Aperfeiçoamento da ferramenta sdhash para identificação de artefatos similares em investigações forenses

André Seiki Kameyama[1],
Vitor Hugo Galhardo Moia[1] **e Marco Aurelio Amaral Henriques**[1]

[1]Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
CEP 13083-852 – Campinas, SP, Brasil

a141858@dac.unicamp.br
{vhgmoia,marco}@dca.fee.unicamp.br

Abstract. *Forensic investigations have to deal with large amounts of data nowadays due to the development of technology, becoming impractical to manually analyze each case. For this reason, methods that can reduce the time needed in an investigation and yet be effective in finding evidence are necessary. In this work, we show how to improve the performance and precision of one of the most consolidated approximate matching tools in the area, the sdhash. Our results show that the modified tool is capable of identify similarities between different artifacts in an easier and faster way.*

Resumo. *Atualmente as investigações forenses têm que lidar com grandes quantidades de dados devido ao avanço da tecnologia, tornando-se inpraticável a análise manual de cada caso. Por esta razão, métodos capazes de reduzir o tempo em uma investigação e ainda serem eficientes em encontrar evidências são necessários. Neste trabalho, nós mostramos como melhorar o desempenho e a precisão de uma das ferramentas de pareamento aproximado mais consolidadas na área, o sdhash. Nossos resultados mostram que a ferramenta modificada é capaz de identificar similaridades entre diferentes artefatos com maior facilidade e rapidez.*

1. Introdução

Em investigações forenses nos dias de hoje, além da diversidade de equipamentos confiscados, há um grande volume de dados que necessitam ser analisados, o que torna inviável uma análise manual dos mesmos. Dessa forma, são essenciais técnicas que permitam realizar a perícia de forma mais rápida e automática, e que ao mesmo tempo sejam confiáveis e escaláveis. Uma das alternativas encontradas é por meio do método conhecido como *Known File Filtering* (KFF), ou filtragem por arquivos conhecidos, utilizado tanto para filtrar arquivos confiáveis, quanto arquivos suspeitos ou ilegais. Nesta abordagem, a equipe de investigação forense utiliza uma base de dados com arquivos conhecidos que poderão ser ignorados (*Whitelist*), como arquivos do sistema operacional, arquivos de programas conhecidos, dentre outros. Ela

pode utilizar também uma base de dados com arquivos ilegais ou suspeitos (*Blacklist*), que contempla malwares, arquivos com conteúdos proibidos, dentre outros.

Uma das possíveis ferramentas para realizar o KFF é aquela baseada em funções criptográfica de hash, como o MD5, SHA-1, SHA-2, por exemplo. Estas funções são caracterizadas por conter os seguintes atributos: são algoritmos que mapeiam dados de comprimento variável para dados de comprimento fixo, também conhecidos como resumo do arquivo original. Os resumos são calculados de forma rápida e fácil, e têm as seguintes características: é impossível gerar com certeza absoluta a mensagem original a partir de seu resumo, é extremamente difícil encontrar duas mensagens que tenham o mesmo resumo e é ainda mais difícil encontrar uma mensagem que tenha o mesmo resumo que uma outra já existente. Além disso, mínimas modificações na mensagem de entrada causam grandes mudanças no resumo (efeito avalanche). Logo, é possível, de forma fácil e escalável, fazer a comparação de resumos de conteúdos dos dispositivos confiscados com os resumos das bases de dados usando funções de hash.

Contudo, existe uma grande limitação das funções de hash que é justamente para casos onde um objeto de interesse tem poucos ou só um byte alterado, o que faz com que seu hash seja completamente diferente da versão sem modificação, dificultando investigações forenses. Vale ressaltar que esta modificação pode ser feita tanto de forma espontânea, através de atualizações de softwares, como de forma intencional, onde um atacante deseja ludibriar uma investigação. Para contornar esta limitação, foram desenvolvidas as funções de Pareamento Aproximado (Approximate Matching), cuja finalidade é encontrar semelhanças entre dois objetos por meio de representações compactas, semelhante aos resumos gerados pelas funções hash. Assim, pequenas alterações em um dado não comprometerão a identificação do mesmo pelo investigador. Entretanto, se por um lado as funções hash necessitam apenas de uma comparação simples de strings, as funções de hash de similaridade requerem funções especiais de comparação, além de ter um processo de geração de resumos muito mais custoso.

