que os elementos animados permitam um aumento na densidade de informação transmitida visualmente sem que haja poluição visual por elementos excessivos — o que tornaria a comparação mais difícil.

Ocorreu uma tentativa inicial de desenvolvimento de animações a partir do Random Art [16], porém foram percebidas algumas características indesejáveis:

- O cálculo de quanta informação é transmitida é difícil — foi estimado em 24 bits para as imagens estáticas de Random Art [15])
- É relativamente fácil gerar imagens em que ocorre *overflow* ou *underflow*, tornando as imagens “planas” — com apenas uma cor ou pouca variação

- As imagens são muito custosas — seria obrigatório o uso de aceleração GPU para gerar animações em tempo real, o que limita bastante o uso simplesmente pela limitação de dispositivos
- É fácil gerar imagens ou animações feias
- É difícil controlar quais itens serão animados, tornando fácil gerar imagens com elementos demais
- Se for realizada a análise de qualidade da imagem proposta por Perrig e Song [16] o custo para geração de animações se torna ainda maior, pois não há garantias que outra tentativa (por exemplo, usando o 100º *hash* do *hash*) irá gerar imagens com as qualidades desejadas
- Uma animação com início (primeiro quadro) ou fim (último quadro) "bem comportados" não tem necessariamente quadros intermediários "bem comportados"
  - uma análise menos custosa (apenas primeiro e último quadro) não garante uma animação completamente "bem comportada"
  - uma análise de mais quadros pode garantir um animação "bem comportada", mas incorrem custos da análise e da re-geração da animação caso a análise identifique uma animação com características indesejáveis

Um dos piores pontos é a necessidade de uso de GPU, pois para que o uso seja bem generalizado para proteção efetiva contra ataques de substituição de certificados, é necessário que o protocolo permita implementações leves e com tecnologias bastante difundidas.

Pensando nisso, foi decidida a utilização de Javascript e SVG, que são suportados em todos navegadores há anos e podem ser utilizados nativamente por meio de WebKits. Além disso, o trabalho foi baseado no esquema SCoP [15], no qual haviam sido identificados menos áreas possivelmente problemáticas e mais fáceis de resolver, como:

- Alguns dos padrões são difíceis de serem comparados. Por exemplo, em imagens com uma única diferença entre padrão horizontal e vertical de um elemento a diferença é muito difícil de perceber, como visto na figura 4.8

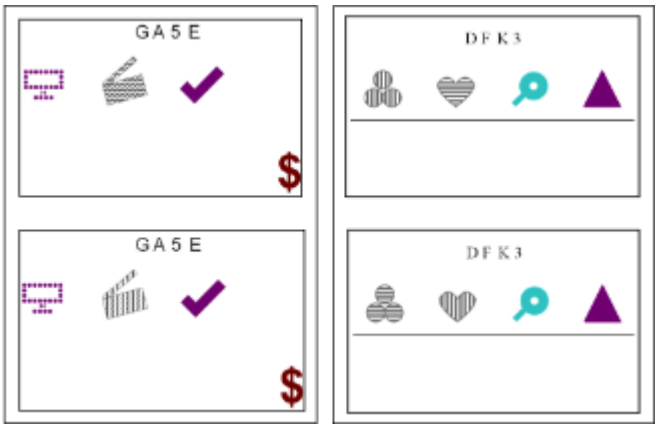


Figura 4.8: *As diferenças nas listras dos ícones do esquema SCoP podem ser difíceis de perceber*

- Algumas das letras e números são parecidos, dificultando a comparação rápida — por exemplo o 2 e o Z são bem parecidos na figura 4.6)
- As imagens tem pouco apelo estético

Baseado nisso, os principais elementos e ideias do protocolo desenvolvido nesse trabalho — daqui em diante chamado de `hashify` — foram definidos:

- Como no esquema SCoP [15] serão usados caracteres e ícones
- Como sugerido em Lin et al. [14] foi introduzido um elemento de assimetria na organização, para facilitar a orientação visual — foi decidido o uso de uma linha de quatro letras em cima de um quadrado de quatro ícones, permitindo rápida orientação visual
- Para evitar poluição visual, ao invés de adicionar mais elementos visuais, será adicionada informação no eixo temporal — por meio de animações como, por exemplo, mudança de cores, será possível adicionar mais bits de informação sem poluir demais a imagem
- adição de elementos animados, como
  - introdução dos elementos em ordens diferentes
  - os elementos podem ter posições iniciais diferentes das finais
  - os elementos pode ter uma animação circulando os mesmo durante a introdução
  - bordas dos elementos podem variar entre cores ou ter traços de tamanhos diferentes, e circular o elemento em direções diferentes
- elementos similares deverão ser removidos
- As animações devem ser geradas em tempo real
- As animações devem ser curtas, com apenas 2 segundos, para que seja possível comparar animações em dispositivos diferentes sem necessidade de sincronizar as mesmas

