FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

GABRIEL LIBONI ALCALA FREGUGLIA

CRIPTOANÁLISE APLICADA EM ALGORITMOS DE *HASHING*

Americana, SP

2016

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

GABRIEL LIBONI ALCALA FREGUGLIA

CRIPTOANÁLISE APLICADA EM ALGORITMOS DE *HASHING*

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do Prof. Me. Clerivaldo José Roccia

Área de concentração: Criptografia

Americana, SP.

2016

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

F931c FREGUGLIA, Gabriel Liboni Alcala
Criptoanálise aplicada em algoritmos de *hashing* / Gabriel Liboni Alcala Freguglia. – Americana: 2016.
51f.

Monografia (Curso de Tecnologia em Segurança da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Ms. Clerivaldo Jose Roccia

1. Criptografia I. ROCCIA, Clerivaldo Jose II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.518.5

Gabriel Liboni Alcala Freguglia

APLICAÇÃO DA CRIPTOANÁLISE EM ALGORITMOS DE *HASHING*

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Criptografia.

Americana, 06 de dezembro de 2016.

Banca Examinadora:

Clerivaldo José Roccia
Mestre
FATEC Americana

Humberto Celeste Innarelli
Doutor
FATEC Americana

Raquel Gomes Marcelino
Mestre
FATEC Americana

AGRADECIMENTOS

Agradeço primeiramente a meus pais, por terem feito de mim quem eu sou hoje.

Agradeço à minha companheira Gabriela que passou estas semanas no mesmo barco que eu e me mantendo são.

Não obstante, também devo a meus colegas de classe, que por vários dias foram os únicos motivos de comparecer à faculdade. Em especial, agradeço ao Davi, por me fornecer o *hardware* necessário para a realização dos testes, além de ouvir todas as minhas reclamações no fim do curso. Aos colegas Rafaela, Nelio, Guilherme, Leonardo, Hugo, e Renan em especial por transformar noites maçantes em intervalos de risada.

Aos professores Clerivaldo e Alexandre Aguado, que me mostraram um novo mundo dentro da Segurança da Informação, ao professor Leandro Halle pelo rápido auxílio e disponibilidade quando necessitei, e ao professor Benedito Cruz por todos os semestres que tivemos aulas com ele.

Por último, mas não menos importante, a todos que participaram da minha vida até hoje, pois sem cada um de vocês com certeza não estaria aqui hoje.

RESUMO

A criptografia é extremamente importante dentro da segurança da informação, funcionando como uma espécie de fechadura digital mantendo os dados seguros. Em um mundo interconectado, em que informações sensíveis têm de permanecer disponíveis todo o tempo, a criptografia é peça-chave para a manutenção da segurança dos dados na Internet, permitindo que somente quem possua as chaves consiga abri-la ou, no caso, decifrá-la. Porém do mesmo modo que as fechaduras físicas, possuem falhas: assim como uma chave-mestra consegue abrir diversas fechaduras, ou um pé-de-cabra pode forçar uma porta a se abrir, existem técnicas utilizadas por pessoas especializadas que visam quebrar esta segurança fornecida. Em fechaduras físicas, existem modelos mais e menos resistentes, mais velhos e mais novos, e esta analogia também vale para a criptografia: diferentes métodos de encriptação existem, assim como diferentes algoritmos foram utilizados ao longo da história. Este estudo tem como objetivo analisar a criptografia contemporânea, examinando historicamente como a criptografia ajuda na manutenção dos pilares da segurança e permitindo uma visão de como o advento da Internet influenciou na evolução deste campo. Em seguida, foi feito um exame da criptoanálise, com foco nas técnicas de quebras de senha e quais são as principais variáveis consideradas pelos criptoanalíticos ao efetuar um ataque a bancos de senha. Na parte prática desta monografia, uma simulação de ataque a senhas criptografadas foi executada utilizando um computador doméstico, efetuando uma análise comparativa dos algoritmos estudados. Analisando os resultados, espera-se que seja possível aumentar o entendimento sobre a segurança das informações armazenadas na rede, tanto pelos usuários, compreendendo como fortalecer suas senhas, como para analistas, engenheiros, arquitetos e desenvolvedores de software, exibindo os algoritmos já legados e métodos criptográficos para criar sistemas mais resistentes.

Palavras Chave: criptoanálise, *hashing*, senhas

ABSTRACT

Cryptography is extremely important inside Information Security, working as a digital keylock and keeping the data safe. In an interconnected world in which sensitive information must remain available all the time, cryptography plays a key role in maintaining the data safe on the Internet, allowing only those who have the keys to unlock it – or, in this case, to decipher it. But just like regular locks, they have flaws: like a master key can open many different locks, or a crowbar can force a door open, there are techniques used by specialized people to break the security offered by this. Different models of physical locks provide safety in a variety of degrees, they can be older or newer, and this analogy fits cryptography too: many different encryption methods exists, and a plethora of algorithms have been used throughout the history. This study aims to analyze modern cryptography, showing historically the algorithms that help maintaining the pillars of security and viewing how the Internet growth influenced this field's evolution. Following, a view was made on cryptanalysis, focusing on the password breaking techniques and what are the main variables that cryptanalytics need to have in mind when attacking a password database. On the practical part of this monograph, a scaled attack simulation against encrypted passwords was accomplished using a common home computer. Analyzing the results, it is hoped that a better understanding can be achieved about the real security of the information stored on the web, both by users – knowing how to strengthen their passwords – and by software analysts, engineers, architects and developers, showing algorithms no longer used, and different encryption methods to use when creating systems, making them safer.

Keywords: *cryptanalysis, hashing, passwords*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

3DES: *Triple* DES, algoritmo criptográfico que funciona aplicando o DES três vezes consecutivas.

CPU: *Central Processing Unit*, conhecido popularmente como processador. Peça central de processamento em um computador.

DES: *Data Encryption Standard*, algoritmo criptográfico desenvolvido na IBM.

EFF: *Electronic Frontier Foundation*, organização não-governamental que atua na proteção dos direitos digitais dos cidadãos.

FLOPS: *Floating Point Operations Per Second*, medida de análise de cálculo de processadores digitais que considera a quantidade de operações matemáticas com ponto flutuantes (números com casa decimal) feitas por segundo.

GPU: *Graphical Processing Unit*, conhecido popularmente como placa de vídeo. Responsável por efetuar cálculos vetoriais para exibição de aplicativos gráficos.

HSM: *Hardware Security Module*, um módulo de *hardware* que lida somente com criptografia.

MD4: *Message-Digest* 4, algoritmo criptográfico criado com intuito de ter rapidez na geração de *hashes*.

MD5: *Message-Digest* 5, algoritmo criptográfico criado por Ron Rivest, amplamente utilizado na Internet, evolução do MD4.

NBS: *National Bureau of Standards*, agência governamental norte-americana responsável por gerenciar diversos tipos de padrões. Gerenciava inclusive padrões de tecnologia quando não existia uma agência especializada para tal.

NIST: *National Institute of Standards and Technology*, agência governamental norte-americana responsável por gerenciar os padrões de tecnologia.

NSA: *National Security Agency*, agência governamental de inteligência norte-americana.

PCI-SSC: *Payment Card Industry Security Standards Council*, conselho de entidade montada pelas empresas de cartão de crédito responsável por criar e fiscalizar padrões de segurança das empresas que fornecem serviços.

SHA: *Secure Hash Algorithm*, algoritmo criptográfico criado pelo NSA na década de 1990. Foi revisado várias vezes desde sua criação.

SHS: *Secure Hash Standard*, nome original do algoritmo que viria a se tornar o SHA.

SUMÁRIO

1 INTRODUÇÃO

Um grande número de pessoas armazena informações pessoais em algum lugar da Internet. Sejam fotos, dados financeiros, conversas, amizades, ou ainda informações sigilosas de empresas e governos, saibam elas ou não. Em uma era interconectada estas informações necessitam ser compartilhadas, para serem acessadas por pessoas em diferentes lugares. E nada mais apropriado para isto do que a rede mundial de computadores.

A Internet, desde seu princípio, tem como objetivo o compartilhamento de dados. Porém nem todos os dados devem ser acessados por todas as pessoas, e para que seja mantida esta confidencialidade é necessário que estes dados sejam armazenados utilizando técnicas de segurança da informação. Por mais que existam sistemas de identificação de usuários e seja mantida a segurança física dos locais aonde estes dados são armazenados, interceptação de dados e invasões lógicas são incidentes que ocorrem com grande frequência. Considerando que inúmeras novas vulnerabilidades novas em todos os tipos de tecnologia são descobertas a cada dia, inclusive empresas grandes do ramo de tecnologia como Yahoo!, LinkedIn, Verizon, entre diversas outras acabam sendo notícia de vazamentos massivos de dados, conforme apontam estudos da própria Verizon (2016).

Esta preocupação, todavia, não é nova. Conforme Christensen (2007) coloca, na segunda guerra mundial, a comunicação militar já se fazia via rádio, divulgando a informação no ar e permitindo que qualquer um que tivesse acesso à frequência certa conseguisse receber estes dados, sendo o recipiente desejado ou não. Os alemães conseguiram vantagens táticas ao mitigar este risco na II Guerra Mundial, utilizando com um canal seguro para comunicação interna das tropas.

Para obter este feito, a Alemanha utilizou-se da criptografia, uma técnica que tem como objetivo transformar a mensagem que se deseja enviar em um texto ilegível através de algoritmos ou cifras. A criptografia impede que mesmo que se capture a mensagem, não seja possível entendê-la sem o conhecimento exato de como ela foi transformada.

Porém, será que após todos estes anos de evolução, estes algoritmos utilizados atualmente conseguem ainda manter a segurança desejada? A segurança da informação se mantém sobre três pilares principais: confidencialidade, integridade e disponibilidade. A criptografia auxilia na manutenção não de um pilar, mas de todos, direta ou indiretamente: ao não permitir que leitores não-autorizados leiam a mensagem, mantém a confidencialidade; ao falhar na identificação de um *checksum* - impressões digitais do tamanho de um arquivo, para se validar sua integridade e origem - evitando que seja lida uma mensagem incorreta, auxilia na integridade; e ao garantir a possibilidade de que dados sejam mais acessíveis remotamente, ajuda a sustentar a disponibilidade.

Tendo em vista a importância da criptografia, este trabalho tem como objetivo geral fornecer uma visão geral da real segurança que a criptografia fornece no que tange o armazenamento de senhas, fornecendo com base em dados comparativos uma explicação para as boas práticas recomendadas para a criação de senhas.

Especificamente, visa estudar a evolução da criptografia, exibindo algoritmos padrões dentro da informática e os motivos pelos quais deixaram de ser, visualizando como suas evoluções estão interligadas e quais seus principais usos; pretende fornecer uma explicação teórica dos métodos de ação dos atacantes para obter informações a partir de dados criptografados; será efetuado um ataque prático com base na teoria analisada; resultados serão coletados e analisados para que se entenda os intuitos dos ataques; e por fim, as boas práticas são incentivadas novamente em luz dos resultados e com embasamento científico.

O método escolhido para se abordar a parte prática foi o experimental. Segundo Goode e Hatt (2006), este método desempenha uma função dupla, permitindo a descoberta de conexões casuais, necessário ao se estudar relações similares entre diferentes algoritmos, e para atingir a demonstrabilidade, foco principal deste estudo.

No capítulo inicial deste trabalho, será exibida uma breve história da criptografia, qual sua importância desde o surgimento do computador e o porquê de se necessitar de criptografias fortes para manter seguros os dados os usuários, analisando brevemente os ataques e vazamentos de dados de empresas grandes.

No segundo capítulo, é exibido o lado do defensor, mostrando como a criptografia foi utilizada com o passar dos anos, e como é utilizada hoje. Será apresentada a história de alguns dos algoritmos criptográficos mais utilizados durante a computação contemporânea e legada, e quais motivos as fizeram se tornar padrão e posteriormente sair deste posto. O terceiro capítulo abordará o lado dos ofensores, exibindo diferentes abordagens que podem ser utilizadas em um ataque, e como o avanço dos hardwares impactou a criptoanálise.

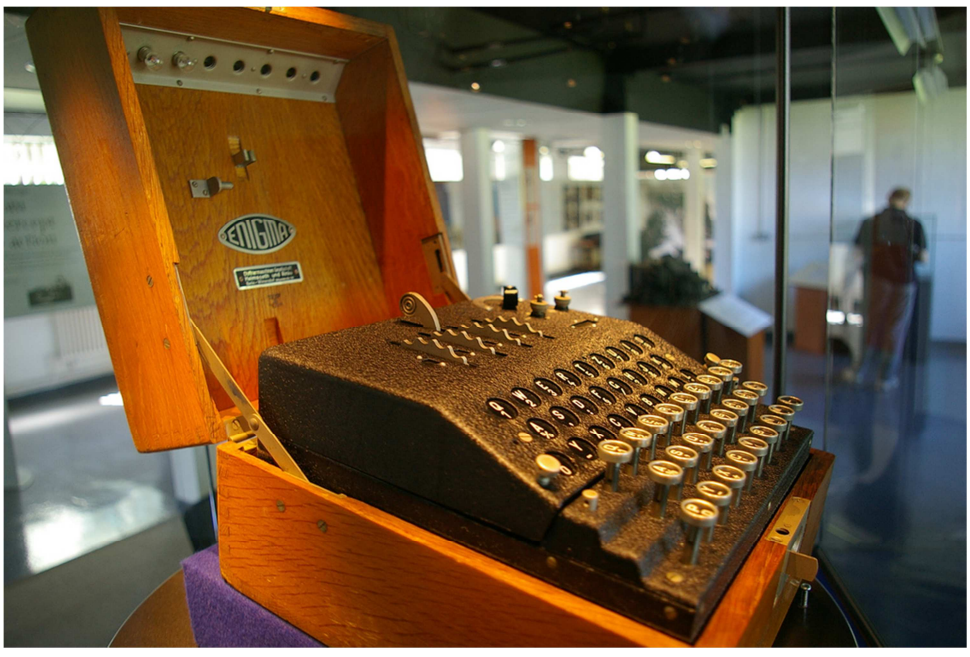
Em seguida, no quarto capítulo serão feitos testes de pequena escala simulando um ataque *hacker* a um banco de dados de senhas, e em seguida testes usando diferentes amostras em um algoritmo específico para se analisar a força das senhas. Por fim, os resultados serão coletados, explicados e analisados e visto o que pode ser feito pelos usuários para que seus dados não sejam comprometidos, o quanto realmente se pode confiar na segurança de sua informação na Internet e se há ou não métodos que podem ser adotados por pessoas responsáveis no desenvolvimento de sistemas que possam assegurar a segurança dos usuários.

