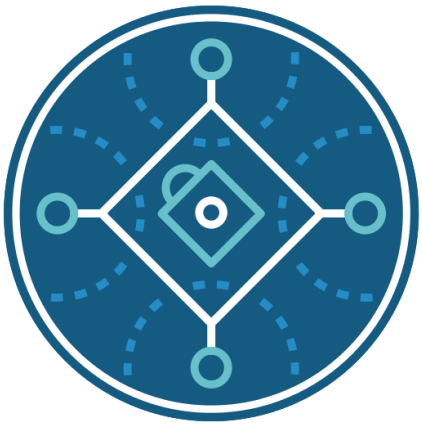
SBSEG15

MINICURSOS

XV SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS

9 A 12 DE NOVEMBRO | FLORIANÓPOLIS/SC



SBSEG15

XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Florianópolis SC, 9 a 12 de novembro de 2015

MINICURSOS

Sociedade Brasileira de Computação – SBC

Organizadores
Eduardo Souto, UFAM
Michelle Wangham, UNIVALI
Joni da Silva Fraga, UFSC

Realização
Universidade Federal de Santa Catarina – UFSC
Universidade do Vale do Itajaí – UNIVALI

Promoção
Sociedade Brasileira de Computação – SBC

Capa: Luciana Soares Fernandes
Editoração: Carlos Maziero, UFPR

Dados Internacionais de Catalogação na Publicação (CIP)

S57 Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (15. : 2015 : Florianópolis, SC)
XV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais : minicursos, 9 a 12 de novembro de 2015 / Sociedade Brasileira de Computação ; Organizadores, Eduardo Souto, Michelle Wangham, Joni da Silva Fraga. — Porto Alegre, RS : Sociedade Brasileira de Computação, 2015.
v, 183 p. il. ; 23 cm.

ISBN: 978-85-7669-304-8

1. Ciência da computação. 2. Informática. 3. Segurança da informação. 4. Segurança de sistemas. I. Souto, Eduardo. II. Wangham, Michelle. III. Fraga, Joni da Silva. V. Título. VI. Título: Minicursos [do] XV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais.

CDU 004(082)

Sociedade Brasileira de Computação – SBC

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

Mensagem do Coordenador de Minicursos

Este livro apresenta a seleção de Minicursos da 15a edição do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg), realizado em Florianópolis -SC, entre os dias 09 a 12 de novembro de 2015. As sessões de minicursos do evento representam uma oportunidade para acadêmicos e profissionais se aprofundarem em temas relevantes e atuais da área, e que normalmente não são cobertos em grades curriculares das universidades. A reconhecida qualidade dos textos produzidos pelos autores dos Minicursos tem elevado estes textos à categoria de documentos de referência para trabalhos acadêmicos e formação complementar de estudantes, pesquisadores e profissionais.

Em 2015, foram submetidas 10 propostas, das quais 4 foram selecionadas para publicação e apresentação, representando assim uma taxa de aceitação de 40%. O comitê de avaliação dos minicursos foi composto por 16 renomados pesquisadores, que desempenharam um excelente trabalho no processo de elaboração dos pareceres para seleção das propostas. Esta edição reúne, portanto, quatro capítulos, produzidos pelos autores das propostas aceitas. No Capítulo 1, os autores discutem a utilização programática de criptografia por desenvolvedores de software com pouca ou nenhuma experiência em segurança da informação e criptografia. O texto mostra aos programadores de software, por meio de exemplos reais e trechos de código, os bons e maus usos da criptografia. O Capítulo 2 discorre sobre o ambiente de aplicações de Mobile Crowd Sensing, focando principalmente nas questões relacionadas à privacidade, segurança e a confiabilidade das informações. No Capítulo 3, os autores descrevem os aspectos que ajudam a tornar uma implementação de criptografia em software eficiente e segura. E, finalmente, o Capítulo 4 apresenta as principais técnicas e trabalhos relacionados à detecção de spam.

Como Coordenador de Minicursos, gostaria de agradecer a todos envolvidos na produção deste livro. Primeiramente aos coordenadores gerais do SBSEG 2015, Michelle Wangham (UNIVALI) e Joni Fraga (UFSC), pelo convite para a coordenação de minicursos, além de todo o suporte necessário para a realização do evento. Agradeço também a todos os membros do comitê de avaliação por terem aceitado o meu convite e dedicado grande esforço para produzir as revisões de alta qualidade para todos os trabalhos submetidos. Por fim, um agradecimento especial a todos os autores que prestigiaram a iniciativa de realização dos minicursos do SBSEG, submetendo propostas que refletem os resultados das suas pesquisas, estudos e projetos acadêmicos.