Dentre as diversas ferramentas que implementam os conceitos das funções de Pareamento Aproximado, podemos citar o sdhash [Roussev, V. 2010] por se destacar das demais devido às suas atraentes características e popularidade na área. Contudo, esta ferramenta apresenta altos custos para geração e comparação de resumos e outras limitações que este trabalho objetiva minimizar. Em particular, investigaremos o alto custo do processo de geração de resumos e a precisão do sdhash. Nossos resultados preliminares mostram que é possível reduzir seu custo sem prejudicar a acurácia, o que poderá auxiliar na execução de investigações forenses de forma mais rápida e eficiente.

2. Trabalhos relacionados

Muitas pesquisas foram realizadas nesta área até serem propostos métodos que implementam as funções de Pareamento Aproximado. Estes, por sua vez, são capazes de detectar similaridades (no nível de bytes) através de compactas representações criadas para os objetos e dizer o quão parecidos eles são. O foco deste trabalho está em trabalhar com a similaridade no nível de bytes, por ser mais eficiente e independente de formato, mas também existem trabalhos que exploram o nível sintático e semântico.

Os dois algoritmos mais populares e utilizados da área são o ssdeep [Kornblum, J. 2006] e o sdhash. A ferramenta ssdeep[1] foi apresentada como prova de conceito de Context Triggered Piecewise Hashing (CTPH) e ganhou grande aceitação. A idéia básica desta ferramenta é simples: dividir uma entrada em partes de tamanho variável, calcular o hash de cada uma de forma independente e concatenar tais hashes de forma a criar um resumo de similaridade no final (também chamado de *fingerprint* ou *digest*). É importante notar que o ssdeep garante que um certo objeto tenha um número de partes entre 32 e 64. Para dividir uma entrada em partes, o algoritmo identifica pontos de gatilho usando uma função especial de hash de rolamento. Cada parte é então passada a uma função de hash (FNV) da qual são selecionados apenas os 6 bits menos significativos e usa uma tabela Base64 para selecionar e utilizar um caractere para representação. No final, os caracteres são concatenados para formar o resumo final do arquivo. Assim, dois arquivos são semelhantes se possuem partes em comum.

Outra grande contribuição veio com o trabalho de Frank Breitinger, F. e Roussev, V. (2014) em realizar testes automatizados e precisos em cima de dados reais para estudar a performance prática dos métodos citados anteriormente. Nesses testes, foi estabelecida uma métrica própria chamada Approximate Longest Common String (aLCS[2]). A ideia principal não é comparar os objetos byte por byte, mas sim em pacotes de tamanho variável. Para pegar um pacote é feita uma derivação da abordagem padrão para dados de impressões digitais por polinômios aleatórios [Rabin, M. O. 1981].

Frank Breitinger, F. e Roussev, V. mostraram que o sdhash se sobressai em relação ao ssdeep nos critérios de precisão e robustez na detecção de similaridade entre objetos de tamanhos diferentes, justificando a escolha do mesmo como base deste trabalho. Outros algoritmos foram propostos na área de pareamento aproximado [Harichandran, V. S., Breitinger, F., & Baggili, I. 2016] [Lee, A., & Atkison, T. 2017], contudo, não tiveram o mesmo impacto de seus antecessores e serão estudos com mais detalhes em trabalhos futuros.