2 CRIPTOGRAFIA

> “A matemática é tal que sua investigação abrange o não-físico, aquele que a usa na investigação de objetos físicos [...] está depleto de verdade.” – (IVRY, 1974)

O registro mais antigo de criptografia que se conhece foi encontrado em aproximadamente 1900 a.C., na cidade de Menet Khufu, no Egito (KAHN, 1973). Não era nada como o que conhecemos hoje, nem tão complexo quanto. Este registro nada mais é do que a troca de hieróglifos comuns por outros rebuscados e de raro uso. À época, servia não para impedir a leitura, mas sim para enaltecer a dignidade e autoridade do nobre cuja tumba foi inscrita com tais hieróglifos, Khnumhotep II. Porém a finalidade, frequência de uso, e complexidade da criptografia foi alterada muito com o tempo, passando a obter ampla visibilidade a partir da segunda guerra mundial (CHRISTENSEN, 2007). Os matemáticos polacos Jerzy Rózycki, Hekryk Zygalski e Marian Rejewski conseguiram, através da determinação de padrões, originar um algoritmo para decifrar a ordem das posições dos rotores da máquina Enigma (Figura 1), utilizada pelos alemães para trocar informações seguramente na época.

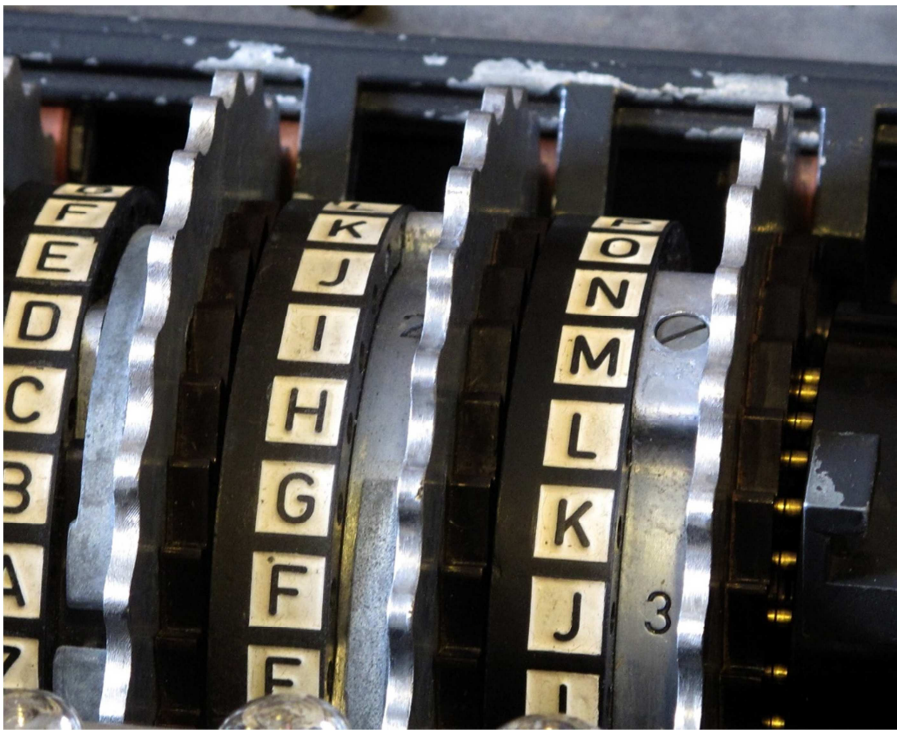
Figura 1: A máquina Enigma



Fonte: HISTORY (2015)

Conforme Christensen (2007) explica, a máquina Enigma se assemelhava a uma máquina de escrever com rotores internos (Figura 2). Tais rotores podiam ser girados e, de acordo com a combinação de posições nas quais os colocavam, cada pressionamento de tecla produzia uma saída diferente. Ao posicionar os rotores da mesma maneira em que estavam na máquina que gerou o texto e digitar o texto recebido, a saída era o texto original, a mensagem em texto claro.

Figura 2: Rotores da Enigma



Fonte: OLEKSIAK (2003)

Este método, após o advento da criptografia como objeto de estudo, ficou conhecido como criptografia de chave privada, ou criptografia simétrica: a mesma chave é aplicada para criptografar o texto claro e para decriptografar a mensagem transformada.

Esta técnica é utilizada amplamente em vários mercados, como o de cartão de crédito. Conforme o PCI-SSC (2016), em empresas que armazenam e processam dados de cartão de crédito, o consórcio das empresas de cartão de crédito obriga que os dados sejam criptografados e decriptografados sempre através de uma máquina especial chamada *Hardware Security Module*, ou HSM (Figura 3), para aplicar a criptografia simétrica 3DES, de modo que estes dados não fiquem transitando na memória dos computadores utilizados para fins diversos.

Figura 3: nShield Connect+ - um *Hardware Security Module*

Fonte: NEODATA (2016)

Ao longo dos anos, a criptologia (palavra de origem grega, junção das palavras *kryptós*, “escondido”, e *logia*, “estudo”, denota o campo responsável pelo estudo da criptografia em si e seus algoritmos) avançou bastante, acompanhando a evolução tecnológica que fez os computadores se tornarem cada vez mais cruciais na troca de informações. Por mais que a criptografia simétrica tenha seu uso, este modelo passou a não mais atender a todas as necessidades de troca de dados digitais, principalmente pelo problema logístico das chaves necessitarem ser pré-compartilhadas (SINGH, 1999).

Transações financeiras online necessitam ser criptografadas, caso contrário a simples obtenção de pacotes trafegados permite ao atacante obter dados críticos como dados de cartão de crédito e senhas bancárias. Porém protegê-las também com criptografia simétrica também não seria viável, devido à necessidade de se fornecer a chave ao usuário, ou seja, de posse da chave os atacantes também conseguiriam acesso fácil aos dados. Para casos como este, se é usada a criptografia assimétrica, ou criptografia de chave pública: um par de chaves é utilizado, um disponível ao público, e outro fica de posse do servidor (SALOMAA, 2013). Mensagens criptografadas com uma somente conseguem ser decriptadas com a outra, e vice-versa. Assim, é possível disponibilizar uma chave ao público e ainda assim manter a confidencialidade de uma conversa ou troca de dados.

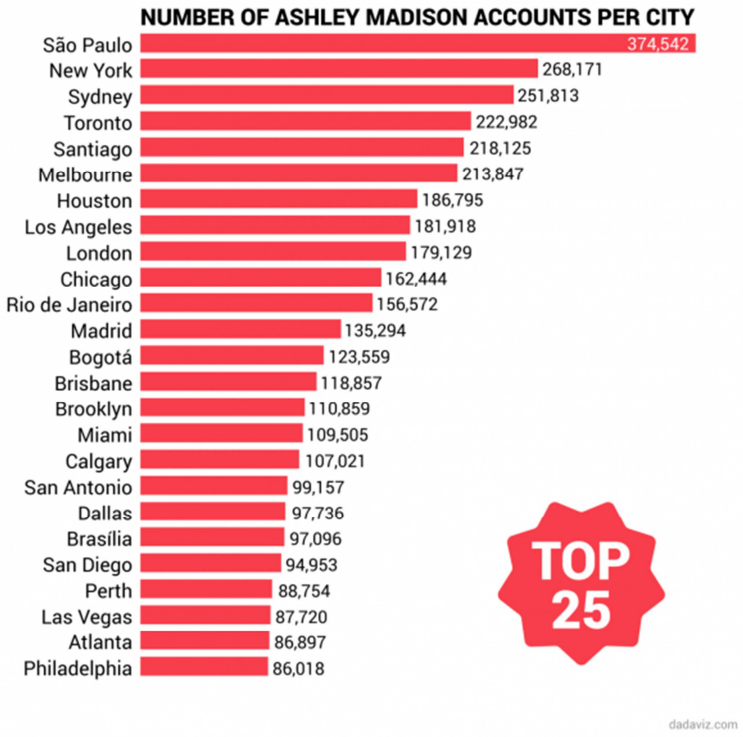
O armazenamento de senhas, por sua vez, não se encaixa em nenhum dos casos anteriores, já que não necessita ser uma conversa de duas vias. Por ser apenas um mecanismo de autenticação, não necessita ser transformado em texto claro, mas necessita que: 1) a mesma sequência de caracteres, ao ser criptografada, gere sempre o mesmo texto cifrado, ou seja, deve ser determinístico, e 2) combinações diferentes não gerem textos cifrados iguais entre si. Esta técnica de criptografia de mão única é conhecida como *hashing*. Atingindo isto, se garante que a senha digitada corretamente pelo usuário, e somente ela, consiga ser criptografada novamente e irá sempre coincidir com o que foi armazenado no banco. Será mostrado a seguir a importância de que estes algoritmos sejam implementados de maneira segura, em virtude do crescimento da utilização da Internet.

2.1 OBTENÇÃO ILEGAL DE DADOS CRIPTOGRAFADOS

Um dos métodos mais comuns que atacantes se utilizam para tentar descobrir senhas se dá de forma *off-line*. Este tipo de ataque se dá em duas etapas: inicialmente são efetuados ataques direcionados e, de posse dos dados obtidos através deste ataque direcionado, são feitos os ataques às senhas em si. Ataques direcionados podem tomar qualquer forma, como interceptação de e-mails ou pacotes, engenharia social entre diversos outros. Estes têm como objetivo um alvo em específico, como uma empresa, um funcionário, ou uma pessoa de interesse, e são personalizados de acordo com a situação.

Após o ataque direcionado e o subsequente acesso aos dados, estes são então subtraídos e copiados para um local de fácil acesso ao invasor. Os atacantes têm como intenção obter o maior número de informações possíveis a partir dos dados obtidos, para obter lucro com a venda das informações ou para conseguir acessos mais valiosos com os dados obtidos. Um vazamento de dados de clientes empresariais da gigante americana de telefonia Verizon foi anunciado no mercado negro por cem mil dólares americanos no ano de 2016, ou dez mil dólares por cada pacote com cem mil registros. A mesma Verizon disponibiliza, anualmente, uma lista dos vazamentos de dados mais importantes ocorridos. Entre eles, temos empresas e serviços de grande público como Snapchat, eBay, iCloud, Sony Pictures Entertainment (VERIZON, 2015), Ashley Madison (VERIZON, 2016), Yahoo! (REUTERS, 2016), entre outros. Na Figura 4, vemos um exemplo de gráfico montado com base em dados extraídos do vazamento do Ashley Madison:

Figura 4: Gráfico baseado em dados vazados do site Ashley Madison



Fonte: PATON (2015)

Porém, para obter os acessos mais valiosos, os hackers necessitam inicialmente obter a senha que foi capturada no ataque, em texto claro. De posse da senha combinada com outros dados obtidos, como e-mail, nome, documento, é possível iniciar a tentativa de obtenção de outros acessos. Caso o usuário utilize a mesma senha em diversos sistemas, os atacantes conseguem acesso a dados de valor dele.

Tendo em vista a frequência e o tamanho dos *sites* que se tornam alvos de ataques *hackers*, é necessário que os algoritmos utilizados na encriptação das senhas sejam eficientes, de modo a não revelar facilmente os dados caso estes venham a ser expostos. Vários algoritmos se tornaram padrão com o tempo e estes mesmos acabaram sendo substituídos conforme falhas foram sendo achadas ou foram subjugados pelo poder computacional crescente. Serão exibidos agora alguns destes algoritmos, suas histórias e utilizações.

2.2 ALGORITMOS MAIS UTILIZADOS

O vazamento de informações do site de infidelidade Ashley Madison, em 2015, foi bastante divulgado pois, mesmo antes de qualquer senha ser quebrada, expôs várias contas de e-mail de usuários cadastrados no site de infidelidade, causando furor público. *Experts* em segurança analisaram os dados deste vazamento, que foi disponibilizado publicamente, e ao analisar as senhas foi descoberto que uma criptografia forte foi utilizada (no caso, o algoritmo bcrypt com fator de trabalho 12, e a técnica *salt* para composição das senhas, sendo ambos explicados mais à frente). Esta combinação foi suficiente para inviabilizar uma quebra massiva destas senhas (GOODIN, 2015), apesar do fato constatado por Constantin (2015) de que falhas de programação permitiam que usuários mantivessem senhas fracas armazenadas desde 2012.

Porém discussões sobre técnicas de programação estão fora do escopo desta dissertação, que é analisar a real segurança da criptografia em si. No que tange este aspecto, os responsáveis pelo site Ashley Madison utilizaram técnicas avançadas recomendadas, segundo Malvoni e Knezovic (2014) – a adição de *salts* às senhas,

juntamente com a utilização do algoritmo bcrypt, são técnicas comprovadamente eficientes para aumentar a segurança.

A quebra de senhas criptografadas se dá basicamente através de duas abordagens se feita em massa: tentativas utilizando todas as combinações de caracteres – conhecido como força bruta – ou através de bancos de *strings* (sequências de caracteres) – ataques de dicionário ou *rainbow tables*, quando se usam bancos pré-calculados. Estas técnicas serão analisadas mais profundamente no próximo capítulo.

Segundo Morris e Thompson (1979), os *salts* visam inviabilizar ao máximo estas duas abordagens. Os *salts* são caracteres pseudoaleatórios adicionados ao final da senha digitada pelo usuário. O método de aplicação do *salt* pode variar, adicionando-o à senha em texto claro e criptografando o texto concatenado, ou já após a criptografia da senha isolada, adiciona-se o *salt*, criptografado ou não, e em seguida criptografa-se novamente. Em algoritmos criados já com o intuito de armazenamento de senhas, o *salt* é entrado como um dos parâmetros, sendo tratado nativamente por estes.

Isto dificulta ambos os métodos de abordagem: no caso da força bruta, aumenta a quantidade de caracteres que foi utilizada para gerar o *hash*, e inviabiliza ataques com listas de palavras.

A aplicação de *hashing* nas senhas se dá desde a década de 1970, conforme reportam Malvoni e Knezovic (2014). Ao longo dos anos, vários algoritmos de *hashing* foram utilizados, e foram sendo substituídos por outros conforme algoritmos mais recentes foram surgindo ou falhas foram sendo achadas.

Pelo fato dos algoritmos de *hash* serem determinísticos e singulares por entrada, estes são considerados inseguros quando são descobertas falhas que permitem que entradas diferentes produzam o mesmo resultado criptografado. A este fenômeno dá-se o nome de colisão:

> Uma colisão interna ocorre se uma função dentro de um algoritmo criptográfico processa diferentes argumentos de entrada, mas retorna um argumento idêntico na saída. [...] O termo colisão interna implica em si que a colisão não pode ser detectada na saída do algoritmo. (SHRAMM *et al*, 2004)

Algoritmos diferentes foram surgindo através dos anos, se tornando padrão de uso governamental e comercial, e devido ao amplo uso se tornaram alvo de ataques de *hackers* e pesquisadores do mundo todo com o intuito de quebrar estes algoritmos, para fins de reconhecimento acadêmico ou financeiro. Abaixo seguem alguns dos algoritmos de *hashing* mais importantes desde a década de 1970.

2.2.1 DES

O DES não é propriamente um algoritmo de *hashing*, porém foi o predecessor destes. Foi desenvolvido na IBM no início da década de 1970 e, em 1977 foi submetido ao NBS (National Bureau of Standards) dos Estados Unidos da América, para se tornar um padrão federal de criptografia (ISLAM; AZAD; PATHAN, 2014). Sob consultoria da NSA (National Security Agency), uma versão do DES com poucas modificações foi aceita como padrão a fim de armazenar dados importantes não-confidenciais do governo americano.