Eduardo Souto (Universidade Federal do Amazonas)
Coordenador de Minicursos do SBSEG 2015

Comitê de Avaliação de Minicursos

Alberto Schaeffer-Filho, UFRGS
Aldri dos Santos, UFPR
André Santos. UECE
Antonio Rocha, UFF
Carlos Westphall, UFSC
Célio Vinicius Neves de Albuquerque, UFF
Eduardo Feitosa, UFAM
Eduardo Souto, UFAM
Joaquim Celestino Júnior, UECE
Julio Hernandez, UNICAMP
Lau Cheuk Lung, UFSC
Michelle Wangham, UNIVALI
Paulo André da Silva Gonçalves, UFPE
Raul Ceretta Nunes UFSM
Ricardo Dahab, UNICAMP
Rossana Andrade, UFC

Sumário

Mensagens dos organizadores iv

Comitês v

1 *Introdução à Criptografia para Programadores: Evitando Maus Usos.*
Alexandre Braga e Ricardo Dahab 1

2 *Segurança em Mobile Crowd Sensing.*
Joélisson Joaquim de V. Laurido e Eduardo Luzeiro Feitosa 51

3 *Implementação Eficiente e Segura de Algoritmos Criptográficos.*
Armando Faz Hernández, Roberto Cabral, Diego F. Aranha, Julio López 93

4 *Abordagens para Detecção de Spam de E-mail.*
Cleber K. Olivo, Altair O. Santin e Luiz Eduardo S. Oliveira 141

Capítulo 1

Introdução à Criptografia para Programadores: Evitando Maus Usos da Criptografia em Sistemas de Software

Alexandre Braga e Ricardo Dahab

Abstract

Studies have shown that vulnerabilities in cryptographic software are generally caused by implementation defects and mismanagement of cryptographic parameters. In addition, we see the recurring presence of several cryptographic bad practices in various software and mobile applications in particular. Possibly, these vulnerabilities were included unintentionally by inexperienced programmers without expert support. Along this vein, this short course addresses the programmatic use of cryptography by software developers with little or no experience in information security and cryptography. The material is introductory and aims to show software developers, through real examples and code snippets, the good and bad uses of cryptography and thus facilitate further improvements in future studies.

Resumo

Estudos têm revelado que vulnerabilidades em softwares criptográficos são causadas em geral por defeitos de implementação e pela má gestão de parâmetros criptográficos. Além disso, percebe-se a presença recorrente de diversas práticas ruins de criptografia em softwares diversos e aplicativos móveis em particular. Possivelmente, estas vulnerabilidades foram incluídas sem intenção por programadores inexperientes e sem apoio de especialistas. Desta forma, este minicurso aborda a utilização programática de criptografia por desenvolvedores de software com pouca ou nenhuma experiência em segurança da informação e criptografia. O material é introdutório e tem o objetivo de mostrar aos programadores de software, por meio de exemplos reais e trechos de código, os bons e maus usos da criptografia e, assim, facilitar o aprofundamento em estudos futuros.

1.1. Introdução

Atualmente, a proliferação de smartphones e tablets e o advento da computação em nuvem estão mudando a forma como o software é desenvolvido e distribuído. Se por um lado há uma pulverização do esforço de construção de aplicativos, por outro surge, em escala sem precedentes, uma grande quantidade de aplicativos móveis disponíveis para aquisição em qualquer lugar e prontos para uso a qualquer momento.

Neste universo de aplicativos móveis sempre conectados, o uso de funções de segurança baseadas em técnicas criptográficas é cada vez maior. Observa-se que a utilização de criptografia tem aumentado muito, seja em termos do volume de dados criptografados (por exemplo, os smartphones mais modernos possuem sistemas de arquivos cifrados como uma opção padrão), seja em relação à quantidade de aplicativos com serviços criptográficos incluídos em seu funcionamento (por exemplo, uma loja de aplicativos pode vir a possuir centenas de aplicativos voltados a proteção criptográfica de dados: veja https://play.google.com/store/search?q=cryptography&c=apps).

Além dos casos de uso já tradicionais do software criptográfico autônomo, tais como encriptação/decriptação e assinatura/verificação, há diversas situações novas, intrinsecamente relacionadas à lógica da aplicação e do negócio, tornando cada vez mais diversificado o universo de ameaças ao software criptográfico moderno e exigindo cada vez mais do programador comum, geralmente não especializado em criptografia.

Este texto aborda a utilização programática de criptografia por desenvolvedores de software com pouca ou nenhuma experiência em segurança da informação e criptografia. O texto é introdutório e tem o objetivo de mostrar aos programadores de software, por meio de exemplos reais e trechos de código, os bons e maus usos da criptografia e, assim, facilitar o aprofundamento em estudos futuros.