3. SDHASH

O algoritmo sdhash[3], Similarity Digest Hash, desenvolvido em 2010 por *Vassil Roussev,* veio quatro anos depois do ssdeep com o intuito de melhorá-lo em alguns pontos. Ao invés de dividir uma entrada em partes, ele tem por objetivo selecionar características (*features*) únicas para representar um objeto de entrada e produzir seu hash de similaridade, que denominaremos resumo. Dois resumos podem ser comparados para quantificar o quão similares eles são. Além da detecção da similaridade no contexto de semelhança entre dois objetos, o sdhash também é capaz de detectar objetos contidos um no outro (*Containment Detection*), como uma imagem inserida dentro de um documento, por exemplo.

Existem algumas limitações nesta ferramenta, como por exemplo, a incapacidade de trabalhar com arquivos menores do que 512 bytes. Além disso, existem alguns problemas de precisão, como a insensibilidade a alterações nas partes inicial e

[1] http://ssdeep.sourceforge.net/#download
[2] http://www.dasec.h-da.de/staff/breitinger-frank/
[3] http://roussev.net/sdhash/sdhash.html.

final de um objeto: um atacante poderia alterar um objeto de forma maliciosa e a ferramenta não seria capaz de detectar tal modificação. O custoso processo de geração e comparação de resumos em relação a outras ferramentas da área também é um impedimento para ampla adoção do sdhash pela comunidade forense.

As próximas subseções explicarão o funcionamento detalhado dessa ferramenta nas suas etapas de extração de *features*, geração e comparação de resumos.

3.1 Geração dos resumos com a ferramenta sdhash

Para gerar um resumo, o sdhash precisa, a partir de um objeto de entrada, extrair e selecionar as *features* do mesmo. Na etapa inicial, o algoritmo percorre os bytes do arquivo de entrada e inicia o processo de geração das *features*, onde uma janela deslizante de tamanho fixo *W* (64 bytes) seleciona os *W* bytes do arquivo e calcula a entropia desta sequência de bytes levando em consideração seu tamanho e a probabilidade empírica de encontrar determinado byte na tabela ASCII. Uma normalização é feita para que este valor esteja entre 0 e 1000. Em seguida, o valor calculado, que é a representação da *feature*, é armazenado e a janela se desloca em um byte. O processo se repete até o final do arquivo.

No processo de filtragem, são descartadas as *features* com entropia inferior a 100 ou superior a 990. Isso é necessário para reduzir o número de falsos-positivos, casos nos quais o algoritmo diz que objetos completamente diferentes aparentam ser similares. A razão do limite inferior é justificada pelo fato de as *features* com baixa entropia possuírem pouca informação. Um clássico exemplo disso seria uma seção inteira apenas com zeros, algo relativamente comum em arquivos totalmente distintos. Já as *features* com entropia quase máxima são aquelas com tamanha quantidade de informação que deixam de representar o objeto, pois trata-se de uma configuração caótica e difícil de interpretar.

Tendo em mãos as *features* já filtradas com suas respectivas entropias, é realizada a etapa de seleção, que ocorre da seguinte forma: outra janela deslizante de tamanho fixo *B* (igual a 64, por default), iniciada na primeira *feature* extraída, percorrerá todo o vetor de *features* representadas por suas entropias, denominado *Rprec*. Durante cada iteração, a *feature* que possuir a entropia de menor valor e que se posicionar mais à esquerda da janela será eleita a vencedora daquela iteração. A contabilização do número de vitórias de cada *feature* fica armazenada no vetor *Rpop*. Em seguida a janela é deslocada uma posição para a direita e o processo se repete até que a janela alcance a última *feature*. Ao fim, serão selecionadas apenas as *features* que venceram um número determinado *T* de vezes (*threshold*) e essas serão escolhidas para representar o objeto. Esta etapa é muito importante, pois será um dos pontos de melhoria abordados. A figura a seguir ilustra este processo, utilizando *B*=*8* e *T*=4. Neste exemplo, apenas duas *features* serão selecionadas.