O DES criptografa os dados em blocos de 64 *bits*, utilizando uma chave de 56 *bits*. Através da utilização da cifra de Feistel, gera 16 chaves de 48 *bits* diferentes a partir da chave inicial de 56 *bits* e processa a mensagem através de 16 rodadas de permutação e substituição.

A permutação, na criptografia, é um passo dentro dos algoritmos que desloca *bits* entre si, baseado em tabelas que contém dados de posições a serem alternadas. A substituição se baseia em outras tabelas, chamadas de S-Boxes, que possuem valores fixos e são acessadas utilizando partes dos dados que estão sendo tratados como índices.

Apesar de não terem sido encontradas colisões, o algoritmo foi comprovado inseguro pela organização sem fins lucrativos Electronic Frontier Foundation (EFF,

1998), que desenvolveu em 1998 uma máquina, chamada de “EFF DES Cracker”, invalidando as argumentações feitas por oficiais do governo norte-americano de que seriam necessários vários meses e uma rede de computadores milionária para quebrar uma única mensagem. O EFF DES Cracker, criado com o custo estimado inferior a duzentos e cinquenta mil dólares americanos, quebrou uma mensagem em três dias.

O DES teve uma sobrevida graças ao 3DES (Triple DES), que consiste em basicamente reaplicar o DES três vezes seguidas para se aumentar a segurança. O 3DES é bastante utilizado em aplicações financeiras, sendo seu uso bastante restrito fora deste mercado, justamente por oferecer pouca segurança. Sua sobrevivência se dá pelo fato de que as trocas de dados entre instituições financeiras ter ao menos mais uma camada de criptografia forte por cima dos dados encriptados com o 3DES, e uma vez que estes dados estejam decriptados, há a necessidade de velocidade no processamento.

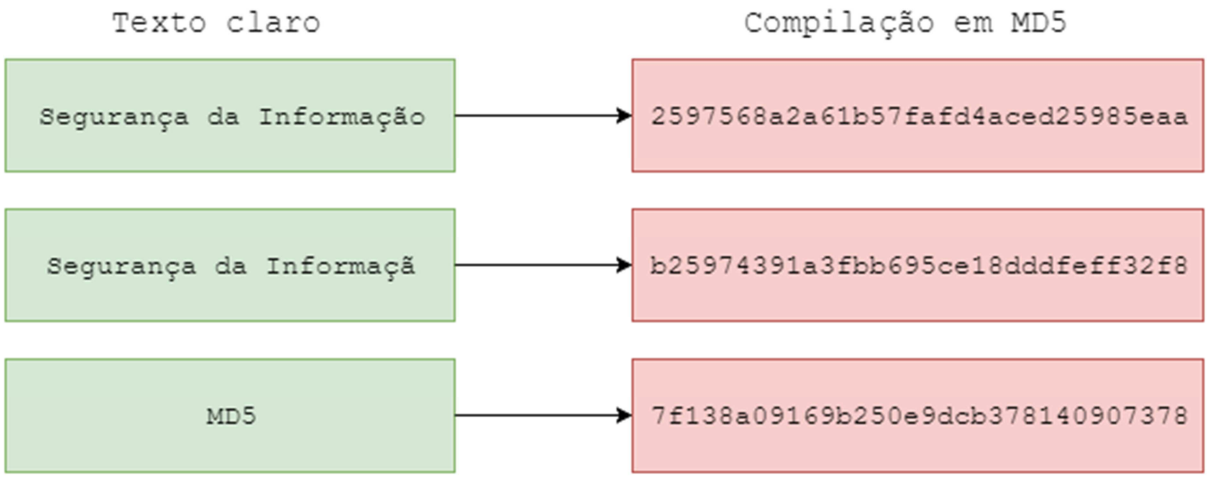
2.2.2 MD5

O MD5 foi criado por Rivest em 1992 (STEVENS, 2006), para estender seu predecessor, MD4, considerado inseguro (RIVEST, 1992). O MD4, desenvolvido com a velocidade de processamento sendo o foco principal, não garantia mais a segurança, inviabilizando sua utilização em virtude dos já crescentes riscos de ataques criptoanalíticos à época.

O MD5, assim como o DES, é um algoritmo de encriptação de função genérica, ou seja, não tem como foco principal o *hashing* de senhas. Possui como característica a entrada de uma mensagem de tamanho variável e produz sempre uma saída de 128 *bits*, conhecida como “impressão digital” ou “compilação” da mensagem.

Esta é uma característica inerente aos algoritmos de *hashing*: para dificultar a análise, a saída é sempre composta de um número predefinido de bits, evitando que o atacante saiba de antemão a quantidade de caracteres que foi encriptada. Um exemplo desta característica é exibido na Figura 5.

Figura 5: Compilações de mensagens em MD5



Fonte: Elaborado pelo autor (2016)

Conforme Stevens (2006) apresenta, doze anos após sua criação, em 2004, foi identificada a primeira colisão neste algoritmo, comprovando sua obsolescência. O MD5 ainda é utilizado na computação moderna, mas na maioria das vezes como um método de comparação de *checksum*. Devido às colisões encontradas, a utilização deste algoritmo não é recomendada para esta finalidade, sendo mais utilizado em sites legados e/ou com técnicas rudimentares de programação.

2.2.3 SHA

O SHA (Secure Hash Algorithm) foi criado em 1993 pelo NSA (BIHAM et al, 2005) sob o nome inicial de SHS (Secure Hash Standard), segundo Rijmen e Oswald (2005). O SHS, que teve como base o MD4, foi escolhido como padrão americano de algoritmo de *hash* pelo NIST (National Institute of Standards and Technology).

Em 1995, uma alteração foi feita no SHA-0 para a correção uma falha técnica e, apesar de nenhuma justificativa ter sido feita acredita-se, segundo Chaubaud e

Joux (1998), que o NSA tenha conseguido encontrar colisões no código. A esta versão corrigida se deu o nome de SHA-1.

O SHA-1 foi implementado amplamente ao redor do mundo, e segundo Wang, Yin e Yu (2005) foi adotado como padrão em vários mercados, principalmente em assinaturas digitais, e utilizado em diversos sistemas para autenticação de usuários, reconhecimento de chaves e geração pseudoaleatória de números.

Em 2015 foi efetuado com sucesso o primeiro ataque real completo de colisão ao SHA-1 (STEVENS et al, 2016) e Microsoft, Google e Mozilla anunciaram que a partir de 2017 seus navegadores de Internet não mais suportarão certificados digitais criptografados com este algoritmo.

A principal fraqueza descoberta no SHA-1 está em um dos passos de expansão das chaves. Este passo consiste em transformar uma sequência de caracteres em uma sequência maior, repetindo determinados bits de acordo com tabelas predefinidas. O SHA-1 possui uma falha técnica que permite que, com a abordagem certa, estes passos fiquem previsíveis e sejam encontrados padrões.

Em 2002, antes da confirmação da obsolescência do SHA-1 o NIST anunciou o sucessor do SHA-1, o SHA-2. Este último possui três implementações diferentes de acordo com a quantidade de bits de saída, sendo 256, 384 ou 512 (SKLAVOS; KOUFOPAVLOU, 2005). Para evitar confusões, o SHA-2 foi renomeado de acordo com os *bits* de saída, SHA-256, SHA-384 e SHA-512, apesar de algumas literaturas os citarem como SHA-2 (256), SHA-2 (384) e SHA-2 (512). As principais diferenças em relação ao SHA-1 são o tamanho da mensagem compilada e a possibilidade de se utilizar maiores blocos de entrada – o SHA-512 permite blocos de entrada de até 2^{128} bits (CHAVES et al, 2006).

O SHA-3 foi escolhido através de uma competição pública realizada pelo NIST em 2012, tendo o algoritmo Keccak como vencedor, com o SHA-3 se tornando oficial em 2015 (NIST, 2015). Este algoritmo não foi criado com intenção de substituir o SHA-2 imediatamente, devido a vulnerabilidades não haverem sido descobertas, mas sim como um backup caso alguma falha seja encontrada, dado o tempo necessário para o desenvolvimento de novos padrões.

2.2.4 bcrypt

Ao contrário dos algoritmos citados anteriormente, que possuem função genérica, o bcrypt é um algoritmo feito especificamente para utilizar um alto poder computacional através da divisão do texto de entrada e aplicação da criptografia em cada uma das partes por diversas vezes de acordo com a necessidade dos usuários (ERTAUL *et al* 2016). Este cenário é específico para o armazenamento de senhas, dificultando os ataques em massa por deixá-los impraticáveis de serem realizados em tempo hábil.

O bcrypt é baseado na cifra de blocos Blowfish, e foi criado especificamente para ser resistente à ataques de força bruta e continuar robusto independentemente dos avanços tecnológicos em *hardware* (MALVONI; KNEZOVIC, 2014). Isto se dá pelo fato de utilizar caixas de permutação variáveis, o custo alto de configuração das chaves e o fator de trabalho que é definido pelo usuário.

As tabelas de permutação variáveis criam a necessidade de cada instância em que está rodando o algoritmo necessitar alocar um espaço na memória equivalente a 4KB, limitando sua utilização em processos paralelos. As chaves com custo alto reduzem a capacidade de operações por *clock* dos processadores, e o fator de trabalho irá definir quantas vezes o algoritmo irá rodar recursivamente, com o único intuito de utilizar mais processamento.

Um diferencial deste algoritmo e de todos os pensados especificamente para armazenamento de senhas é a integração nativa dos *salts*. Em algoritmos genéricos, a aplicação do *salt* deve ser feita separadamente, enquanto em algoritmos como o bcrypt o *salt* é utilizado como parâmetro de entrada e integrado na geração dos *hashes*.

3 TÉCNICAS DE *HACKING*

A força do algoritmo utilizado para armazenamento das senhas está ligada diretamente com a segurança: quanto mais difícil e custoso for para o atacante gerar *hashes* baseados em combinações de caracteres, menos comparações conseguirá fazer.

A afirmação acima é verdadeira para ataques *off-line*, ou seja, quando o atacante obtém uma cópia dos dados e efetua seus ataques diretamente nestas cópias, sem contatar o servidor em que estes dados estavam armazenados originalmente (GOYAL *et al*, 2005). Todavia, este não é o único tipo de ataque: nos ataques *online*, os atacantes tentam descobrir a senha através do próprio sistema.

Conforme a questão da segurança dos dados foi tendo mais visibilidade, os ataques *online* foram caindo em desuso, pois vários sistemas bloqueiam a conta do usuário, forçam uma espera de tempo predeterminada, entre outras medidas de segurança, caso a senha seja entrada incorretamente por um número pré-estabelecido de vezes. Devido a estas restrições esta técnica é mais utilizada para prejudicar diretamente o usuário privando-o de acesso, mas seu uso para obtenção direta de senhas se tornou inviável conforme sistemas foram adotando medidas mais estritas de segurança.

3.1 ATAQUES A BANCOS DE DADOS *OFF-LINE*

Tendo que os ataques massivos mais comuns são feitos *off-line*, é necessário então que os dados criptografados sejam mantidos seguros aumentando o tempo necessário para que o atacante consiga achar a combinação de caracteres idêntica à que foi criptografada.

Existem diversos meios para os atacantes atingirem este objetivo, e aplicam variadas técnicas como ataques de dicionário, ataques com *rainbow tables*, e força bruta. A aplicação de cada uma das técnicas depende muito do cenário e da capacidade de processamento disponível a tais atacantes.

3.1.1 ATAQUES DE DICIONARIO

De acordo com Shirey (2007), ataques de dicionário são aqueles que utilizam a técnica de força bruta realizando testes com todas as combinações de caracteres contidas em uma longa lista.

Esta lista, chamada de arquivo de dicionário ou simplesmente lista de palavras, contém geralmente palavras presentes em um dicionário e outras combinações prováveis de senha, podendo ser alteradas pelos atacantes para se adequarem ao ataque que está sendo feito, como adicionar termos relativos a uma área específica, gírias, ou ainda adicionar senhas comuns obtidas a partir de ataques anteriores.

Esta técnica visa aumentar a eficácia do ataque, descartando combinações de caracteres cujas entropias e/ou complexidades são muito altas para serem utilizadas e lembradas por seres humanos (NARAYANAN E SHMATIKOV, 2005), mas ainda assim mantendo um nível de complexidade exigido dos usuários pelo sistema do qual os dados foram subtraídos.

Porém, ataques de dicionário têm como desvantagem o fato de não serem eficazes em todos os cenários. Caso a senha não conste na lista em que se está testando, ela não será descoberta. Este é um ponto pelo qual muitos *websites*, ao aconselhar usuários, recomendam evitar palavras que existam no dicionário e a utilizar combinações de diferentes conjuntos de caracteres.

> Nós informalmente pesquisamos os conselhos dados a novos usuários [...] Alguns conselhos típicos são: “[Uma boa] senha deve consistir de caracteres variados ou caracteres especiais, e não devem ser compostos de palavras encontradas em dicionários. Ela não deve ser anotada em lugares facilmente acessíveis e especialmente não devem ficar perto do *login*. Ela pode ser inteira composta ou de caracteres maiúsculos ou minúsculos. (YAN et al, 2004)

3.1.2 *RAINBOW TABLES*

Ataques com uso de *rainbow tables* foram descritos inicialmente por Martin Hellman em 1980 (OECHSLIN, 2003), porém sem esta nomenclatura. Hellman

descreveu uma técnica em que uma criptoanálise poderia ter seu tempo diminuído em detrimento de um uso maior de memória ao utilizar dados pré-calculados.

Estes ataques utilizam como base dicionários de combinações de caracteres, aplicando o algoritmo desejado neste dicionário e armazenando os resultados. Os dados a serem quebrados são então comparados diretamente a estes resultados já criptografados.

Os ataques com *rainbow tables* têm vantagem caso sejam feitos repetidamente – em ataques normais de dicionário, seria necessário rodar novamente o algoritmo de criptografia para cada palavra do dicionário para se comparar novamente com o *hash* a cada ataque. Caso o ataque seja feito somente uma vez, esta técnica não apresenta ganhos sobre o ataque dicionário padrão.

O escopo deste ataque é reduzido. Além de ser vantajoso apenas em ataques repetidos, a tabela terá de ser recalculada caso outro algoritmo seja utilizado. Por exemplo, um dicionário transformado em *rainbow table* com o algoritmo SHA1 não poderá ser reutilizado em um ataque a dados criptografados através do algoritmo bcrypt.

3.1.3 FORÇA BRUTA

Ataques de força bruta, segundo o Internet Security Glossary (Glossário de Segurança da Internet) consistem em uma técnica criptoanalítica envolvendo um procedimento vigoroso que tenta variadas soluções para um problema (SHIREY, 2007).

