hashify: Uma Ferramenta para Visualização de Hashes com Animações*

Jorge Miguel Ribeiro, Daniel Macêdo Batista, José Coelho de Pina

[1]Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo (USP)

`jorgemiguelribeiro92@gmail.com, {batista,coelho}@ime.usp.br`

Abstract. *Comparing hashes is an essential operation in digital security, but tedious and error-prone when the hashes are in the form of hexadecimal strings. This paper introduces a new hash visualization software with animations. The software, called* `hashify`*, uses 4 characters and 4 SVG icons to generate 2-second animations that transmit around 48 bits of a sequence derived from the original hash. It was implemented as a JavaScript library and embedded in a Firefox extension prototype, which uses the library to display a stamp of the HTTPS certificate used on a web page. The probability of collision and the results of a user survey attest to the effectiveness of* `hashify`*.*

Resumo. *Comparar hashes é uma operação importante em segurança digital, porém tediosa e propensa a erros quando os hashes estão na forma de strings hexadecimais. Esse artigo apresenta um novo software de visualização de hashes com animações. O software, nomeado* `hashify`*, utiliza 4 caracteres e 4 ícones SVG para gerar animações de 2 segundos que transmitem por volta de 48 bits de uma sequência derivada do hash original. Ele foi implementado como uma biblioteca JavaScript e foi embutido em um protótipo de extensão do Firefox, que utiliza a biblioteca para mostrar uma estampa do certificado HTTPS utilizado em uma página web. A probabilidade de colisão e os resultados de uma pesquisa feita com usuários atestam a eficácia do* `hashify`*.*

1. Introdução

A criptografia permite codificação e decodificação para a troca segura de mensagens e é utilizada de forma quase invisível na Internet, garantindo a privacidade e possibilitando comunicações pessoais confidencias. Cotidianamente, a forma mais visível da criptografia para usuários não técnicos está presente 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 mas também uma garantia da autenticidade do *website* sendo acessado. Infelizmente, poucos usuários sabem exatamente a

*Código-fonte: `https://gitlab.com/jorgemiguelribeiro92/hashify` Documentação: `https://gitlab.com/jorgemiguelribeiro92/hashify/-/tree/master/doc` Vídeo: `https://youtu.be/raFkfRRXBM0` — Estas e todas as outras URLs referenciadas no artigo tiveram último acesso em 30 de Julho de 2020.

importância do protocolo HTTPS e como a verificação da validade de um certificado digital é um elemento crítico do protocolo. Não são raras as ocasiões em que, apesar das advertências emitidas pelo navegador, usuários acessam o conteúdo de um *website* sem garantia de confidencialidade e privacidade [Akhawe and Felt 2013, Felt et al. 2015].

Cada certificado possui um *hash* obtido por uma função como a SHA-256. Esse *hash* é a impressão digital (*fingerprint*) do certificado. Considerando que o usuário seja capaz de obter, e manter, o *hash* de forma segura, um modo de validar o certificado em acessos posteriores é comparando o seu *hash* com o *hash* previamente armazenado. Entretanto, a comparação das *strings* que representam os *hashes* é difícil de ser realizada visualmente, como pode ser observado no caso abaixo em que há apenas 1 caractere diferente entre os *hashes*:

```
e0d31d7599daebee65a15a639736d65dae6963d2
e0d31d7599daebee65a15a636736d65bae6963d2
```

De fato, seres humanos são lentos e suscetíveis a erros para verificar *strings* "sem sentido" [Perrig and Song 1999]. Após certo tempo comparando *hashes* sem a ocorrência de ataques, tendem a tomar atalhos nas comparações, por exemplo, comparando apenas o início e o final dos *hashes* [Tan et al. 2017]. Levando isso em conta, atacantes podem gerar certificados ou conteúdos com *hashes* que possuam apenas alguns bits similares, algo factível considerando o poder de processamento de computadores atuais.

Como seres humanos são efetivos ao comparar objetos geométricos e formas em geral, que não sejam caóticas, uma maneira de evitar a fatiga da comparação é transformar *hashes* em imagens e comparar essas imagens. Dessa forma o processo seria mais agradável e, esperançosamente, mais habitual e menos propenso a erros. Esse processo de transformação tem sido desenvolvido há alguns anos [Perrig and Song 1999, Lin et al. 2010, Davis 2011, Maina Olembo et al. 2014, Tan et al. 2017] com limitações.