A última etapa consiste da geração da representação do resumo. Para isto, é calculado o hash das *features* escolhidas no passo anterior por meio da função SHA-1, que retorna 160 bits, e os resultados são divididos em cinco sub-hashes de 32 bits. De cada um destes são retirados os 11 primeiros bits e inseridos em Filtros de *Bloom*.

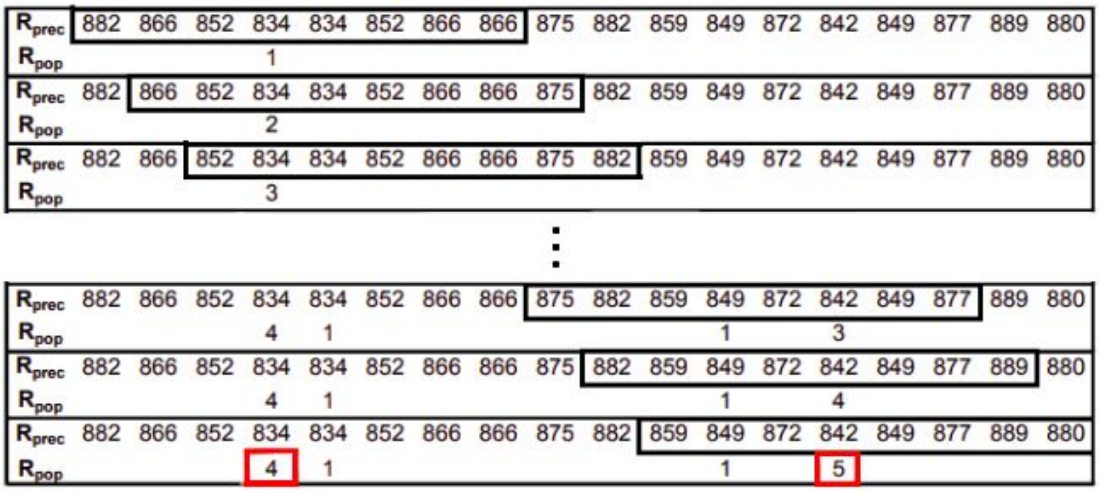

Figura 1. Processo de seleção de *features* baseado nas entropias normalizadas e filtradas. Adaptado de *Roussev, V. (2010).*

No projeto do sdhash, foi definido que cada filtro tem um tamanho fixo de 256 KBytes e um número máximo de elementos por filtro (160, por default). Contudo, novos filtros são criados se necessário. Desta forma, o resumo de um objeto é uma sequência de Filtros de *Bloom*, cada um representando em média 10 KBytes dos dados originais.

3.2. Comparação de dois resumos com o sdhash

Para estabelecer o nível de similaridade entre dois objetos, o sdhash compara seus resumos, compostos de sequências de Filtros de *Bloom*. O primeiro filtro do primeiro objeto é comparado com todos os filtros do segundo objeto. É selecionada a maior pontuação de similaridade e o processo se repete com todos os outros filtros do primeiro objeto, que novamente são comparados com todos os filtros do segundo objeto. Por fim, uma média é calculada para chegar a um valor entre 0 e 100. A interpretação desse valor é comentada do manual do sdhash[4].

4. Proposta

Com o intuito de melhorar o processo de identificação de objetos similares utilizando o sdhash, este trabalho implementou uma nova versão com algumas modificações que visam aprimorar tanto sua eficiência como sua precisão. Nas próximas subseções, discutimos duas deficiências do sdhash que este trabalho buscou melhorar.

4.1. Janela circular na seleção de *features*

Um dos problemas da atual versão do sdhash é a seleção de *features*. Para uma *feature* ser selecionada, ela deve possuir o menor valor de entropia em relação a outras que compõem uma janela deslizante. Contudo, na forma como a janela atual está implementada, as primeiras e as últimas *features* são comparadas um número menor de

[4] http://roussev.net/sdhash/tutorial/sdhash-tutorial.pdf

vezes em relação às demais. Por exemplo, a primeira *feature* da janela deslizante tem apenas uma chance de ser a vencedora, pois na próxima iteração, ela sairá da janela e dará lugar a outra *feature*. Dependendo do *threshold* (*T*) escolhido, algumas *features* nunca serão escolhidas para representar o objeto. Portanto, alterações tanto no início como no fim de um objeto, podem não ser espelhadas no resumo.