Durante o desenvolvimento, foi decidido que alguns caracteres similares seriam removidos para diminuir a chance de erros de comparação, baseado em exemplos comerciais, como os códigos de compartilhamento de níveis de Super Mario Maker 2. Sistemas de compartilhamento de níveis como o da figura 4.9 foram projetados para diminuir a confusão na hora de compartilhar níveis, desativando alguns caracteres para evitar erros no compartilhamento dos níveis, que possuem códigos no estilo “WQV-126-GNF”.

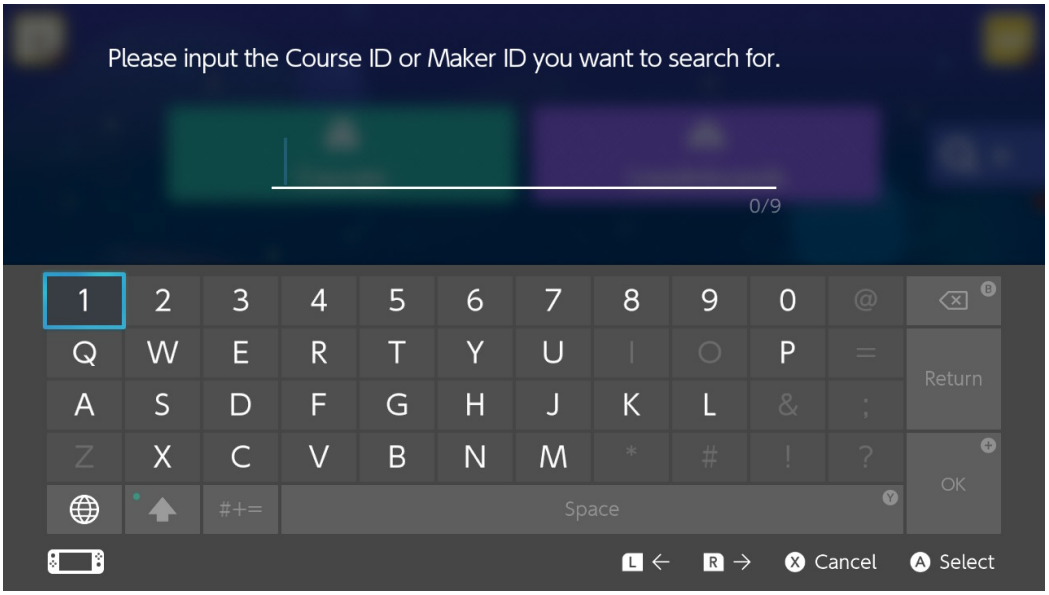


Figura 4.9: *Screenshot do teclado de códigos de compartilhamento de níveis e usuários do jogo Super Mario Maker 2, versão 1.1.*

Os 32 caracteres e ícones que são utilizados para gerar a animação são selecionados a partir de uma sequência de hashes de hashes gerada a partir do hash de entrada.

Os bits do hash original não são utilizados diretamente para evitar que existam bits 'principais' — o que evita, por exemplo, ataques que buscam gerar conteúdos com hashes que possuem apenas os bits determinados que geram os elementos mais visíveis ou chamativos. Além disso, ao utilizar sequências geradas por hashes de hashes é possível introduzir um *salt* no gerador da sequência, o que pode aumentar bastante a segurança e é possível estender a sequência conforme necessário.

O esquema `hashify` foi implementado numa biblioteca Javascript *opensource* chamada `hashify.js`, disponibilizada junto a esse trabalho. Um exemplo de uso pode ser encontrado junto às páginas do trabalho também.

Além disso foi desenvolvida uma extensão para o Firefox que permite visualizar a estampa animada do certificado HTTPS dos websites visitados. A extensão também está disponível junto a esse trabalho.