Este artigo apresenta e descreve o software `hashify` que, embutido como uma extensão do navegador Firefox, gera uma estampa visual única a partir de um *hash*, no caso, do certificado de um *website*. Com essa estampa é possível verificar de forma rápida e visual a ocorrência de ataques de substituição de *website*, 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. Outras aplicações do software desenvolvido são na comparação de chaves públicas, pareamento seguro de dispositivos ou mesmo verificação visual de uma senha digitada à medida que ela vai sendo digitada. `hashify` avança o estado da arte em relação aos trabalhos relacionados ao gerar imagens animadas, o que evita poluição visual sem que ocorra a perda de muitos bits transmitidos. Ele está disponível no domínio público sob a licença MIT, permitindo assim a reutilização em trabalhos e aplicações futuras.

O restante deste artigo está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. Em seguida, a Seção 3 descreve a arquitetura e as principais funcionalidades do `hashify`. Detalhes da implementação, relatos de experimentos e uma discussão sobre a eficácia do `hashify` em termos de facilidade de comparação das imagens e de probabilidade de colisão estão na Seção 4. A Seção 5 descreve demonstrações das funcionalidades do `hashify`. Finalmente, conclusões e trabalhos futuros são apresentados na Seção 6.

2. Trabalhos Relacionados

Uma das primeiras tentativas bem sucedidas de transformar *hashes* em imagens ocorreu com o desenvolvimento do `Random Art` [Perrig and Song 1999]. O software gera árvores de expressões derivadas a partir dos bits de uma dada *string*. Essas árvores descrevem funções que são avaliadas a fim de gerar um valor RGB para cada pixel de uma imagem. A Figura 1 mostra exemplos de imagens geradas por duas implementações do `Random Art`.

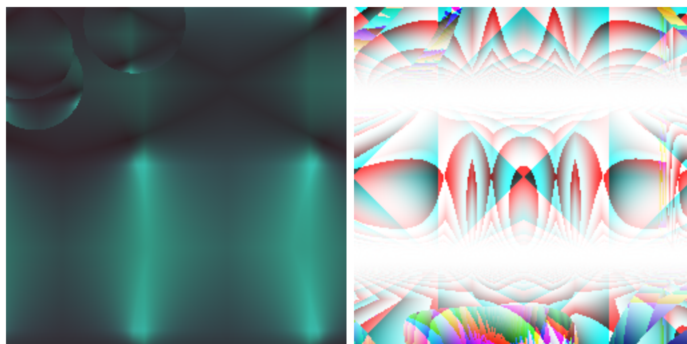

Figura 1. Imagens geradas com `Random Art` para o *hash* e0d31d7599daebee65a15a636736d65dae6963d2. A imagem da esquerda foi gerada pelo software em `http://www.random-art.org/online/`. A imagem da direita foi gerada por uma adaptação do software em `http://math.andrej.com/2010/04/21/random-art-in-python/`. Nesse segundo caso foi usada a composição de funções de *hash* para gerar números aleatórios.

Outra abordagem foi utilizada em um esquema de visualização de *hashes* para pareamento de dispositivos: a visualização com `T`-Flags [Lin et al. 2010], que utiliza blocos de cores e um caractere `T` para definir a orientação. A Figura 2 mostra exemplos de imagens geradas. Essa abordagem tem fácil cálculo da quantidade de informação transmitida e as cores utilizadas foram escolhidas para evitar erros devido a deficiências visuais.

Figura 2. Imagens geradas para comparação de `T`-Flags no pareamento de dispositivos (imagem extraída de [Lin et al. 2010]).

Outros esquemas tomam uma abordagem de gerar a imagem a partir de pedaços menores que se encaixam, como pode ser visto na Figura 3, que mostra um exemplo de imagem gerada pelo software `RoboHash` [Davis 2011]. Esse esquema permite bastante liberdade artística à implementação, mas pode sofrer ataques de similaridade por utilizar diretamente os bits do *hash* na escolha dos elementos.

Figura 3. Imagem gerada com `RoboHash` para o *hash* e0d31d7599daebee65a15a636736d65dae6963d2. A imagem foi gerada pelo software em `https://robohash.org/`.

Há ainda as imagens do `OpenSSH`, `Vash` e `Unicorn` [Tan et al. 2017]. A arte ASCII do `OpenSSH` é uma das mais utilizadas e tem implementação rápida, porém gera estruturas desordenadas. `Vash` e `Unicorn` 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 na prática [Tan et al. 2017] e `Vash` precisa do uso de GPU para geração de imagens em tempo real. A Figura 4 mostra exemplos desses esquemas.