A solução proposta é utilizar uma janela circular na etapa de seleção das *features* (Figura 1). Assim, o problema descrito acima é contornado fazendo com que todas as *features* sejam comparadas um mesmo número de vezes na etapa de seleção.

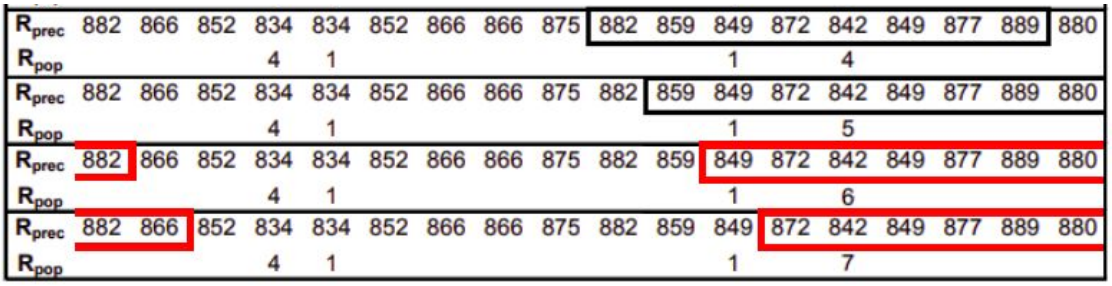

Figura 2. Exemplo da seleção de *features* utilizando a janela circular. Adaptado de *Roussev, V. (2010)*.

4.2. Desempenho com outras funções de hash

Outra proposta de melhoria é em relação a eficiência do sdhash. A função hash SHA-1 é utilizada em cada *feature* selecionada para inserção da mesma em um Filtro de *Bloom*. Contudo, a função SHA-1 é computacionalmente mais custosa que outras funções e neste contexto, nossa hipótese é que a utilização de funções computacionalmente mais baratas, como o MD5 e FNV-64 [G. Fowler, L. Noll, P. Vo], tornaria o processo de geração de resumo mais eficiente sem perda de precisão.

Mesmo que as funções de hash escolhidas sejam mais propensas a colisões do que a função SHA-1, isso não é um problema. A geração de um resumo requer o cálculo de vários hashes, um por feature, de forma que a colisão teria que acontecer para um grande número de features de modo a causar um resultado expressivo.

5. Configuração dos experimentos

Com o intuito de validar as alterações sugeridas, realizamos testes utilizando uma base de arquivos gerada de forma aleatória com tamanho predeterminado. Nesta base controlamos exatamente a porcentagem de alteração de um arquivo em relação à sua versão original, tendo como base critérios sugeridos em outros trabalhos [Breitinger, F., Stivaktakis, G., & Roussev, V. 2014]. Desta forma, sabemos realmente o quão dois arquivos são similares e podemos validar os resultados do sdhash. Cada versão similar de um arquivo foi criada utilizando as funções de manipulação *Fragment Detection* (um arquivo é um fragmento do outro, equivalente a *Containment Detection*), *Single Common Block Correlation* (os dois arquivos têm o mesmo tamanho, mas apenas uma parte igual) e *Alignment Robustness* (um arquivo é a cópia do outro, mas prefixado com uma sequência de bytes aleatórios).

Também utilizamos a base T5 [Roussev, V 2011], que contém dados reais e conta com diferentes extensões de arquivos: PDF, XLS, DOC, TXT, JPG, GIF, PPT e

HTML. Foram selecionados 80 MB de arquivos desta base. A implementação do sdhash v3.4[5] foi utilizada como base de comparação.