Uma das considerações da extensão desenvolvida foi permitir a adição de um salt para a geração das estampas animadas. Um *salt* é uma sequência de bytes ou caracteres que é introduzida numa sequência de hashes de hashes para tornar únicos os hashes seguintes, aumentando exponencialmente o trabalho de busca por uma sequência inicial que gere sequências parecidas — para cada salt possível existe o espaço de busca inicial. Ou seja, se um hash tem chance de colisão $\frac{1}{2^x}$ antes da inclusão de salts, o custo de um ataque geral de substituição de conteúdo é essencialmente idêntico ao custo de se gerar um conteúdo alterado com o hash idêntico ao original, $\frac{1}{2^x}$, e o custo para realizar um ataque individual fica custoso como a chance de gerar um ataque geral antes da inclusão de salts — o custo de gerar um conteúdo alterado que gere uma sequência de bits semelhantes apenas à sequência gerada com o salt.

4.2

Figura 4.10: *Os 32 ícones que podem ser usados pela biblioteca, criados pelo waqas17waqas através do site https://www.fiverr.com*

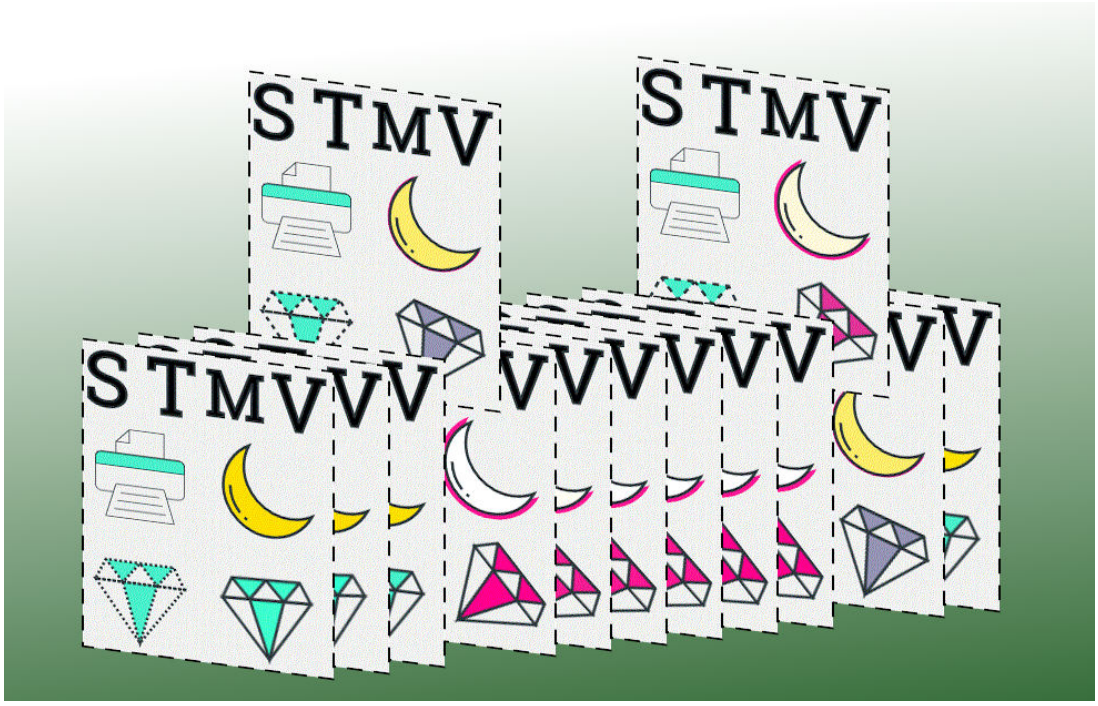


Figura 4.11: *Quadros de uma animação gerada pela biblioteca desenvolvida*

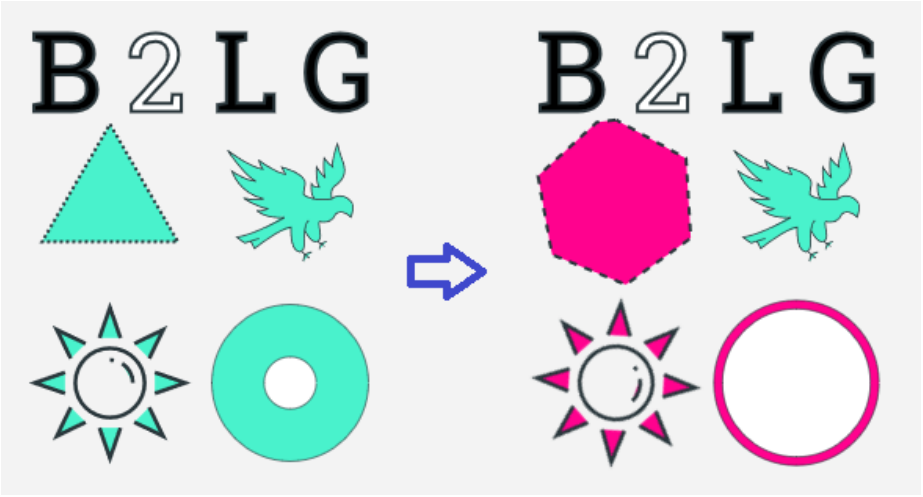


Figura 4.12: *Outro exemplo de animação gerada pela biblioteca, mostrando os quadros iniciais e finais*