No âmbito da criptoanálise, a técnica de força bruta consiste em tentar todas as combinações possíveis dentro de um cenário pré-estabelecido. Neste caso, não são utilizadas tabelas com combinações possíveis, ao contrário dos ataques de dicionário e *rainbow tables*. As combinações são geradas instantaneamente pelo sistema que se está utilizando, dados os conjuntos de caracteres a serem testados.

Este tipo de ataque possui menos eficácia, exatamente por testar cada combinação possível, e por isto é mais lento. Este método de aproximação, porém, é

mais eficiente, trazendo a certeza de que a combinação sempre será achada, cabendo ao atacante decidir se o tempo estimado de quebra é viável ou não para cada ataque.

3.2 *HARDWARE* UTILIZADO

Além do tipo de ataque com o qual o hacker irá realizar a criptoanálise, também entra como variável seu poder computacional. Neste ponto, o avanço da tecnologia também alterou drasticamente os ataques *off-line*.

O processador dos computadores (CPU, ou Central Processing Unit) sempre foi tido como o principal componente de computação em um sistema computadorizado (LE et al, 2010), porém os avanços na tecnologia trouxeram melhorias a todos os componentes dos computadores.

Um dos componentes que mais evoluíram foi a GPU (Graphical Processing Unit), conhecida também como placa de vídeo. O progresso da capacidade gráfica digital se deu graças ao processamento cada vez maior que estes componentes passaram a ter.

Para efeitos de comparação, em 2008, a série GT200 da nVidia atingiu o poder de 933 GFLOPS (Floating Point Operations Per Second – Operações de ponto flutuante por segundo), 10 vezes superior à CPU Intel Xeon 3.2GHz, da mesma época (PHONG *et al*, 2010).

Em 2016, ao analisar a placa de vídeo topo de linha da nVidia com o processador mais potente da Intel, conforme visto na Tabela 1, as diferenças ficam mais claras:

Tabela 1: Comparativo entre GPU e CPU topo de linha em 2016

	nVidia Titan X (2016)	**Intel Core i7-6950X Extreme Edition (2016)**
Número de núcleos	3584 núcleos CUDA	10 núcleos, 20 threads
Operações de ponto flutuante por segundo	11 TFLOPS	96.65 GFLOPS (benchmark)

Preço sugerido, de acordo com o fabricante	US$ 1.200,00	US$ 1723,00 – US$ 1743,00
Velocidade por núcleo	1.5 GHz	3.5 GHz

Fonte: NVIDIA (2016), INTEL (2016)

Tendo um preço menor e uma capacidade de processamento puro mais elevada, as GPUs são mais indicadas para criptoanálises no geral, ocupando o lugar das CPUs neste campo.

Com todas as variáveis exibidas, no próximo capítulo os algoritmos serão postos à prova, sendo testados em um ambiente controlado para efeito de comparação.

4 APLICAÇÃO PRÁTICA

Os cenários foram montados com base na fundamentação teórica, aplicando a metodologia experimental. As variáveis a serem testadas se baseiam no segundo capítulo, que engloba o objetivo principal deste projeto: os algoritmos a serem comparados. Para conseguir demonstrabilidade, foram geradas sequências aleatórias de *strings* simulando senhas, e divididos em dois cenários distintos utilizando a mesma base de teste: a técnica de força bruta aplicada em diversos algoritmos utilizados para armazenamento de senhas, utilizando uma GPU comum, testando as *strings* em quatro algoritmos escolhidos dentre os citados no decorrer deste trabalho.

A placa de vídeo utilizada é uma AMD Radeon HD5770 VaporX, com capacidade de processamento de 1360 GFLOPS em 800 núcleos funcionando a um máximo de 860 MHz.

Todos os algoritmos citados, com exceção do DES – que nunca chegou a ser utilizado massivamente para armazenamento *online* de senhas – serão levados a teste. Serão criptografadas sequências de caracteres (*strings*) geradas aleatoriamente através de um gerador *online* em no *website* random.org (Figura 6). Inicialmente serão testados diferentes conjuntos de caracteres para todos os algoritmos, sempre em sequências de oito caracteres, e na segunda etapa, serão feitos testes mais realistas utilizando sequências com conjuntos fixos e tamanhos variáveis.

Devido aos ataques de dicionário e as *rainbow tables* serem derivações da técnica de força bruta, esta última será utilizada. Apesar dos atacantes utilizarem as duas primeiras técnicas como opções primárias, é impossível a realização de testes com elas sem subjetividade. Ambas adicionam variáveis indesejadas que vão contra o intuito destes testes, que é a realização de um *benchmark* comparativo para testar a força dos algoritmos.

Figura 6: *Screenshot* do *website* random.org

Fonte: Elaborado pelo autor (2016)

Para se efetuar os testes, será utilizado o programa oclHashcat. Este *software* possui suporte nativo a todos os algoritmos estudados, além de possuir integração de baixo nível diretamente com os drivers dos processadores e GPUs, produzindo resultados mais assertivos.

Ao gerar as sequências de caracteres através do site random.org, foi visto que o *website* não disponibiliza a opção de caracteres especiais, então estes serão adicionados manualmente de maneira pseudoaleatória, baseado no conjunto exibido na Tabela 2, que será exibida mais abaixo.

Utilizando ferramentas *online* de geração de *hash*, as *strings* foram convertidas utilizando os algoritmos que seriam testados. Em seguida foram definidas as máscaras a serem utilizadas, correspondentes aos conjuntos de caracteres utilizados nos textos puros. Máscaras são parâmetros de entrada

colocados no *software* que indicam quantos e quais caracteres serão testados, no seguinte formato:

$$?\,c_1?\,c_2?\,c_3\,\ldots?\,c_n$$

No lugar de c1, entra-se o conjunto de caracteres correspondente ao primeiro caracter da *string* a ser testada, no lugar de c2, o conjunto do segundo caracter, e assim sucessivamente até a n-ésima posição, sendo n a quantidade de caracteres presente na máscara. Estes conjuntos de caracteres se encontram na Tabela 2, baseado na documentação do oclHashcat. Uma máscara para 8 caracteres numéricos seria, por exemplo, ?d?d?d?d?d?d?d?d.

Tabela 2: Conjuntos de caracteres presentes no oclHashcat

Nome do conjunto	Comando correspondente	Quantidade de caracteres	Caracteres
Letras minúsculas	?l	26	abcdefghijklmnopqrstuvwxyz
Letras maiúsculas	?u	26	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Números inteiros	?d	10	0123456789
Caracteres especiais	?s	33	«space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
Todos	?a	95	?l?u?d?s

Fonte: Elaborado pelo autor (2016)

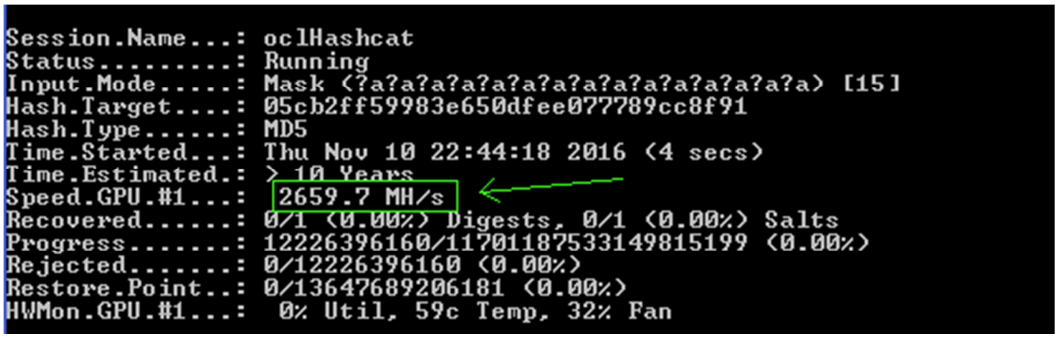
Durante os testes do primeiro cenário, foi descoberta uma limitação do *software* que impediu a visualização direta das estimativas: a partir de um determinado número de possibilidades, a quantidade total era calculada erroneamente, devido a limitações do sistema. O método foi então alterado: foram calculados os números de possibilidades baseados nas quantidades de caracteres em cada conjunto, conforme a Tabela 2. A fórmula utilizada para cálculo de possibilidades foi:

$$P = \left(\sum cp\right)^{qc}$$

Nesta fórmula, é calculado o número total de possibilidades (P) como sendo a somatória da quantidade de caracteres possíveis dos conjuntos utilizados (cp) elevada à quantidade de caracteres da *string* (qc).

Na Figura 7, são exibidas as últimas medições obtidas no *status* do oclHashcat:

Figura 7: Média de tentativas por segundo do algoritmo MD5



Fonte: Elaborado pelo autor (2016)

Nesta figura, uma parte de um *screenshot* da execução do *software* oclHashcat, é possível visualizar seu funcionamento. Os dados mais relevantes exibidos nesta tela de *status* dele são:

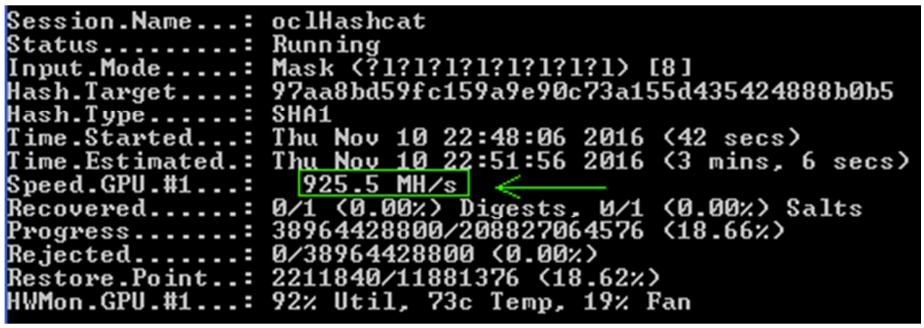
- *Input Mode*: neste item é exibido o tipo de ataque sendo feito, se é um ataque dicionário, *rainbow table*, entre outros. Neste caso é exibido *mask*, que é um ataque de máscara, outra denominação para o ataque de força bruta. Na máscara é vista uma repetição da sequência de caracteres “?a”, que significa a utilização de todos os conjuntos de caracteres, seguida da notação [15], que denota 15 caracteres.
- *Hash Target*: o *hash* de entrada que está sendo atacado.
- *Hash Type*: neste item é mostrado o algoritmo utilizado.
- *Time Estimated*: o tempo estimado de quebra. A medida máxima de tempo que o *software* consegue exibir é a mostrada na figura, ou seja,

acima de dez anos. Por este motivo, os dados foram calculados e este dado não foi utilizado diretamente dos dados coletados.

- *Speed GPU #1*: quantidade de tentativas por segundo.
- *Progress*: Quantidade de tentativas já realizadas e quantas serão feitas no total
- *HWMon GPU #1*: Status físico da placa, indicando por exemplo sua temperatura atual e a capacidade da ventoinha, para evitar superaquecimentos.

Na Figura 8, temos estas médias para o algoritmo SHA1:

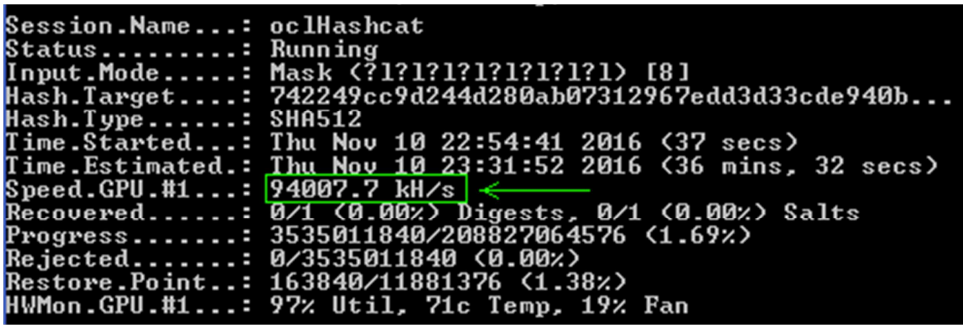
Figura 8: Média de tentativas por segundo do algoritmo SHA1



Fonte: Elaborado pelo autor (2016)

Na Figura 9, esta mesma medição é feita com o algoritmo SHA-512:

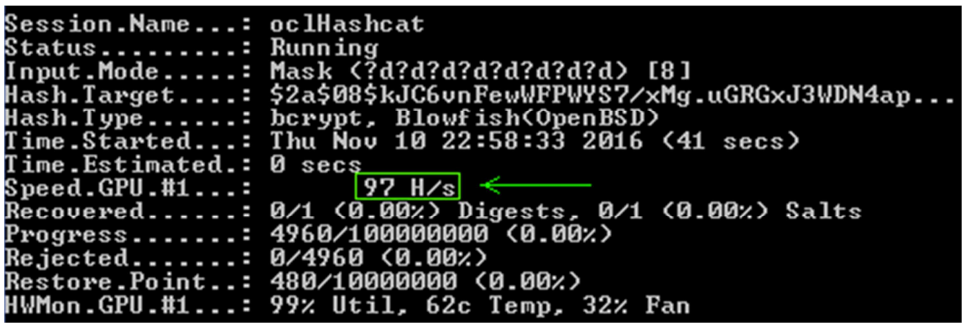
Figura 9: Média de tentativas por segundo do algoritmo SHA-512



Fonte: Elaborado pelo autor (2016)

A medição do algoritmo bcrypt resultou conforme visto na Figura 10. É válido ressaltar a grande diferença em relação aos algoritmos anteriores, em que comparações eram feitas aos milhares ou milhões por segundo; o bcrypt fica abaixo de cem tentativas a cada segundo:

Figura 10: Média de tentativas por segundo do algoritmo bcrypt



Fonte: Elaborado pelo autor (2016)

As estimativas de tempo de quebra foram então calculadas a partir da divisão do número de possibilidades pela quantidade de tentativas por segundo que o software conseguia efetuar. Para se obter esta quantidade de tentativas, foram iniciados testes com cada algoritmo e acompanhada a performance. Ao estabilizar-se a velocidade, foram recolhidas as cinco últimas medições e efetuado um cálculo de média simples entre eles, conforme foi exibido nas Figuras anteriores, e os cálculos resultaram nos dados exibidos na Tabela 3:

Tabela 3: Tentativas de combinação por segundo atingidas

Algoritmo	Tentativas por segundo
MD5	2669540000

SHA1	915980000
SHA-512	94118340
bcrypt	96,6

Fonte: Elaborado pelo autor (2016)

Considerando que o método de ataque foi escolhido foi o de força bruta, duas variáveis serão consideradas nos cenários a seguir: a quantidade de caracteres, ou tamanho da *string*, e o conjunto de caracteres utilizado. A seguir, serão exibidos e explicados os cenários em que os testes foram efetuados, assim como suas justificativas e coletas de resultados.