6. Resultados

Tendo implementado a janela circular, observamos que o número de *features* selecionadas aumentou, o que era esperado uma vez que temos mais iterações. Consequentemente, o tamanho do resumo também aumentou; além disso o processo como um todo se tornou mais justo (já que todas as partes têm chances iguais) e mais preciso, como detalhado mais abaixo.

Tabela 1. Comparação do número de *features* selecionadas para objetos de vários tamanhos entre o sdhash original e o sdhash com a janela circular.

Tamanho do objeto	Número de *features* selecionadas		
	Sdhash original	Sdhash com janela circular	Variação
1024 KiB	17597	17921	+1,84%
512 KiB	8687	8904	+2,50%
256 KiB	4370	4443	+1,67%
64 KiB	1082	1106	+2,22%
16 KiB	273	274	+0,37%
4 KiB	71	73	+2,82%
1 KiB	18	19	+5,56%

Para verificar a precisão da janela circular, foram realizados testes com a versão do sdhash original e a modificada, utilizando a base de dados controlada. As tabelas a seguir mostram que a janela circular, embora não tendo feito com que as comparações evoluíssem dentro dos níveis de similaridade, como explicado na seção *3.2*, trouxe um aumento de confiabilidade, principalmente para os arquivos maiores.

Tabela 2. Resultado da comparação entre o sdhash original e a versão com janela circular. Objetos gerados pelo modo *Fragment Detection.*

Tamanho dos objetos	Bytes alterados em relação ao objeto original					
	2%		5%		10%	
	Original	Janela Circular	Original	Janela Circular	Original	Janela Circular
1024 KiB	75	71	51	62	57	70
512 KiB	45	46	99	98	79	82
256 KiB	96	96	80	77	33	36
64 KiB	92	100	100	100	100	100
16 KiB	0	0	0	0	94	100
4 KiB	0	0	0	0	0	0
1 KiB	0	0	0	0	0	0

[5] https://github.com/sdhash

Tabela 3. Resultado da comparação entre o sdhash original e a versão com janela circular. Objetos gerados pelo modo *Single Common Block Correlation.*

Tamanho dos objetos	Bytes alterados em relação ao objeto original					
	10%		30%		50%	
	Original	Janela Circular	Original	Janela Circular	Original	Janela Circular
1024 KiB	4	5	22	23	39	39
512 KiB	3	2	24	27	32	44
256 KiB	4	5	25	26	30	32
64 KiB	1	2	11	10	20	17
16 KiB	0	0	0	0	31	32
4 KiB	0	0	0	0	22	24
1 KiB	0	0	0	0	25	26

Tabela 4. Resultado da comparação entre o sdhash original e a versão com janela circular. Objetos gerados pelo modo *Alignment Robustness.*

Tamanho dos objetos	Bytes alterados em relação ao objeto original					
	5%		10%		20%	
	Original	Janela Circular	Original	Janela Circular	Original	Janela Circular
1024 KiB	27	28	90	95	57	48
512 KiB	55	68	25	46	87	90
256 KiB	46	34	67	75	31	36
64 KiB	56	49	62	57	51	47
16 KiB	92	93	86	86	59	61
4 KiB	100	100	100	100	100	100
1 KiB	92	100	92	100	92	100

Para verificar o impacto da troca da função de hash, foram realizados outros experimentos em que variamos os seguintes parâmetros: função de hash, número de bits selecionados pela função de hash e tipo de arquivo. Nos testes foram medidos o tempo de execução e a precisão da ferramenta. O indicador de precisão utilizado foi o desvio padrão da distribuição das *features* em um histograma, visto que quanto menor este desvio, maior é a certeza de que a *feature* é representativa do arquivo. Cada *feature* é representada por um número *b* de bits extraídos do hash. O sdhash utiliza 55 dos 160 bits do SHA-1, divididos em 5 partes de 11 bits, para setar um Filtro de *Bloom*. Quanto maior o número de bits, menor são as chances de colisões. Nossos experimentos buscam verificar se outras funções de hash possuem a mesma precisão do SHA-1 para este contexto. Desta forma, variamos o número de bits entre 11 e 19 para representar as *features* a fim de verificar a diferença entre as funções de hash. As Figs. 3 e 4 mostram a relação do tempo de execução e do desvio padrão com o número de bits selecionados.