4.3 Pesquisa

Entre os dias 25/10/2019 e 04/12/2019 foi aplicada uma pesquisa anônima para alguns usuários, com o intuito de verificar se algum dos elementos não tinha sua variação percebida.

Foi feita a pergunta “As duas estampas visuais abaixo são idênticas?” sobre 10 animações, cada uma variando um elemento visual como ícone ou espessura da linha. Além disso, duas das perguntas eram de controle, sendo a pergunta 2 totalmente idêntica e a pergunta 10 totalmente diferente. As perguntas tinham ordem randomizada, para evitar viés causado por fadiga nas últimas questões.

A partir dos resultados da pesquisa, especificamente a pergunta 7, foi percebido que a diferença na espessura da linha era pouco percebida, então a biblioteca desenvolvida foi atualizada para não utilizar essa opção.

Outro item interessante que foi percebido na pesquisa foi que mais da metade dos usuários não percebeu diferença na pergunta 6 — que tinha uma única diferença em uma das letras. Alguns comentários de usuários podem explicar o porquê: foi relatado que havia certa confusão se as letras deveriam ser comparadas também, isto é, faziam parte das estampas visuais. Isso destaca um erro da pesquisa, que mostra as animações em um plano de fundo isolado, quando foram desenvolvidas sempre em popups (como na extensão). Então se recomenda que as animações sempre tenham um quadro ou linha ao seu redor, para “ligar” as letras e ícones.

4.4 Probabilidade de colisão

O protocolo proposto extrai dos hashes a seguinte quantidade de bits:

- 5 bits para cada caractere escolhido pelo algoritmo (2^5 possibilidades)
- 5 bits para cada ícone escolhido pelo algoritmo (2^5 possibilidades)
- 1 bit para cada ícone devido à animação de cores/rotação
- 1 bit para cada ícone devido à movimentação do contorno

Como existem 4 caracteres e 4 ícones, obtemos um total de $5 \times 4 + (5 + 1 + 1) \times 4 = 20 + 28 = 48$ bits por animação.

Admitindo que a hipóteses do algoritmo SHA-256 — resistência pré-imagem, resistência de segunda pré-imagem e resistência de colisão — são verdadeiras, podemos assumir que a distribuição dos hashes é uniforme e não há correlação óbvia entre os mesmos. Com isso a chance de colisão considerando a geração de k certificados com hashs uniformemente distribuídos [18] pode ser calculada a partir da fórmula 4.1

$$probCol(N, k) = 1 - \frac{N-1}{N} \times \frac{N-2}{N} \times \cdots \times \frac{N-(k-2)}{N} \times \frac{N-(k-1)}{N} \approx 1 - e^{\frac{-k(k-1)}{2N}} \tag{4.1}$$

Resumidamente, a chance de colisão de um hash contra outro é $1 - \frac{N-1}{N}$, no caso de 48 bits é $\frac{1}{2^{48}}$.

Considerando uma impressora de má qualidade (que não permita distinguir espessura de linha ou tracejado muito bem) ainda assim são transmitidos pelo menos 40 bits de informação (apenas os caracteres e ícones distintos), fornecendo uma chance de colisão de $\frac{1}{2^{40}}$.

Capítulo 5

Conclusões

O trabalho produziu com sucesso uma biblioteca Javascript que utiliza 4 letras e 4 ícones SVG para gerar animações de 2 segundos que transmitem por volta de 48 bits de uma sequência derivada do hash original, além de um protótipo de extensão do Firefox que utiliza essa biblioteca para mostrar a estampa do certificado HTTPS utilizado em uma página web.

Além do uso na extensão de verificação de conexões `https`, estampas visuais tem outros usos possíveis, como estabelecimento de paridade de dispositivos [23][14] ou até mesmo confirmação visual de que o usuário digitou a senha corretamente [23] — confirmação essa sem vazar nenhuma informação que permita terceiros desvendar a senha, considerando que cada usuário teria um *salt* individual secreto configurado no navegador.