4.1 CENÁRIO A: IMPACTO DA VARIEDADE DE CARACTERES

Este cenário foi concebido para demonstrar como a variedade de caracteres influencia o tempo de quebra de cada algoritmo. Todas as *strings* utilizadas neste cenário possuem oito caracteres, sendo alterados os conjuntos de caracteres em cada teste, para analisar como a utilização de diferentes caracteres influencia no tempo de quebra de uma senha em cada algoritmo. Na Tabela 4, foram exibidas as *strings* e suas contrapartes em MD5:

Tabela 4: Hashes em MD5, cenário A

String	MD5
50984060	284890737c938228324c0f733990d507
gxuqtkao	3e3d683639f82fd265c9d4fd0af67fa3
7aq8szpm	807d0d99e964c13284b09958f5062d3c
bLZY1H1s	8618ba272e77e71ec9d5773caf414a2d
dA*LCgy0	17b0f00d4095ac5945a3b719631c1c50

Fonte: Elaborado pelo autor (2016)

Já na Tabela 5, as mesmas *strings* se encontram criptografadas em SHA1:

Tabela 5: Hashes em SHA1, cenário A

String	SHA1
50984060	31c7ac57523d3437f0bf5a8f50e7d4441b8ae6f5
gxuqtkao	97aa8bd59fc159a9e90c73a155d435424888b0b5
7aq8szpm	552266cdabaad890d98264e493bfc35424800c76
bLZY1H1s	d6e987eb96050a257bf441df4a24225189d7810f
dA*LCgy0	40d8fed28c66a78c5dd56482951ab7262c59baf0

Fonte: Elaborado pelo autor (2016)

Na Tabela 6, foi utilizado o algoritmo SHA-512 para efetuar a criptografia:

Tabela 6: Hashes em SHA-512, cenário A

String	SHA-512
50984060	1f1edf5fd07253740ba187977ae1e87bdc68374960bfe3131a0b03552ec8e48f585aef9 300e57b805c84ddf8332fe41724ac5368647437c47413523300bd3a18
gxuqtkao	742249cc9d244d280ab07312967edd3d33cde940b995d7df31df34d0042dc7b16354d8 bd9de1a7648cd124bbddd0cdaef43336e0b315251cdb98430e20a1c8c5
7aq8szpm	82dd3a1790dc1adb7afa13b296720e7a0a1eb75f313e60d378d5f36d766199fdce5176f 916f38c8931af8b32d2441d559b309edab35bdd2c41b5a56f09362f53
bLZY1H1s	546a4803f74d6e6ff48e5567a53e6d5ceeba92961081e9be587e0382f8cc6b2ce193cad a0604e67da9a7b3ea3744bd3ed61cb516ecba94ab92b1e4387cc25815
dA*LCgy0	06bb6e4d82efee303a8d5494825fdff6eaa0dff7f462e3e29fee14574462a9a9f325a8d5a 699c347e827cff88d91492c7b5801c4c301c915a8d391c1698bb3c0

Fonte: Elaborado pelo autor (2016)

Finalmente, são exibidos os mesmos *hashes* encriptados utilizando o bcrypt na Tabela 7:

Tabela 7: Hashes em bcrypt, cenário A

String	bcrypt
50984060	$2a$08$kJC6vnFewWFPWYS7/xMg.uGRGxJ3WDN4apztdbOq3YuiaDL38HF6i

gxuqtkao	$2a$08$n2Ot9nzEW.xNDsi6s55m7eGmydTPQIp05Mt94oRXERC1xssJLUnUe
7aq8szpm	$2a$08$L7iuQKIans11xDXDHG83UO1zjKbw.Ea3gy.9.n6Z2pBbVJ763lJz6
bLZY1H1s	$2a$08$YkKwYTuoUARYonBOzgLSguPjyV3/FSPg4nlH3vIgUedHOqkAHlLia
dA*LCgy0	$2a$08$iHUbjmJVeId/tlzQE9Ufm.0vFx97IqJcVyHF7XQ9ts8RI.1ph5uvq

Fonte: Elaborado pelo autor (2016)

Aplicando a equação citada anteriormente para se descobrir a quantidade de possibilidades para cada um dos testes, são gerados os dados apresentados a seguir na Tabela 8:

Tabela 8: Número de combinações possíveis para cada string

String	Comprimento da *string*	Tipo de caracteres	Quantidade de caracteres possíveis	Número de possibilidades
50984060	8	Números	10	1073741824
gxuqtkao	8	Letras minúsculas	26	3,02231E+23
7aq8szpm	8	Letras minúsculas, Números	36	3,24519E+32
bLZY1H1s	8	Letras minúsculas, Letras maiúsculas, Números	62	9,80797E+55
dA*LCgy0	8	Todos	95	6,21654E+85

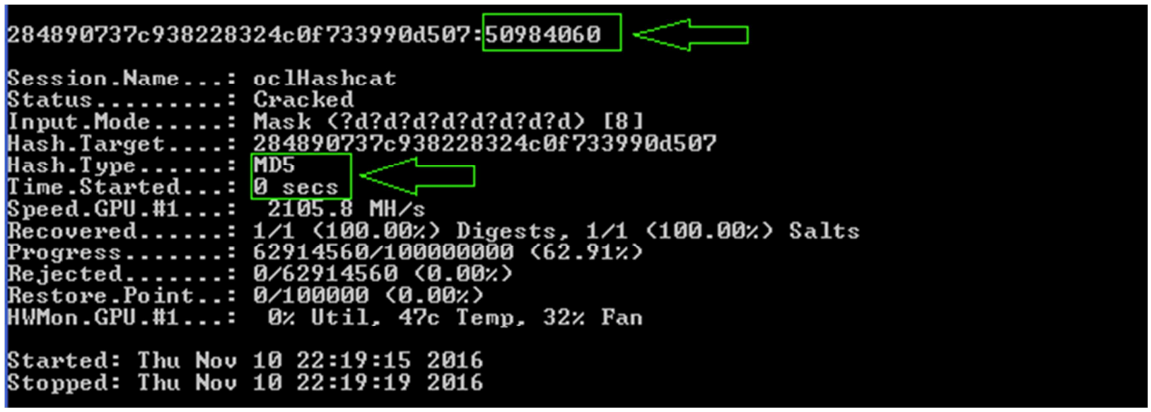
Fonte: Elaborado pelo autor (2016)

4.2 COLETA DE RESULTADOS DO CENÁRIO A

Nos primeiros testes, com conjuntos menores de caracteres, todos os algoritmos exceto o bcrypt foram quebrados corretamente em tempo hábil. Na Figura 11 é verificada a quebra da *string* “50984060” criptografada através do algoritmo MD5 em tempo inferior a um segundo. Nesta figura é exibido um exemplo da saída do oclHashcat quando a quebra é efetuada. Além dos dados já citados na medição de velocidade, a primeira linha exibe o *hash* original seguido de seu texto claro, e

abaixo os horários de início e fim de execução. Este tempo diverge do tempo de quebra pois leva em conta a inicialização do *software*:

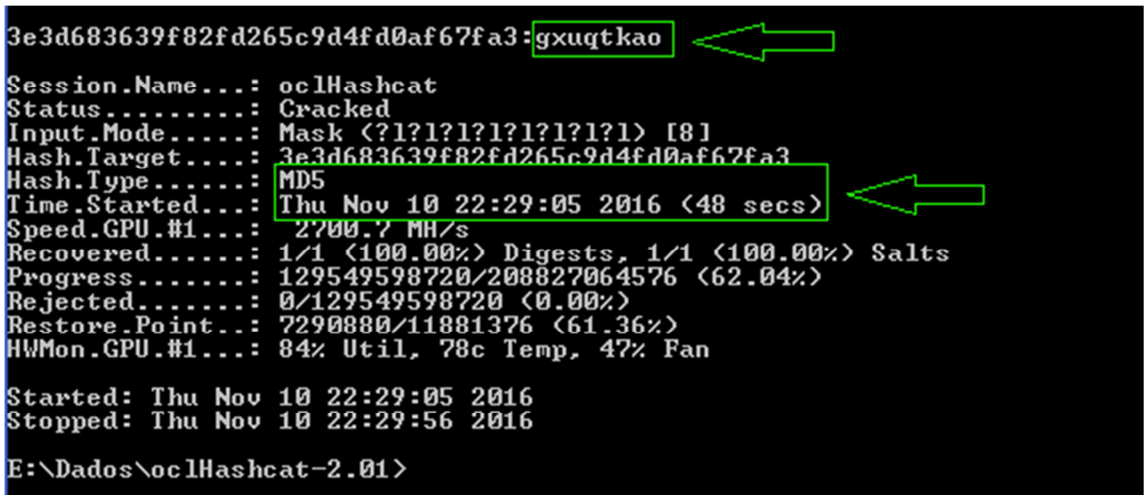
Figura 11: Hash MD5 quebrado com string de oito números



Fonte: Elaborado pelo autor (2016)

Na Figura 12, a *string* “gxuqtkao”, também criptografada utilizando o MD5, foi quebrada em 48 segundos:

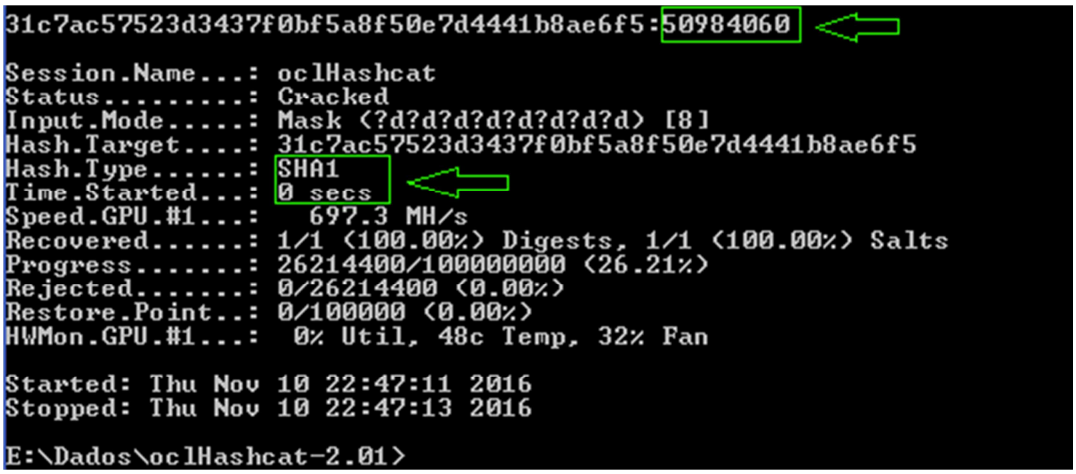
Figura 12: Hash MD5 quebrado com string de oito letras minúsculas



Fonte: Elaborado pelo autor (2016)

Ao executar a quebra da *string* “50984060” criptografada através do SHA1, o sucesso também foi obtido em menos de um segundo, conforme Figura 13:

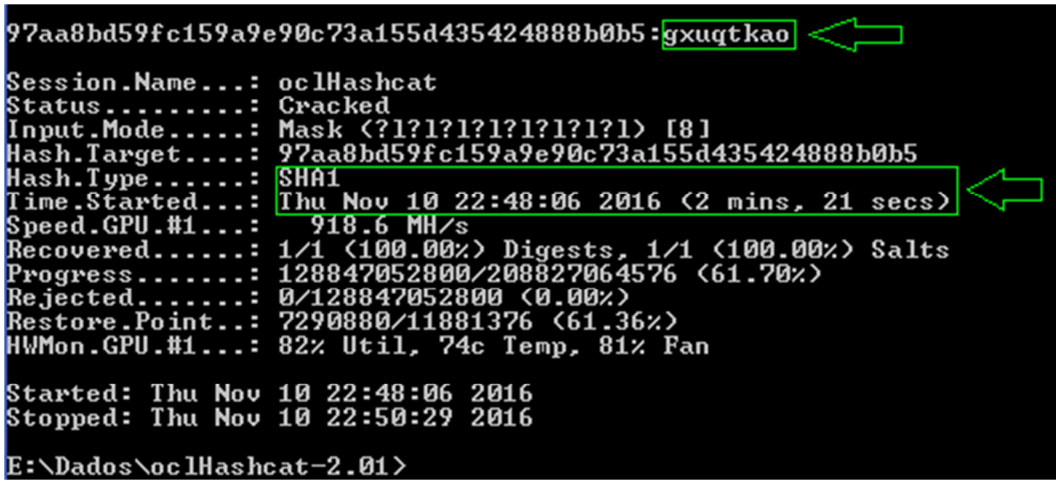
Figura 13: Hash SHA1 quebrado com string de oito números



Fonte: Elaborado pelo autor (2016)

Já a *string* “gxuqtkao” encriptada com o algoritmo SHA1 levou pouco mais de dois minutos para ser revelada, conforme Figura 14.

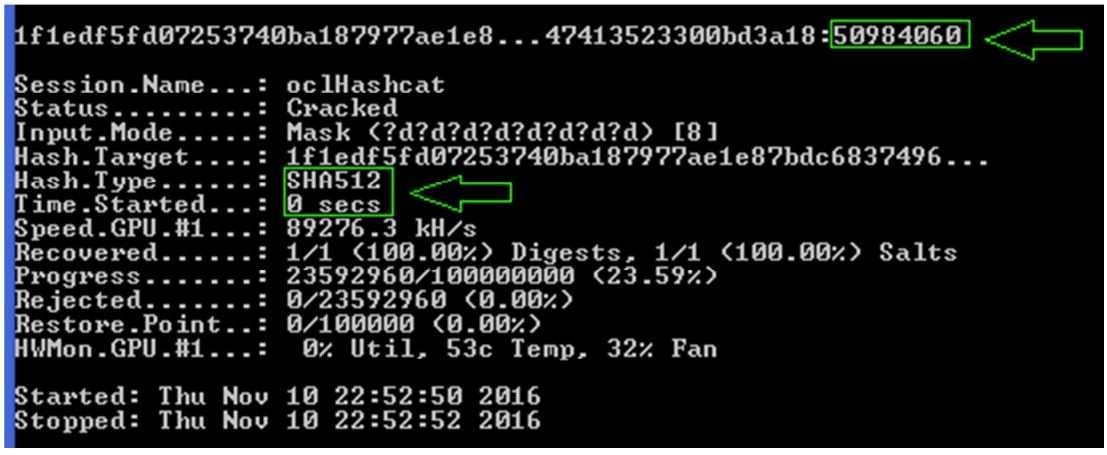
Figura 14: Hash SHA1 quebrado com string de oito letras minúsculas



Fonte: Elaborado pelo autor (2016)

Na Figura 15, observamos a *string* “50984060” também quebrada em menos de um segundo quando utilizado o algoritmo SHA-512.