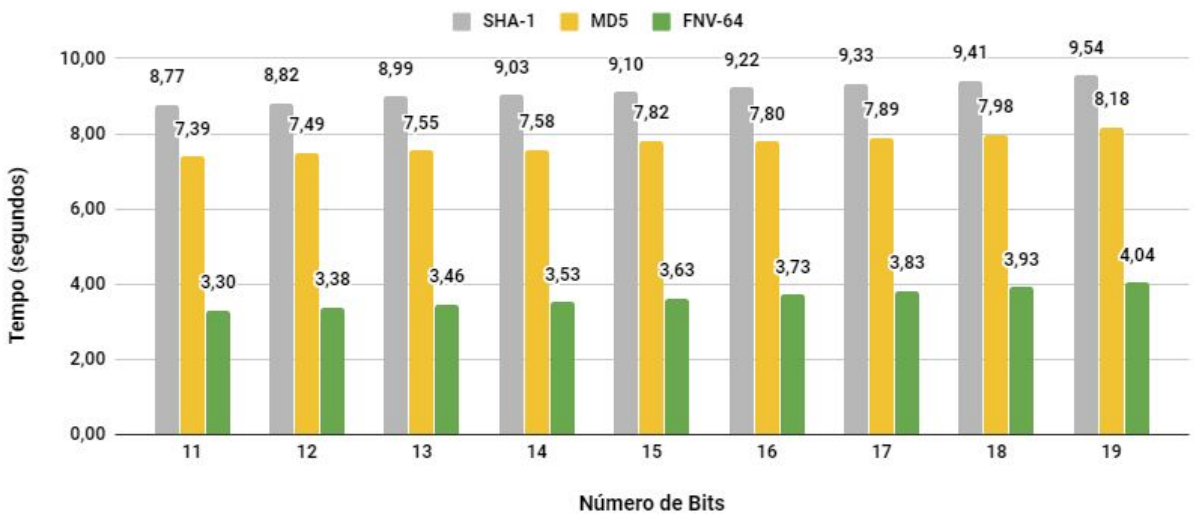

Figura 3. Tempo de execução para diferentes números de bits e funções hash utilizando a base com dados reais.

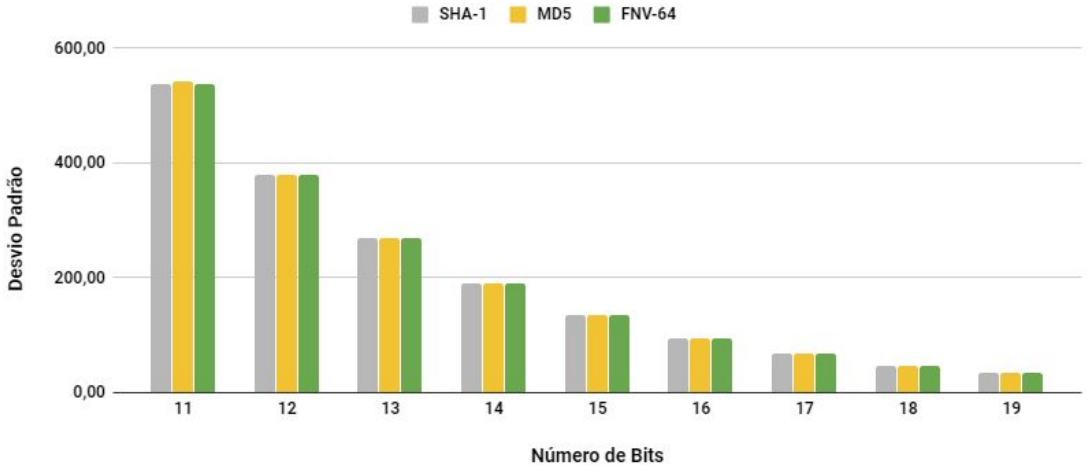

Figura 4. Desvio padrão para diferentes números de bits e funções hash utilizando a base com dados reais.