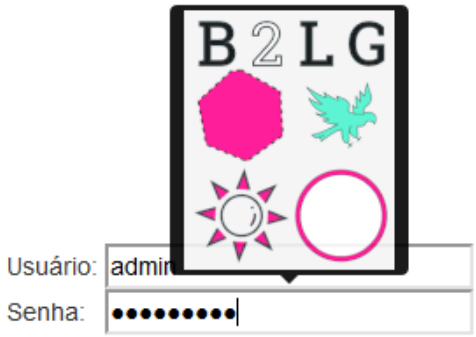


Figura 5.1: *Protótipo de verificação visual da senha digitada por meio de estampas visuais*

Um dos próximos passos no trabalho seria tentar estender ainda mais a quantidade de bits na animação sem adicionar poluição visual.

Algumas das melhorias sugeridas:

- As cores de início e fim podem ser sorteadas
- Adicionar mais cores dos ícones, levando em consideração problemas visuais como daltonismo.
- Adicionar *morfagem* de um ícone para outro — necessita de melhoria na biblioteca utilizada, permitiria mais 4 bits de informação por ícone. Por outro lado, esse item específico deve

ser considerado com cuidado e deve ser analisado se não adicionaria poluição visual.

- As letras podem mudar de fundo preto ou branco, adicionando mais 2 bits de informação por letra.

Por fim, após a implementação de todas adições possíveis será necessário realizar estudos similares aos implementados em Tan et al. [23], assim como um estudo de retenção de informação, similar ao realizado em Maina Olembo et al. [15].

Bibliografia

[1] Lucian Armasu. *Google To Remove China's Root Certificate Authority From Chrome Over Ties To Forged Certificate*. [Acessado em 18/jun/19, 13:38]. URL: https://www.tomshardware.com/news/google-bans-cnnic-root-ca,28873.html (ver p. 23).

[2] Mozilla Blog. *Distrust of Symantec TLS certificates | Mozilla Blog*. [Online; acessado em 18/jun/2019]. Mar. de 2018. URL: https://blog.mozilla.org/security/2018/03/12/distrust-symantec-tls-certificates/ (ver p. 23).

[3] The Black Chamber. *The Black Chamber - Letter Frequencies*. [Online; acessado em 15/set/2019]. URL: http://www.simonsingh.net/The_Black_Chamber/letterfrequencies.html (ver p. 6).

[4] Wikimedia Commons. *File:Modelo da nova carteira de identidade brasileira.jpg — Wikimedia Commons, the free media repository*. [Online; acessado 10 de Setembro de 2019]. 2019. URL: https://commons.wikimedia.org/w/index.php?title=File:Modelo_da_nova_carteira_de_identidade_brasileira.jpg&oldid=362620786 (ver p. 19).

[5] Wikimedia Commons. *File:Public-Key-Infrastructure.svg — Wikimedia Commons, the free media repository*. [Online; acessado em 25/set/2019]. 2018. URL: https://commons.wikimedia.org/w/index.php?title=File:Public-Key-Infrastructure.svg&oldid=302237450 (ver p. 15).

[6] Wikimedia Commons. *File:SHA-1.svg — Wikimedia Commons, the free media repository*. [Online; acessado em 15/set/2019]. 2017. URL: https://commons.wikimedia.org/w/index.php?title=File:SHA-1.svg&oldid=234989680 (ver p. 11).

[7] Wikimedia Commons. *File:SHA-2.svg — Wikimedia Commons, the free media repository*. [Online; acessado em 25/set/2019]. 2017. URL: https://commons.wikimedia.org/w/index.php?title=File:SHA-2.svg&oldid=240791937 (ver p. 13).

[8] D Eastlake e T. Hansen. *RFC6234 - US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. RFC 6234. [Online; acessado 15/set/2019]. RFC Editor, maio de 2011. URL: https://tools.ietf.org/html/rfc6234 (ver pp. 11, 12).

[9] Dan Goodin. *Microsoft to retire support for SHA1 certificates in the next 4 months*. [Online; acessado em 17/set/2019]. URL: https://arstechnica.com/information-technology/2016/05/microsoft-to-retire-support-for-sha1-certificates-in-the-next-4-months/ (ver p. 10).

[10] R. Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. [Online; acessado 10 de Setembro de 2019]. RFC Editor, set. de 2009. URL: https://tools.ietf.org/html/rfc5652 (ver pp. 8, 16, 17).