Figura 15: Hash SHA-512 quebrado com string de oito números



Fonte: Elaborado pelo autor (2016)

Nos testes posteriores com estes algoritmos, assim como todos os realizados com o bcrypt, o tempo estimado se encontrou além do aceitável de finalização, sendo necessário calculá-los manualmente. Tendo as quantidades de combinações possíveis para cada caso exibidas na tabela 3, aplica-se então o cálculo de tempo

estimado, resultando nos dados apresentados na Tabela 9. Para facilitar a visualização, os dados foram transformados de segundos para as unidades de tempo que se adequam a cada caso:

Tabela 9: Tempo estimado em segundos de quebra no cenário A

String	Comprimento da *string*	MD5	SHA1	SHA-512	bcrypt
50984060	8	< 1 segundo	< 1 segundo	1 segundo	12 dias
gxuqtkao	8	78 segundos	4 minutos	37 minutos	69 anos
7aq8szpm	8	18 minutos	51 minutos	8 horas	926 anos
bLZY1H1s	8	23 horas	3 dias	27 dias	71 mil anos
dA*LCgy0	8	29 dias	3 meses	2 anos	795 milhões de anos

Fonte: Elaborado pelo autor (2016)

4.3 CENÁRIO B: IMPACTO DA QUANTIDADE DE CARACTERES

Este cenário foi concebido para se demonstrar, dentro de um conjunto de caracteres pré-determinado, o quanto o tamanho da *string* pode influenciar na quebra de cada um dos algoritmos testados. Foram utilizadas combinações de caracteres que possuam todos os quatro seguintes conjuntos: números, letras minúsculas, letras maiúsculas e caracteres especiais, e em seguida apenas letras minúsculas, ambos simulando dois cenários distintos de senhas reais. Os caracteres especiais seguem as mesmas regras e restrições do cenário A. Na Tabela 10, são exibidas as *strings* a serem utilizadas nos testes do cenário B, e como elas ficaram após a aplicação do algoritmo MD5.

Tabela 10: Hashes em MD5, cenário B

String	MD5
hQfK*d4Q	8e47f9a2d568e6d972b626c48e391895
h(vwMJzK9	980f769baa7442063a3dc97a8a1766e8
)YxDD8zka$	62fde819732386282464af124dd065a8
4Zt8n&&rMcX	9059434a5e7503b85e50f02e81cdd6ef
hpKyF9XERc%X	9bfd33bee023a1fbcf5865ac8db49d48

xigpgbjs	88908608e332c33099e0d96cc0dd5a4d
hdtuyhmtz	18dccd98d09f01df6e8bd6ddba76244b
glrccgwpwk	1adeb63855a8de18993128750314728e
atbumtyorqu	f24beb9d07de40aad64d77e3439f45cb
raqtvjbuvhgu	f3c3b86772aea7845d38dbf518751bf9
pexafxfskvkwl	ea3e27ba8f70e649b630e2942b824a64
bhtavmunmikwqc	a04606561e192931e01f1771f839bd73
gzmugeizsqcbyfq	e689e0e7f493bd5e99257f04dd51241c
xrbzkhzuuiznkrja	6dee499dcab24570fb3f684aec39353c

Fonte: Elaborado pelo autor (2016)

Na Tabela 11, são exibidas as mesmas *strings* sob o algoritmo SHA1.

Tabela 11: Hashes em SHA1, cenário B

String	SHA1
hQfK*d4Q	a4ee90f1d563c0ec7a518e8d12a122f44fc1cb34
h(vwMJzK9	9761ffbbfebdbfad2fa16b505e8842e6c6582161
)YxDD8zka$	06c2345625fdc2b046d0257de21fbb59fcf74ec1
4Zt8n&&rMcX	4170645351e38c4e505bedf9fbd79094b3e28ccb
hpKyF9XERc%X	52c445ad1fefa20fbab9359c590c36f02bc546c0
xigpgbjs	56819a344dec38834a0995eeab8cef41d2fb1a82
hdtuyhmtz	0ed225b6d9cbcda8b171c9fa714f75bae28686f9
glrccgwpwk	3aefcc146d13df8d0bdd3a2c8eec8115cabaf971
atbumtyorqu	b8d6fa6a3fbd6138eb3a1663d631a159469938b6
raqtvjbuvhgu	e134f1aa297bf22053058cc20017ebf5467c1c5e
pexafxfskvkwl	ec0229569688c95ec7390070e3339436895c3acb
bhtavmunmikwqc	a96d64550914dc7e111d3623705bb63a22ae8a05
gzmugeizsqcbyfq	ef701dae18f4234e97e864051897b4d654843f7a
xrbzkhzuuiznkrja	83f3785332579a4fd26c4750bfa5fccd336a7997

Fonte: Elaborado pelo autor (2016)

Os conjuntos de caracteres das *strings* do cenário B foram escolhidos tendo como base dois tipos de senhas utilizados pelos usuários: um utilizando todos os conjuntos de caracteres, conforme sugerido pela maioria dos *websites*, e um conjunto contendo apenas caracteres minúsculos, simulando senhas simples de fácil

memorização que podem ser utilizadas pelos usuários. As *strings* selecionadas foram exibidas nas tabelas anteriores e seguem novamente na Tabela 12, agora criptografadas utilizando o SHA-512.

Tabela 12: Hashes em SHA-512, cenário B

String	SHA-512
hQfK*d4Q	71f32d7e57a7fed1b8136a2b2ff88615725b848ab1e4354f794e390bc06138b72 a3b1d3f22072b965d7eb47042b82ee1923429680937ae06b299fe1595059974
h(vwMJzK9	a99098ac7004022478c2db572c2358eccdccede1003a6bfa450137a23ed09a3 e66b85774f38a4a882ffaf6a82ba023156d8c3d32ad058775cec5254d8a7fdc4c
)YxDD8zka$	1c5af01dbc6f4770fedba4d47bc5b99d2a16801a9e95bada6bdaf01723e0ff3ed 65eea86c2228678269e2290795bed85fcc71fd1b49009ee1cdc324eaef93c56
4Zt8n&&rMcX	8438adda64a4a03e8c314cb6b037504835ca4c5ac99192013bab3aae4a80d93 ce33397a810e9500a6bd5b97bc5b39dbfcbf443e30b7556f2d15aa6fa9f5f993d
hpKyF9XERc%X	df98826d9145b460b4102ec6889fc05b8e0405f8206b20a6db4e1fcae46344f5 10b6b4f8b8657ae87be9c972912af76f5026eb4934f78dec38a5ea00e37b38ba
xigpgbjs	4d6ee1e66364fdb7ce7c1174aa0083af024849683be13292c7b64ca89defc6f6 572cd5f47ce124937fc88d49c878ee8320095da4c2297d0aab757416b4c46e8c
hdtuyhmtz	6af061a11e38b89d4c3c47e79ac7d397576fbb2fa097ffb8e149530262d243c55 fe10a9f69dae49324891899eeac3513a0669e84c8896b5057beea6bcaae37ed
glrccgwpwk	59192c630026afcc5444f537f610eb6ae45cfdc0dbd0ab0bb9ab5e358d5a87f4 48b9888cd74e4930b4dc51bfa03af5a82d98a66a12bbddd153ae11da0a544c91
atbumtyorqu	ba46b968bee23b0bf958849488927d241b83e0628a0dbd4b7b663f1a056ad66 fe20b8ef31b8f0612361376907929a5498d66662bab389671edce93190a3d064c
raqtvjbuvhgu	1a7b5211538eff98a5c5a405e6106a3be94e7e0f81ae4650996e4bee1518a35 18bd4429f8f13d4810bc119a4dbcbf653cd1b782dfa9d278db4d088da557e468f
pexafxfskvkwl	ba81d18fdc2eb783e436448367625a79af96adeebf9e7dab453e62849261b81 6baaf1b898d48087245013f0058363558f9275fc9f4762d08ac87dfa19d5c29de
bhtavmunmikwqc	007fc089dfbc78d22f0e959737d800cf0617bdb71c2ab5a091427219f9f499e4d 665dade7cc28c3115731850a9ebcd3a0b529e85667af2062b0df961fe82de25
gzmugeizsqcbyfq	29d7ae3bf1209d73b650174dcaff3657604457f22d22e1ff8977623b890533e0 951e2483ffaba454d6206b77820941e3515ff04212ef9550cfafd7cbd42539aa
xrbzkhzuuiznkrja	0b149a7f82b0a8f4255627d10dddd1645fb3b22f02bcc5c634f517d87e40742c5 560a6b7d1e36ea4d017cbbdd52a7dfd5c5252d01cf5143e699da31e44a1bbb7

Fonte: Elaborado pelo autor (2016)

Utilizando o algoritmo bcrypt, temos os seguintes resultados:

Tabela 13: Hashes em bcrypt, cenário B

String	bcrypt
hQfK*d4Q	$2a$08$Jw6OvhoI4ZM2v6QjJe2bbOdIzQVMeoB7H6/dx.J.yTuL8Bi7VKvJG
h(vwMJzK9	$2a$08$.FRUUWAhrJgnj/wOa1GxO.Hha552q7nrDKNEfmKHBKuG2dLQ8f4TO
)YxDD8zka$	$2a$08$9yLOQX4Qdyd9duCuMuUHnuw9AscFr3NrFTij6WjV.Wv1rms9qRF8y
4Zt8n&&rMcX	$2a$08$0MYEENS42NGuIovEsL9FA.FKg0EMZDDzSi5paWo0UaEVAj7xkXAWO
hpKyF9XERc%X	$2a$08$FiiWd.ouZOCCTEOYel0UjOhR50mrt/iHEAfQFSw.BnmjpzGNAE5oK
xigpgbjs	$2a$08$6v/kIVkyP2fOjUt1KrSvs.P.jwwDu//bVghShz.ibjKkx9vp3QTB2
hdtuyhmtz	$2a$08$TgJQQUS0S9Xayuyv1.awhesQoNN5f.o.x77lJAOyBca8qCgTx3k5e
glrccgwpwk	$2a$08$.E5nnG5xok3mZZN.YXdpgeT.ESGwN/9p2yPvVclKbZEfkG0rEL7ya
atbumtyorqu	$2a$08$tUSIYSGoG8Fr94OqnAqhnOGi2inCqr1BJnsiFRxkP7uHpr4slRSGK
raqtvjbuvhgu	$2a$08$k2WQR3npOYGng1jLbPD.RuZ/WVJBFPlGkHGPmztod0xcagX5Cndb2
pexafxfskvkwl	$2a$08$rg3OrMVHgCfSSMh3QGpZXuGdvWO6mMwu1H3yjrmz0k.S40hlL5vh2
bhtavmunmikwqc	$2a$08$D/D6XhYrMiHzmIHYm/jGo.zJ.5Ud3thcppeRK11jJlfpOrjUp1weG
gzmugeizsqcbyfq	$2a$08$2S5aTPL5OVuYF/EHxgykbeVHlZh4lg4NjBxYVtOUVHem7JKbAwKLW
xrbzkhzuuiznkrja	$2a$08$nbj0Hw84dhIazlXU06a9suXiQBatBv.dayjowegQGtUije283PhyO

Fonte: Elaborado pelo autor (2016)

Conforme citado no cenário anterior, não é possível simular e angariar os resultados diretamente da saída do oclHashcat. Para se obter os dados, foi necessário calcular-se a quantidade de possibilidades de combinações para cada

string, conforme pode ser visto na Tabela 14. Estes dados serão utilizados para montar mais à frente as tabelas de cálculo estimado de tempo.

Tabela 14: Número de combinações possíveis, cenário B

String	Comprimento da string	Tipo de caracteres	Quantidade de caracteres possíveis	Número de possibilidades
hQfK*d4Q	8	Todos	95	6,6342E+15
h(vwMJzK9	9	Todos	95	6,30249E+17
)YxDD8zka$	10	Todos	95	5,98737E+19
4Zt8n&&rMcX	11	Todos	95	5,688E+21
hpKyF9XERc%X	12	Todos	95	5,4036E+23
xigpgbjs	8	Letras minúsculas	26	2,08827E+11
hdtuyhmtz	9	Letras minúsculas	26	5,4295E+12
glrccgwpwk	10	Letras minúsculas	26	1,41167E+14
atbumtyorqu	11	Letras minúsculas	26	3,67034E+15
raqtvjbuvhgu	12	Letras minúsculas	26	9,5429E+16
pexafxfskvkwl	13	Letras minúsculas	26	2,48115E+18
bhtavmunmikwqc	14	Letras minúsculas	26	6,451E+19
gzmugeizsqcbyfq	15	Letras minúsculas	26	1,67726E+21
xrbzkhzuuiznkrja	16	Letras minúsculas	26	4,36087E+22

Fonte: Elaborado pelo autor (2016)

4.4 COLETA DE RESULTADOS DO CENÁRIO B

Utilizando os dados da Tabela 3, em adição aos dados calculados na Tabela 14, foram então determinados os tempos estimados de quebra para cada combinação de *string* e algoritmo, sendo estes dados apresentados na Tabela 15.

Tabela 15: Tempo estimado de quebra no cenário B

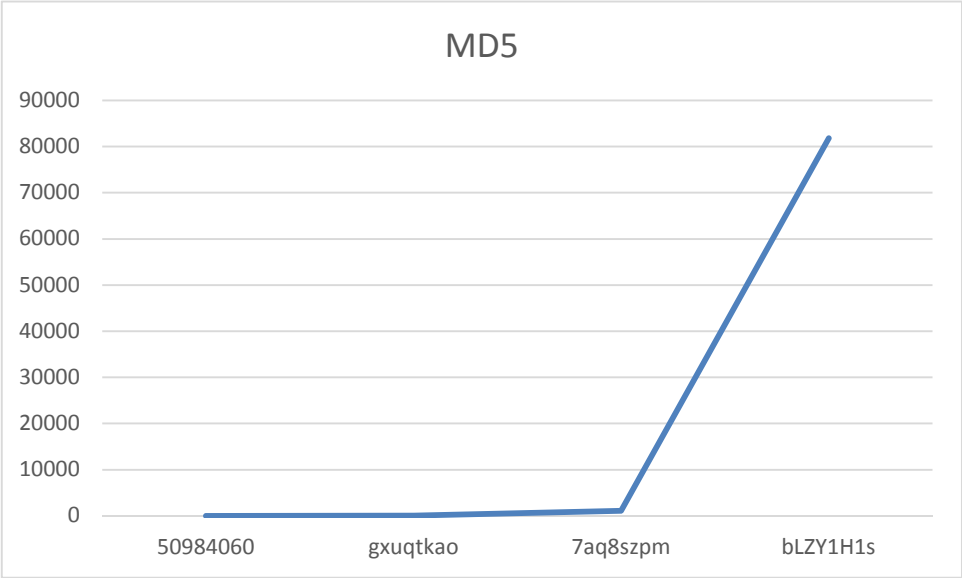
String	Comprimento da string	MD5	SHA1	SHA-512	bcrypt
hQfK*d4Q	8	29 dias	84 dias	2 anos	2 milhões de anos
h(vwMJzK9	9	7 anos	8 mil anos	77 mil anos	206 milhões de anos
)YxDD8zka$	10	711 anos	2 mil anos	20 mil anos	20 bilhões de anos
4Zt8n&&rMcX	11	68 mil anos	197 mil anos	1.9 milhões de anos	2 trilhões de anos
hpKyF9XERc%X	12	6 milhões de anos	19 milhões de anos	182 milhões de anos	177 trilhões de anos
xigpgbjs	8	78 segundos	4 minutos	37 minutos	69 anos
hdtuyhmtz	9	34 minutos	100 minutos	16 horas	1.7 mil anos
glrccgwpwk	10	15 horas	2 dias	17 dias	46 mil anos
atbumtyorqu	11	16 dias	46 dias	1.5 ano	1.2 milhão de anos
raqtvjbuvhgu	12	1.5 ano	3 anos	32 anos	31 milhões de anos
pexafxfskvkwl	13	29 anos	86 anos	836 anos	814 milhões de anos
bhtavmunmikwqc	14	766 anos	2.2 mil anos	22 mil anos	21 bilhões de anos
gzmugeizsqcbyfq	15	20 mil anos	58 mil anos	565 mil anos	551 bilhões de anos
xrbzkhzuuiznkrja	16	518 mil anos	1.5 milhão de anos	14.7 milhão de anos	14.3 trilhões de anos

Fonte: Elaborado pelo autor (2016)

5 RESULTADOS E DISCUSSÕES

Com base nos dados do cenário A, podemos ver que, mantendo o mesmo tamanho de *string*, se torna mais custoso quebrar a criptografia, independentemente do algoritmo utilizado, aumentando-se os conjuntos de caracteres. Na Figura 16 pode-se ver como a utilização de todos os conjuntos aumentou bastante o tempo estimado de quebra no algoritmo MD5.