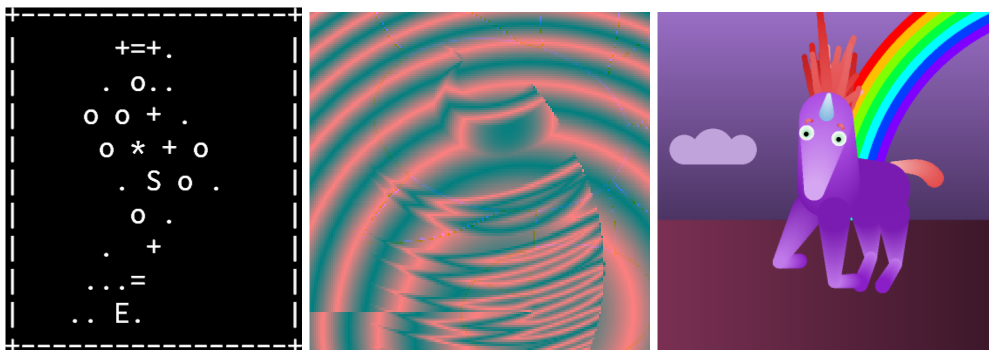

Figura 4. Outras representações visuais de *hashes*. À esquerda, `OpenSSH`; ao meio, `Vash`; à direita, `Unicorn`. (imagens extraídas de [Tan et al. 2017])

Outra proposta é o esquema SCoP apresentado em [Maina Olembo et al. 2014]. Uma imagem gerada por esse esquema pode ser vista na Figura 5. As imagens geradas pelo SCoP combinam caracteres e ícones coloridos, preenchidos com listras verticais ou horizontais. O esquema 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 considerada a remoção de caracteres similares considerando a fonte selecionada, dificultando a comparação em certas situações — vide a imagem à esquerda na Figura 5 que possui uma diferença razoavelmente difícil de perceber entre `Z` e `2`.

3. Arquitetura e Principais Funcionalidades do hashify

Levando os trabalhos relacionados em conta, o algoritmo de geração de imagens do `hashify` foi baseado no esquema SCoP [Maina Olembo et al. 2014] e possui as carac-

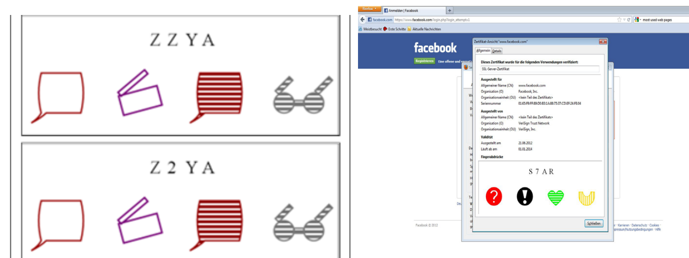

Figura 5. Exemplo do SCoP. À esquerda imagens geradas. À direita utilização para gerar estampa visual de um certificado digital (imagens extraídas de [Maina Olembo et al. 2014]).

terísticas descritas a seguir.

O funcionamento do `hashify` é baseado em um algoritmo de geração de imagens com animações únicas derivadas a partir da composição de funções de *hash* tendo como argumento inicial o *hash* original (*fingerprint*). A Figura 6 mostra uma sequência com 5 quadros da animação gerada pelo `hashify`.

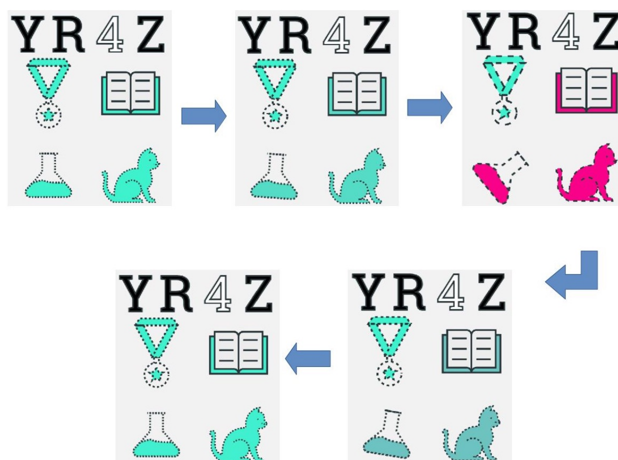

Figura 6. Quadros da animação gerada pelo `hashify` para o *hash* e0d31d7599daebee65a15a636736d65dae6963d2.