[11] J. C. Jones. *The end of SHA-1 on the Public Web*. Mozilla Security Blog. [Online; acessado em 17/set/2019]. URL: https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/ (ver p. 10).

[12] B Kalisky. *PKCS #7: Crytographic Message Syntax Version 1.5*. RFC 2315. [Online; acessado 10 de Setembro de 2019]. RFC Editor, mar. de 1998. URL: https://tools.ietf.org/html/rfc2315 (ver p. 8).

[13] Understanding the Limitations of HTTPS. *Ericlaw*. [Online; acessado em 04/12/2019]. URL: https://textslashplain.com/2018/02/14/understanding-the-limitations-of-https/ (ver p. 22).

[14] Yue-Hsun Lin et al. "SPATE: small-group PKI-less authenticated trust establishment". Em: *IEEE Transactions on Mobile Computing* 9.12 (2010), pp. 1666–1681 (ver pp. 26–28, 33, 39).

[15] M Maina Olembo et al. "Developing and testing SCoP–a visual hash scheme". Em: *Information Management & Computer Security* 22.4 (2014), pp. 382–392 (ver pp. 26, 28, 29, 31–33, 40).

[16] Adrian Perrig e Dawn Song. "Hash Visualization: a New Technique to improve Real-World Security". Em: 1999 (ver pp. 22, 26, 31, 32).

[17] Adrian Perrig e Dawn Xiaodong Song. "Hash Visualization : a New Technique to improve Real-World Security". Em: 2000 (ver p. 21).

[18] Jeff Preshing. *Hash Collision Probabilities*. Preshing on Programming. [Online; acessado em 18/nov/2019]. URL: https://preshing.com/20110504/hash-collision-probabilities/ (ver p. 37).

[19] Emil Protalinski. *Google will drop SHA-1 encryption from Chrome by January 1, 2017*. [Online; acessado em 17/set/2019]. URL: https://venturebeat.com/2015/12/18/google-will-drop-sha-1-encryption-from-chrome-by-january-1-2017/ (ver p. 10).

[20] Kelley Robinson. *What is Public Key Cryptography? - Twillio*. [Online; acessado em 10/set/2019]. URL: https://www.twilio.com/blog/what-is-public-key-cryptography (ver pp. 5, 8).

[21] Marc Stevens, Pierre Karpman e Thomas Peyrin. *The SHAppening: freestart collisions for SHA-1*. [Online; acessado em 17/set/2019]. URL: https://sites.google.com/site/itstheshappening/ (ver p. 10).

[22] Marc Stevens et al. *Announcing the first SHA1 collision*. Google Security Blog. [Online; acessado em 17/set/2019]. URL: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html (ver p. 10).

[23] Joshua Tan et al. "Can Unicorns Help Users Compare Crypto Key Fingerprints?" Em: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: ACM, 2017, pp. 3787–3798. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025733. URL: http://doi.acm.org/10.1145/3025453.3025733 (ver pp. 26, 28, 39, 40).

[24] *The MIT License*. [Online; acessado 10 de Setembro de 2019]. URL: https://opensource.org/licenses/MIT (ver p. 2).

[25] Wikipedia contributors. *Abstract Syntax Notation One — Wikipedia, The Free Encyclopedia*. [Online; accessado 05/nov/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Abstract_Syntax_Notation_One&oldid=919651925 (ver p. 16).

[26] Wikipedia contributors. *Caesar cipher — Wikipedia, The Free Encyclopedia*. [Online; acessado 10 de Setembro de 2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Caesar_cipher&oldid=913878820 (ver p. 6).

[27] Wikipedia contributors. *Cryptographic hash function — Wikipedia, The Free Encyclopedia*. [Online; acessado 15/set/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function&oldid=915451973 (ver p. 9).

[28] Wikipedia contributors. *Kazakhstan man-in-the-middle attack — Wikipedia, The Free Encyclopedia*. [Online; accessado 05/nov/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Kazakhstan_man-in-the-middle_attack&oldid=919799991 (ver pp. 1, 26).

[29] Wikipedia contributors. *ROT13 — Wikipedia, The Free Encyclopedia*. [Online; acessado em 10/set/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=ROT13&oldid=909020591 (ver p. 6).

[30] Wikipedia contributors. *SHA-1 — Wikipedia, The Free Encyclopedia*. [Online; acessado em 17/set/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=SHA-1&oldid=915103829 (ver p. 10).

[31] Wikipedia contributors. *SHA-2 — Wikipedia, The Free Encyclopedia*. [Online; acessado em 18/set/2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=915288790 (ver p. 11).