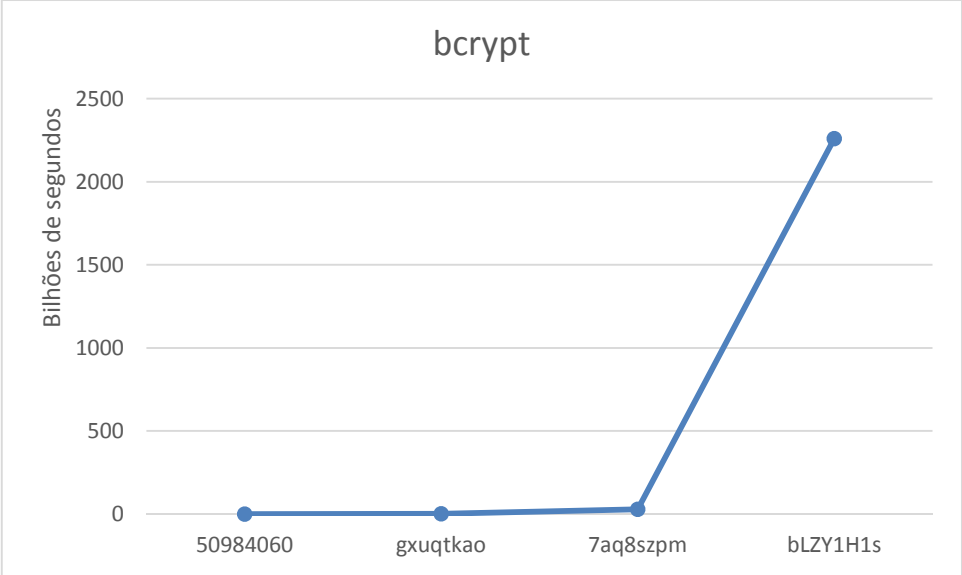
Figura 16: Evolução do tempo de quebra no cenário A, algoritmo MD5



Fonte: Elaborado pelo autor (2016)

Pode-se notar que esta é uma tendência independente analisando-se a evolução sob o algoritmo bcrypt, que segue o mesmo padrão, na Figura 17:

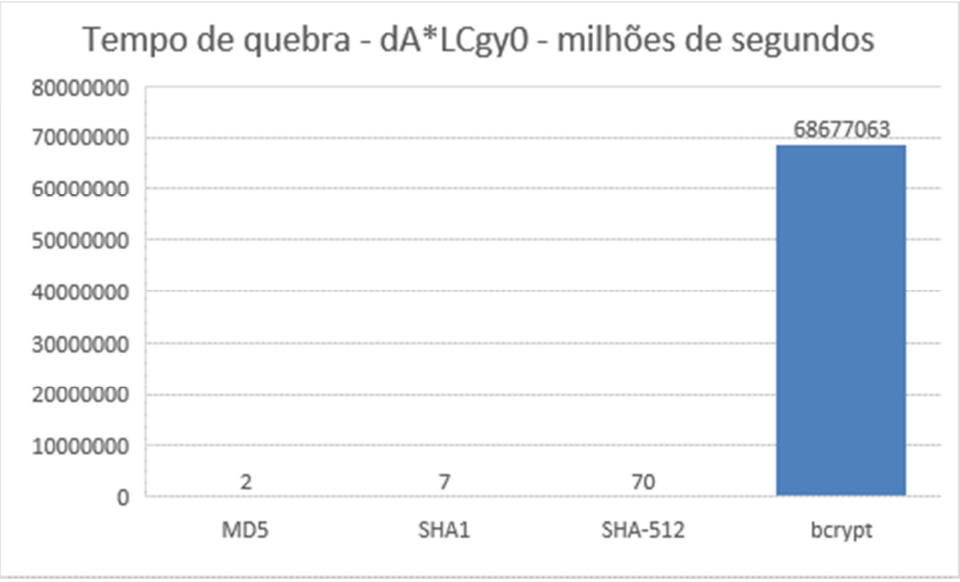
Figura 17: Evolução do tempo de quebra no cenário A, algoritmo bcrypt



Fonte: Elaborado pelo autor (2016)

Observa-se que a utilização de algoritmos legados como SHA1 e MD5 permitiu uma quebra quase instantânea mesmo em um computador doméstico quando utilizadas senhas mais simples, sendo inclusive o SHA-512 também quebrado facilmente com senhas mais fracas. Para base de comparação, o algoritmo bcrypt teve uma força aproximadamente um milhão de vezes maior que o algoritmo SHA-512, o que pode ser comprovado no gráfico exibido na Figura 18.

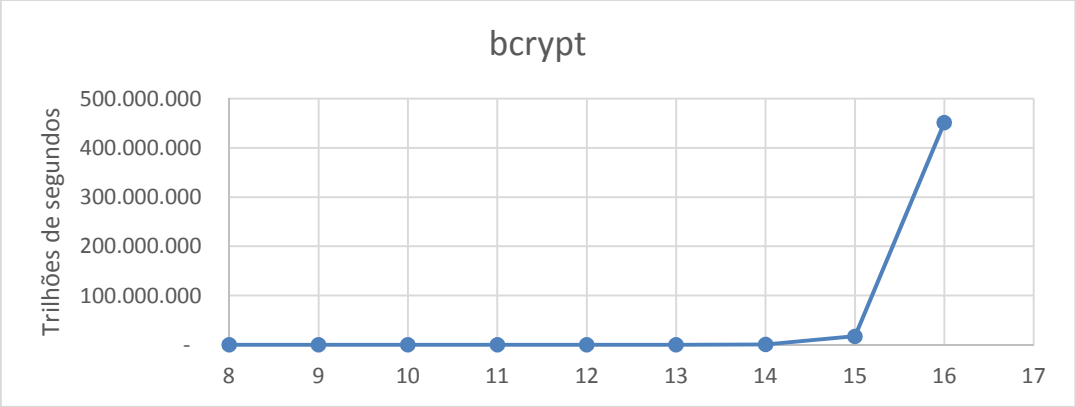
Figura 18: Comparativo de tempos estimados de quebra



Fonte: Elaborado pelo autor (2016)

O tempo estimado de quebra não é real quando utilizados computadores mais potentes ou computação em cluster, e também não indicam a realidade fiel nestes casos: são estimativas calculadas através dos dados brutos obtidos em cada *string*, para cada algoritmo. Apesar de ser apenas uma estimativa, permite que, através destas comparações, seja possível serem comparadas as diversas variáveis que se levam em conta na criptoanálise. Tendo que estes dados devem ser vistos relativamente, e não absolutamente, vemos agora na Figura 18 como a adição de caracteres alterou o tempo estimado no cenário B para as *strings* compostas apenas por caracteres minúsculos, utilizando o algoritmo bcrypt como base.

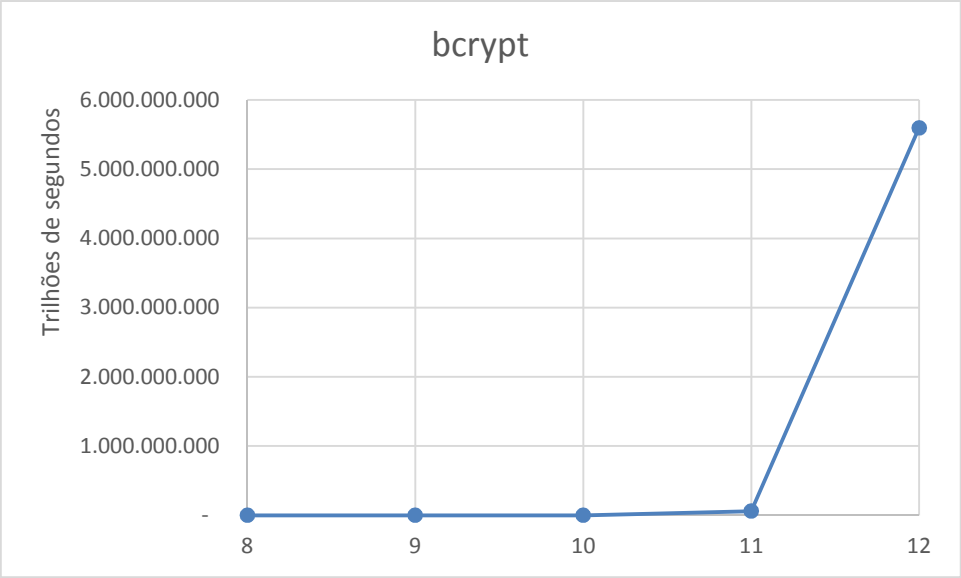
Figura 19: Estimativa de quebra em segundos, somente caracteres minúsculos



Fonte: Elaborado pelo autor (2016)

É verificado o crescimento exponencial conforme o número de caracteres aumenta, resultado da fórmula de quantidade de combinações, que utiliza exatamente a quantidade de caracteres como expoente. Na Figura 20, vemos como esta tendência se repete mesmo com *strings* utilizando todos os conjuntos de caracteres.

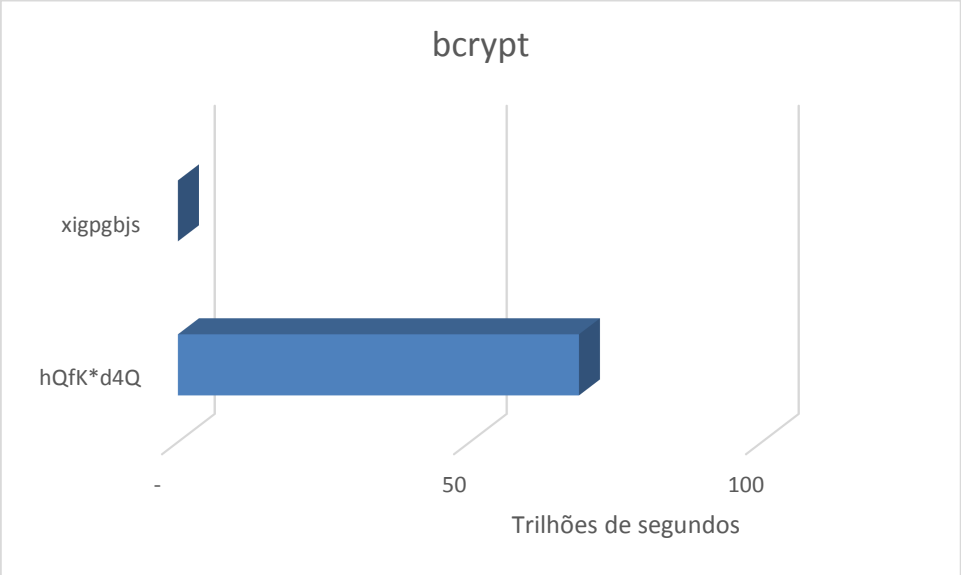
Figura 20: Estimativa de quebra em segundos, todos conjuntos de caracteres



Fonte: Elaborado pelo autor (2016)

Verifica-se também que o uso de todos os conjuntos de caracteres, quando comparado apenas com o uso de letras minúsculas, também irá necessitar de mais processamento para sua quebra e sua segurança, mesmo comparativa cresce exponencialmente, conforme pode ser visto na Figura 21, em que são comparados tempos estimados de quebra de *strings* de oito caracteres no cenário B sob o algoritmo bcrypt.

Figura 21: Estimativa de quebra em segundos, comparativo de oito caracteres



Fonte: Elaborado pelo autor (2016)

Com oito caracteres, o uso de todos os conjuntos de caracteres permite aproximadamente 31 mil vezes mais combinações quando comparado a apenas caracteres minúsculos. Quando se utilizam doze caracteres de comprimento, apenas quatro a mais, o uso do maior conjunto de caracteres produz aproximadamente cinco milhões de vezes mais combinações do que apenas doze letras minúsculas.

6 CONSIDERAÇÕES FINAIS

Os ataques a bancos de dados de empresas acontecem a todo momento e mesmo empresas com ótima reputação no ramo da tecnologia acabam por sofrer vazamentos de dados, então a criptografia se faz necessária em quaisquer bancos de dados que contenham informações sensíveis.

Por mais que as técnicas de criptografia evoluam e consigam teoricamente armazenar os dados seguramente, diariamente novas vulnerabilidades e falhas são encontradas, e algoritmos que são padrão em um dia podem já não ser mais em outro. A criptoanálise avança juntamente com a criptografia, e em conjunto com o rápido avanço das tecnologias computacionais, fazem algoritmos ficarem obsoletos e perderem suas capacidades de manter seguros os dados armazenados, revelando senhas facilmente para atacantes preparados.

Conforme pôde ser visto na simulação realizada, algoritmos específicos para armazenamento de senha são a melhor opção para se manter a segurança. Uma senha de oito caracteres minúsculos, por exemplo, que leva menos de dois minutos para ser quebrada via força bruta em um algoritmo MD5, ou trinta e sete minutos utilizando a segurança máxima do SHA-2, com 512 bits, levaria quase setenta anos para ser quebrada utilizando o bcrypt. A utilização destes algoritmos pensados exclusivamente para se resistir a ataques massivos, em conjunto com técnicas robustas de programação, consegue por si só manter uma segurança mais alta e possivelmente inviabilizar um ataque massivo via força bruta.

Lembrando que os testes foram aplicados via força bruta – ataques de dicionário e *rainbow tables* são mais efetivos por já levarem em conta a previsibilidade humana, porém para efeitos de teste de força de algoritmos adicionariam subjetividade e necessitariam de conhecimento mais profundo sobre a geração de listas de palavras ou de onde encontrá-las, o que fugiria do escopo desta dissertação. Os testes também foram feitos em um computador doméstico, não se assemelhando aos ataques reais, muitas vezes realizados em *clusters*, superando em várias vezes o poder de processamento utilizado aqui.