O `hashify` seleciona os elementos na imagem dentre 32 caracteres e 32 ícones. A Figura 7 exibe os ícones usados no `hashify`. Considerando a fonte utilizada, foram removidos caracteres similares e os seguintes caracteres podem ser utilizados: `B`, `C`, `D`, `E`, `F`, `G`, `H`, `J`, `K`, `L`, `M`, `N`, `O`, `P`, `R`, `S`, `T`, `U`, `V`, `W`, `X`, `Y`, `Z`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`.

Em relação a organização dos elementos na imagem, são colocados 4 caracteres alinhados na parte superior e 4 ícones formando um quadrado na parte inferior. Com essa assimetria na imagem deseja-se facilitar a orientação visual.

Para incrementar a quantidade de informação transmitida visualmente sem que com isso haja poluição visual devido a elementos excessivos, foi adicionada informação temporal por meio de animações. Essas animações podem rotacionar elementos ou variar as cores e traços de bordas. As animações são geradas em tempo real e tem a duração de apenas 2 segundos.

O SHA-256 é utilizado pelo `hashify` na composição de funções de *hash*. Com

Figura 7. 30 ícones que podem ser usados (criados pelo waqas17waqas através do site `https://www.fiverr.com`). Há também um círculo e um losango.

isso, os bits do *hash* original não são utilizados diretamente, evitando ataques que usem conteúdos com apenas determinados bits para gerar os elementos mais visíveis ou chamativos. Ainda na composição, é utilizado um *salt*. Um *salt* é uma sequência de bytes ou caracteres que é introduzida na composição das aplicações de funções de *hash* para tornar únicos os *hashes* seguintes, aumentando significativamente o trabalho de busca por um *hash* inicial que gere ao final imagens parecidas. Vale destacar que é importante que o *salt* não seja previsto por um atacante.

4. Implementação e Resultados

Para a implementação do `hashify` foi utilizado JavaScript e SVG (*Scalable Vector Graphics*), que são suportados nos principais navegadores. A implementação foi feita como uma biblioteca e utilizada para a criação de um protótipo de extensão para o Firefox que permite visualizar a estampa animada do certificado HTTPS dos *websites* visitados. O `hashify` considera que o servidor vai usar sempre o mesmo certificado dentro do período de validade do mesmo. Situações em que um servidor utilize mais de um certificado fogem do escopo da ferramenta.

O `hashify` se comportou corretamente quando instalado no Firefox versão 79.0 (64-bits) no sistema Operacional macOS Catalina versão 10.15.6 e no Firefox versão 68.10.0esr (64-bits) no sistema operacional Debian buster versão 10.4. O código está sob a licença MIT.

Foi realizada uma pesquisa anônima com 22 usuários, com o intuito de verificar se algum dos elementos utilizados na versão inicial do `hashify` não tinha sua variação percebida. Foi feita a pergunta "As duas estampas visuais abaixo são idênticas?" sobre 10 pares de estampas, com cada par variando um elemento visual, como ícone ou espessura da linha por exemplo. Duas das perguntas eram de controle, sendo uma associada a um par totalmente idêntico e outra a um par totalmente diferente. As perguntas foram apresentadas em ordem aleatória, para evitar viés causado por fatiga nas últimas perguntas.

A partir dos resultados da pesquisa foi percebido que a diferença na espessura da linha era pouco percebida, então a biblioteca foi atualizada para não utilizar essa opção. Também foi verificado que mais da metade dos usuários não percebeu diferença entre um par que diferia apenas por um caractere. Alguns comentários de usuários podem

explicar o motivo: foi relatado que havia certa confusão sobre se os caracteres deveriam ser comparados também, isto é, se faziam parte das estampas visuais junto aos ícones. Nesse sentido, na primeira utilização do `hashify` deveria ser exibido um breve tutorial explicando que tudo que está exibido na estampa deve ser considerado ao compará-la.