É possível observar que a mudança da função do SHA-1 para o FNV trouxe uma grande economia em relação ao tempo de execução, enquanto o desvio padrão permaneceu o mesmo. Logo, podemos adotar a função FNV sem receio de perda de precisão. A variação de bits, nos permite concluir que, para este intervalo, o comportamento das funções de hash foi praticamente o mesmo.

Outros testes foram realizados com cada um dos diferentes tipos de arquivo (PDF, XLS, DOC, TXT, JPG, GIF, PPT e HTML) para validar se a escolha da função hash afeta a distribuição de *features* para diferentes tipos de dados. Porém, não foram constatadas variações significativas em nenhum caso.

7. Conclusão

Neste trabalho foram apresentadas formas para o aprimoramento de uma das funções mais utilizadas em investigações forenses digitais para a busca de objetos

similares, o sdhash. Foi mostrado que esta ferramenta possui imprecisões na geração de seus resumos. A solução adotada neste trabalho foi a implementação de uma estrutura de janela circular na etapa de seleção de *features*, o que tornou o processo mais justo e aumentou a sensibilidade do mesmo em relação a detecção de mudanças no começo e final de arquivos. Também foi alvo deste trabalho a melhoria do desempenho do sdhash, na qual foi realizada a troca da função de hash SHA-1 por MD5 e FNV, o que representou, no caso de FNV, uma diminuição de aproximadamente 58% a 62% no tempo de execução sem perda de precisão.

Por fim, algumas considerações sobre pontos que podem ser objeto de estudo em trabalhos futuros. Atualmente o sdhash gera um resumo equivalente a aproximadamente 2,6% do tamanho do arquivo original. Analisando os outros métodos de Pareamento Aproximado, percebe-se que isto é uma desvantagem. Há como buscar alternativas de estruturas de dados para armazenamento das *features* diferentes dos Filtros de *Bloom*. Outro ponto seria reduzir o tamanho da *feature* de 64 para 32 bytes, o que representaria um insignificante acréscimo no número de *features*, mas poderia melhorar sua precisão, mais especificamente na detecção de pequenas alterações.

Referências

Roussev, V. (2010, January). Data fingerprinting with similarity digests. In IFIP International Conference on Digital Forensics(pp. 207-226). Springer, Berlin, Heidelberg.

Rabin, M. O. (1981). Fingerprinting by random polynomials. Technical report.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. Digital investigation, 3, 91-97.

Breitinger, F., & Roussev, V. (2014). Automated evaluation of approximate matching algorithms on real data. Digital Investigation, 11, S10-S17.

Breitinger, F., Stivaktakis, G., & Roussev, V. (2014). Evaluating detection error trade-offs for bytewise approximate matching algorithms. Digital Investigation, 11(2), 81-89.

Roussev, V., An Evaluation of Forensic Similarity Hashes. In Proceedings of the Eleventh Annual DFRWS Conference, pp. S34-41, Aug 2011, New Orleans, LA.

G. Fowler, L. Noll, P. Vo, Fowler/Noll/Vo (FNV) Hash, ONLINE http://isthe.com/chongo/tech/comp/fnv/ - acessado em 06/09/2018.

Harichandran, V. S., Breitinger, F., & Baggili, I. (2016). Bytewise approximate matching: the good, the bad, and the unknown. Journal of Digital Forensics, Security and Law, 11(2), 4.

Lee, A., & Atkison, T. (2017, April). A comparison of fuzzy hashes: evaluation, guidelines, and future suggestions. In Proceedings of the SouthEast Conference (pp. 18-25). ACM.