Porém, seguindo a Lei de Moore (já revisada por ele próprio em luz do *plateau* cada vez mais próximo de ser atingido), que prevê a duplicação da capacidade

computacional a cada dezoito meses, e que a placa utilizada era topo de linha dez anos antes da escrita desta monografia, e o *overhead* gerado pelo paralelismo do processamento, quebras estimadas em milhões de anos ou até mais ainda não seriam viáveis mesmo nos parques computacionais mais avançados disponíveis aos atacantes.

O que os usuários devem sempre ter em mente é que os algoritmos e práticas mais seguras nem sempre são utilizadas nos *websites*, e não há como saber isto na maioria dos casos exceto quando divulgado publicamente ou um vazamento acontecer. Sistemas mais antigos ainda utilizam por vezes algoritmos legados, fazendo com que uma vez que o atacante esteja de posse dos dados, ele consiga extrair facilmente as senhas. Mesmo assim, os usuários ainda têm opções para se manter seguro e aumentar a chance de não ter seus dados revelados no caso de um ataque: fortalecer suas senhas. Apesar de todos os algoritmos terem resultados distintos, uma tendência é que quanto maior o comprimento do texto original e mais conjuntos de caracteres for utilizado, mais tempo demorará para ser decriptado o *hash*.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BIHAM, E. et al. Collisions of SHA-0 and reduced SHA-1. **Annual International Conference on the Theory and Applications of Cryptographic Techniques**. Aarhus, Dinamarca: Springer Berlin Heidelberg, 2005. 22p. Disponível em: <https://www.iacr.org/archive/ eurocrypt2005/34940036/34940036.pdf>. Acesso em: 05 nov. 2016.

CHAUBAUD, F.; JOUX, A. Differential collisions in SHA-0. **Annual International Cryptology Conference**. Santa Barbara, CA, EUA: Springer Berlin Heidelberg, 1998. 16p. Disponível em: <https:// www.iacr.org/archive/eurocrypt2005/34940036/ 34940036.pdf>. Acesso em: 05 nov. 2016.

CHAVES, R. et al. Improving SHA-2 hardware implementations, **Workshop on cryptographic hardware and embedded systems 2006**, p. 298-310. Yokohama, Japão: Springer Berlin Heidelberg, 2006. Disponível em: <http://citeseerx.ist.psu.edu/ viewdoc/download?doi=10.1.1.65.5992&rep=rep1&type=pdf>. Acesso em: 05 nov. 2016.

CHRISTENSEN, C. Polish mathematicians finding patterns in Enigma messages. **The Mathematics Magazine**, v. 80, n.4, p. 274-273. Washington, DC, EUA: Mathematical Association of America, 2007. Disponível em: <http://www. ingeniousmathstat.org/sites/default/files/pdf/upload_library/22/Allendoerfer/christense n247.pdf>. Acesso em: 02 nov. 2016.

CONSTANTIN, L. **Ashley Madison coding blunder made over 11 million passwords easy to crack.** 2015. Disponível em: <http://www.pcworld.com/ article/2982919/security/ashley-madison-coding-blunder-made-over-11-million-passwords-easy-to-crack.html>. Acesso em: 02 nov. 2016.

ELECTRONIC FRONTIER FOUNDATION – EFF. **"EFF DES Cracker" machine brings honesty to crypto debate**. San Francisco, CA, EUA, 1998. Disponível em: <https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/ 19980716_eff_descracker_pressrel.html>. Acesso em: 04 nov. 2016.

ERTAUL, L. et al. Implementation and performance analysis of PBKDF2, bcrypt, scrypt algorithms. **Proceedings of the International Conference on Wireless Networks (ICWN)**. The steering comitee of the world congress in computer science, computer engineering and applied computing. Las Vegas, NV, EUA: WorldComp, 2016. 7p. Disponível em: <http://www.mcs.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf> . Acesso em: 05 nov. 2016.

GOODE, W. J.; HATT, P. K. Métodos em pesquisa social. In: FACHIN, O. **Fundamentos da metodologia**. 5. ed. São Paulo: Editora Saraiva, 2006. 210p.

GOODIN, D. **Lessons learned from cracking 4,000 Ashley Madison passwords.** 2015. Disponível em: <http://arstechnica.com/security/2015/08/cracking-all-hacked-ashley-madison-passwords-could-take-a-lifetime/>. Acesso em: 02 nov. 2016.

GOYAL, V. et al. CompChall: Addressing password guessing attacks. **International Conference on Information Technology**: Coding and Computing (ITCC'05) Volume II, v. 1, p. 739-744. Las Vegas, NV, EUA: IEEE, 2005. Disponível em: <https://arxiv.org/ftp/arxiv/papers/1111/1111.3753.pdf>. Acesso em: 06 nov. 2016.

HISTORY. **History of WW2**: Code Breaking. 2015. Disponível em: <http://www.history.co.uk/study-topics/history-of-ww2/code-breaking>. Acesso em: 20 nov. 2016.

ISLAM, E.; AZAD, S.; PATHAN, A. S. K. **Practical cryptography**: Algorithms and implementations using C++. Boca Raton, FL, EUA: CRC Press, 2014. 358p. Disponível em: <http://eureka.com.ve/libros/cryptography/CRC.Press.Crypto.Nov.2014.ISBN.1482228890.pdf>. Acesso em: 04 nov. 2016.

IVRY, A. **Al-Kindi's metaphysics**: a translation of Ya'qub ibn Ishaq al-Kindi's Treatise "On First Philosophy" (fi al-Falsafah al-Ula). Albany, NY, EUA: State University of New York Press. 1974. 207p**.** Disponível em: <http://www.muslimphilosophy.com/books/kindi-met.pdf>. Acesso em: 02 nov. 2016.

KAHN, D. **The codebreakers**: the story of secret writing. New York, NY, EUA: The New American Library. 1973. 473p**.** Disponível em: <http://mindguruindia.com/wp-content/uploads/2014/06/MP069_The-CodeBreakers.pdf>. Acesso: em 02 nov. 2016.

KREBS, Brian. **Crooks steal, sell Verizon enterprise customer data.** 2016. Disponível em: <http://krebsonsecurity.com/2016/03/crooks-steal-sell-verizon-enterprise-customer-data/>. Acesso em: 02 nov. 2016.

LE, D. et al. Parallel AES algorithm for fast data encryption on GPU. **2010 2nd International Conference on Computer Engineering and Technology (ICCET),** v. 6, p. 1-6. Chengdu, China: IEEE, 2010. 15p. Disponível em: <http://xa.yimg.com/kq/groups/13354653/1984625829/name/naveed1.pdf>. Acesso em: 06 nov. 2016.

MALVONI, K; KNEZOVIC, J. Are your passwords safe: Energy-efficient bcrypt cracking with low-cost parallel hardware. **8th USENIX Workshop on Offensive Technologies**. 2014. 7p**.** Disponível em: <https://www.usenix.org/system/files/conference/woot14/woot14-malvoni.pdf>. Acesso em: 03 nov. 2016.

MORRIS, R; THOMPSON, K. Password security: A case history. **Communications of the ACM**, v. 22, n. 11, p. 594-597, 1979. Disponível em: <https://info.aiaa.org/tac/isg/SOFTC/Public%20Documents/Technical%20Working%20Groups/Cybersecurity/Early_Documents_from_Rao/Password%20Security%20A%20case%20Study.pdf>. Acesso em: 03 nov. 2016.

NARAYANAN, A.; SHMATIKOV, V. Fast dictionary attacks on passwords using time-space tradeoff. **Proceedings of the 12th ACM conference on computer and communications security**, p. 364-372. Alexandria, VA, EUA: Association for Computing Machinery, 2005. Disponível em: <http://www.profsandhu.com/cs6393_s13/p364-narayanan-2005.pdf>. Acesso em: 06 nov. 2016.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY – NIST. **NIST releases SHA-3 cryptographic hash standard**. EUA, 2015. Disponível em: <https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard>. Acesso em: 05 nov. 2016.

NEODATA. **HSM nShield Connect+**. 2016. Disponível em: <http://n1.neodata.com.uy/?project=appliance-de-seguridad-en-red>. Acesso em: 20 nov. 2016.

NIELS, P; MAZIERES, D. A future-adaptable password scheme. **Proceedings of the FREENIX Track**: USENIX Annual Technical Conference. Monterey, CA, EUA, 1999. 13p**.** Disponível em: <https://www.usenix.org/legacy/events/usenix99/full_papers/provos/provos.pdf>. Acesso em: 03 nov. 2016.

NVIDIA. **nVidia Titan X**: The ultimate**.** 2016. Disponível em: <http://www.geforce.com/hardware/10series/titan-x-pascal>. Acesso em: 06 nov. 2016.

NVIDIA. **nVidia Titan X**: The world's ultimate graphics card, available August 2nd. 2016. Disponível em: <http://www.geforce.com/whats-new/articles/nvidia-titan-x-pascal-available-august-2nd>. Acesso em: 06 nov. 2016

OECHSLIN, P. **Making a faster cryptanalytic time-memory trade-off**. Santa Barbara, CA, EUA: Springer Berlin Heidelberg, 2003. 15p. Disponível em: <https://infoscience.epfl.ch/record/99512/files/Oech03.pdf>. Acesso em: 06 nov. 2016.

OLEKSIAK, W. **The hacker who saved thirty million lives**. 2003. Disponível em: <http://culture.pl/en/article/the-hacker-who-saved-thirty-million-lives>. Acesso em: 20 nov. 2016.

PAYMENT CARD INDUSTRY SECURITY STANDARDS COUNCIL – PCI-SSC. **Payment Card Industry (PCI) Data Security Standard**: Requirements and security assessment procedures. 2016. 139p. Disponível em: <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf>. Acesso em: 02 nov. 2016.

PATON, C. et al. **Ashley Madison hack as it happened**: Who is named in cheating site's big data leak?. 2015. Disponível em: <http://www.ibtimes.co.uk/ashley-madison-hack-live-who-is-named-in-cheating-websites-leaked-data-1516104>. Acesso em: 20 nov. 2016.

PHONG, P. H. et al. Password recovery for encrypted ZIP archives using GPUs. **Proceedings of the 2010 Symposium on Information and Communication Technology**, p. 28-33. Hanói, Vietnã: Association for Computing Machinery, 2010. Disponível em: <http://s3.amazonaws.com/academia.edu.documents/45491355/Password_recovery_for_encrypted_ZIP_arch20160509-22369-l5dfxf.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1479230181&Signature=LunN5T%2BEPPhe8vFHwbAnfuuBx1U%3D&response-content-disposition=inline%3B%20filename%3DPassword_recovery_for_encrypted_ZIP_arch.pdf>. Acesso em: 06 nov. 2016.

REUTERS. **Yahoo confirms at least 500 million accounts were hacked.** 2016. Disponível em: <http://fortune.com/2016/09/22/yahoo-hack/>. Acesso em: 02 nov. 2016.

RIJMEN, V; OSWALD, E. Update on SHA-1. ***Cryptographers' Track at the RSA Conference***. San Francisco, CA, EUA: Springer Berlin Heidelberg, 2005. 15p. Disponível em: <https://pdfs.semanticscholar.org/1cef/998b16f0bce7d5e7477d8c2155ef6e878485.pdf>. Acesso em: 05 nov. 2016.

RIVEST, R. **The MD5 message-digest algorithm**. Cambridge, MA, EUA: MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. 21p**.** Disponível em: <https://tools.ietf.org/html/rfc1321>. Acesso em: 04 nov. 2016.

SALOMAA, A. **Public-key criptography**. Turku, Finlândia: Springer Science & Business Media, 2013. 273p.

SCHRAMM, K.; LEANDER, G.; FELKE, P.; PAAR, C. A collision-attack on AES. Combining side channel and differential attack. **International Workshop on Cryptographic Hardware and Embedded Systems**. Bochum, Alemanha, 2004. 14p**.** Disponível em: <http://www.ei.rub.de/media/crypto/veroeffentlichungen/2011/01/28/aes_collisions.pdf>. Acesso em: 04 nov. 2016.

SEARCH FOR EXTRATERRESTRIAL INTELLIGENCE – SETI. **CPU performance**. Disponível em: <https://setiathome.berkeley.edu/cpu_list.php>. Acesso em: 06 nov. 2016.

SHIREY, R. **Internet Security Glossary**, Version 2. Fremont, CA, EUA: IETF, 2007. 365p. Disponível em: <https://tools.ietf.org/html/rfc4949>. Acesso em: 06 nov. 2016.

SINGH, S. **The code book**: The secret history of codes and codebreaking. Londres, Inglaterra: Fourth Estate, 1999. 416p.

SKLAVOS, N.; KOUFOPAVLOU, O. Implementation of the SHA-2 hash family standard using FPGAs. **The Journal of Supercomputing**, v. 31, p. 227-248. EUA: Springer, 2005. Disponível em: <https://www.researchgate.net/profile/Nicolas_Sklavos/publication/220358511_Implementation_of_the_SHA-2_hash_family_standard_using_FPGAs/links/557219af08ae75215868139c.pdf>. Acesso em: 05 nov. 2016.

STEVENS, M. Cryptanalysis of MD5 & SHA-1. **Proceedings of the special-purpose hardware for attacking cryptographic systems**. Washington, DC, EUA, 2012. 37p**.** Disponível em: <http://oai.cwi.nl/oai/asset/21021/21021B.pdf>. Acesso em: 03 nov. 2016.

STEVENS, M. **Fast collision attack on MD5**.Eindhoven, Holanda, 2006: IACR Cryptology ePrint Archive. 13p**.** Disponível em: <http://crppit.epfl.ch/documentation/Hash_Function/Examples/Code_Project/Documentation/104.pdf>. Acesso em: 03 nov. 2016.

STEVENS, M. et al. Freestart collision for full SHA-1. **Annual International Conference on the Theory and Applications of Cryptographic Techniques**, p. 459-483 Viena, Áustria: Springer Berlin Heidelberg, 2016. Disponível em: <https://eprint.iacr.org/ 2015/967.pdf>. Acesso em: 05 nov. 2016.

VERIZON. **2015 Data breach investigations report.** 2015. Disponível em: <http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigation-report_2015_en_xg.pdf>. Acesso em: 02 nov. 2016.

VERIZON. **2016 Data breach investigations report.** 2016. Disponível em: <http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf>. Acesso em: 02 nov. 2016.

WANG, X.; YIN, L. Y.; YU, H. Finding collisions in the full SHA-1. **Annual International Cryptology Conference**, p. 17-36. Santa Barbara, CA, EUA: Springer Berlin Heidelberg, 2005. Disponível em: <https://pdfs.semanticscholar.org/abc6/3b4e03c548830092b056a317bd7ac51e5a47.pdf>. Acesso em: 05 nov. 2016.

YAN, J. J. et al. Password memorability and security: empirical results. **IEEE Security & Privacy**, v. 2, n. 5, p. 25-31. EUA: IEEE, 2004. Disponível em: <http://www.lancaster.ac.uk/people/yanj2/jyan_ieee_pwd.pdf>. Acesso em: 06 nov. 2016.