O `hashify` 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; e 1 bit para cada ícone devido à movimentação do contorno. Como as animações utilizam 4 caracteres e 4 ícones, há um total de $5 \times 4 + (5 + 1 + 1) \times 4 = 48$ bits por animação. Devido as hipóteses de que o SHA-256 possui resistência pré-imagem, resistência de segunda pré-imagem e resistência de colisão, podemos supor que a distribuição das imagens é uniforme e não há correlação óbvia entre as mesmas. Com isso, a probabilidade de colisão entre imagens considerando k valores de *hash* (*fingerprints* dos certificados) e N possíveis imagens pode ser calculada a partir da Equação

$$1 - \frac{N-1}{N} \times \frac{N-2}{N} \times \ldots \times \frac{N-(k-2)}{N} \times \frac{N-(k-1)}{N} \approx 1 - e^{\frac{-k(k-1)}{2N}}$$

Resumidamente, a probabilidade de colisão de um *hash* contra outro é $1 - \frac{N-1}{N}$. No caso de 48 bits, esse valor é $\frac{1}{2^{48}}$ [Preshing 2011].

5. Demonstrações

O `hashify` pode ser demonstrado facilmente com um computador ou *smartphone* acessando a demonstração disponível em `https://jorgemiguelribeiro92.gitlab.io/hashify/src/html/demo.html`. Qualquer *hash* pode ser utilizado na caixa de texto "Hash" e qualquer *string* pode ser utilizada na caixa de texto "Salt".

O protótipo de extensão para o Firefox também pode ser facilmente demonstrado a partir da instalação do mesmo no navegador em dois sistemas diferentes e comparando a estampa visual gerada nos dois sistemas ao acessar o mesmo conteúdo via HTTPS. O vídeo disponível em `https://youtu.be/raFkfRRXBM0` descreve esse processo.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou o `hashify`, uma biblioteca JavaScript que utiliza 4 caracteres e 4 ícones SVG para gerar animações de 2 segundos que transmitem por volta de 48 bits de uma sequência derivada de um *hash* original. A biblioteca foi incorporada em um protótipo de extensão do Firefox que exibe uma estampa visual do certificado HTTPS utilizado em *websites*. `hashify` está disponível em `https://gitlab.com/jorgemiguelribeiro92/hashify/`.

Como trabalhos futuros, pretende-se realizar uma pesquisa com mais usuários, aumentar a quantidade de bits usados na animação e utilizar o `hashify` para confirmação visual de que o usuário digitou uma senha corretamente, antes de enviá-la, sem vazar nenhuma informação que permita a terceiros desvendar essa senha. A Figura 8 exemplifica a ideia. Também pretende-se portar o protótipo da extensão para outros navegadores, como o Google Chrome e o Safari.

Figura 8. Uso do `hashify` para verificação da senha digitada

Agradecimentos

Esta pesquisa é parte do INCT da Internet do Futuro para Cidades Inteligentes, financiado por CNPq (proc. 465446/2014-0), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e FAPESP (procs. 14/50937-1 e 15/24485-9). Também é parte dos projetos FAPESP procs. 18/22979-2 e 18/23098-0.

Referências

[Akhawe and Felt 2013] Akhawe, D. and Felt, A. P. (2013). Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In *USENIX Security 13*, pages 257–272.

[Davis 2011] Davis, C. (2011). RoboHash. `https://robohash.org/`.

[Felt et al. 2015] Felt, A. P., Ainslie, A., Reeder, R. W., Consolvo, S., Thyagaraja, S., Bettes, A., Harris, H., and Grimes, J. (2015). Improving SSL Warnings: Comprehension and Adherence. In *ACM CHI'15*, pages 2893–2902.

[Lin et al. 2010] Lin, Y.-H., Studer, A., Chen, Y.-H., Hsiao, H.-C., Kuo, L.-H., McCune, J. M., Wang, K.-H., Krohn, M., Perrig, A., Yang, B.-Y., et al. (2010). SPATE: Small-Group PKI-less Authenticated Trust Establishment. *IEEE Transactions on Mobile Computing*, 9(12):1666–1681.

[Maina Olembo et al. 2014] Maina Olembo, M., Kilian, T., Stockhardt, S., Hülsing, A., and Volkamer, M. (2014). Developing and Testing SCoP–a Visual Hash Scheme. *Information Management & Computer Security*, 22(4):382–392.

[Perrig and Song 1999] Perrig, A. and Song, D. (1999). Hash Visualization: a New Technique to Improve Real-World Security. In *CryTEC '99*.

[Preshing 2011] Preshing, J. (2011). Hash collision probabilities. `https://preshing.com/20110504/hash-collision-probabilities/`.

[Tan et al. 2017] Tan, J., Bauer, L., Bonneau, J., Cranor, L. F., Thomas, J., and Ur, B. (2017). Can Unicorns Help Users Compare Crypto Key Fingerprints? In *ACM CHI'17*, pages 3787–3798.