Os objetivos do curso são os seguintes: *(i)* apresentar conceitos básicos de criptografia para programadores iniciantes em segurança da informação; *(ii)* mostrar como esses conceitos são utilizados em plataformas de desenvolvimento de software modernas; e *(iii)* ilustrar maus usos de criptografia comumente cometidos durante o desenvolvimento de software. Vale ainda ressaltar que o texto incentiva a utilização correta de implementações criptográficas prontas, a partir de uma API padronizada; por isto, a implementação de algoritmos criptográficos é tratada apenas superficialmente.

O tratamento dado ao tema é prático, a partir de programas de computador escritos na linguagem Java, com a API criptográfica padrão da plataforma Java [38] e a biblioteca criptográfica BouncyCastle [39]. Estas tecnologias foram escolhidas por serem amplamente utilizadas em uma variedade grande de plataformas de computação, desde sistemas corporativos baseados em serviços, sistemas web, até plataformas móveis modernas. Quando apropriado, exemplos reais, extraídos de incidentes causados por mau uso, são utilizados para ilustrar os conceitos apresentados. Os códigos fonte dos programas exemplo podem ser obtidos mediante solicitação aos autores.

O restante deste capítulo está organizado do seguinte modo. A Seção 1.2 explica conceitos básicos da criptografia; a Seção 1.3 mostra os usos adequados dos serviços criptográficos mais comuns; a Seção 1.4 explica os maus usos da criptografia que levam a vulnerabilidades em programas; a Seção 1.5 detalha a implementação em Java de um algoritmo criptográfico simétrico e a Seção 1.6 contém considerações finais.

1.2. Conceitos Básicos

As redes de comunicação abertas, como a Internet, geralmente não oferecem segurança intrínseca, fim-a-fim, para seus usuários. Não existe, por exemplo, sigilo intrínseco para a informação que viaja de um ponto a outro na grande rede. A criptografia é a única tecnologia capaz de garantir o sigilo e a autenticidade da informação em trânsito pelos meios eletrônicos. A criptografia pode ser usada de muitas maneiras, sendo muitas vezes a principal linha de defesa contra bisbilhotagem (*snooping*) e falsificação (*spoofing*).

A Criptografia (do grego *kryptos*, significando oculto) é a ciência que se dedica ao estudo e ao desenvolvimento das técnicas (matemáticas) utilizadas para tornar uma mensagem secreta. Historicamente, o verbo criptografar tem sido usado apenas nesse sentido. Entretanto, a criptografia moderna possui funções como assinaturas digitais, resumo (*hash*) criptográfico e outras, que não se limitam a prover sigilo da informação. De fato, como veremos a seguir, a palavra Criptografia denota hoje um conjunto de técnicas matemáticas das quais uma grande parte dos requisitos, mecanismos e serviços de segurança da informação não podem prescindir.

Esta seção aborda conceitos básicos e serviços criptográficos comumente encontrados em sistemas de software. São eles: objetivos e funções da criptografia; sistemas criptográficos e suas ameaças comuns; criptografia de chave secreta; criptografia de chave pública; encriptação para sigilo e privacidade; autenticação para identificação, irrefutabilidade de mensagens íntegras e autênticas; encriptadores de fluxo e de bloco; distribuição de chaves secretas; distribuição de chaves públicas; acordos de chaves; armazenamento seguro de chaves; gestão do ciclo de vida de chaves criptográficas (geração, distribuição, uso e revogação); chaves de sessão e sistemas criptográficos híbridos. Os livros [1][2][3][4] podem auxiliar no estudo do tema.

1.2.1. Objetivos e funções da criptografia

Historicamente associada ao sigilo, a criptografia moderna também oferece serviços para autenticação, integridade e irrefutabilidade. Os quatro serviços são os seguintes:

1. Confidencialidade (ou sigilo) é obtida com o uso da criptografia para manter a informação secreta, confidencial. Enviar e-mails encriptados e manter arquivos encriptados em cartões de memória são exemplos de confidencialidade.
2. Autenticação é obtida com o uso da criptografia para validar a identidade de uma entidade. Um exemplo de autenticação é o uso de assinaturas digitais para verificar a autoria de uma mensagem de texto ou de um documento eletrônico.
3. Integridade é obtida com o uso da criptografia para garantir que uma porção de dados não foi modificada desde a sua criação. Códigos de detecção de erros são exemplos de mecanismos para verificação de integridade de dados.
4. Irrefutabilidade é obtida pelo uso da criptografia como meio de garantir que o autor de uma mensagem autêntica não possa negar para um terceiro a sua autoria.

Na prática, estes serviços são usados juntos. Por exemplo, uma mensagem de correio eletrônico pode ser encriptada e assinada digitalmente. Deste modo, tanto a confidencialidade quanto a autenticação estarão garantidas. Visto que a assinatura digital é única para a mensagem, a integridade também é preservada.

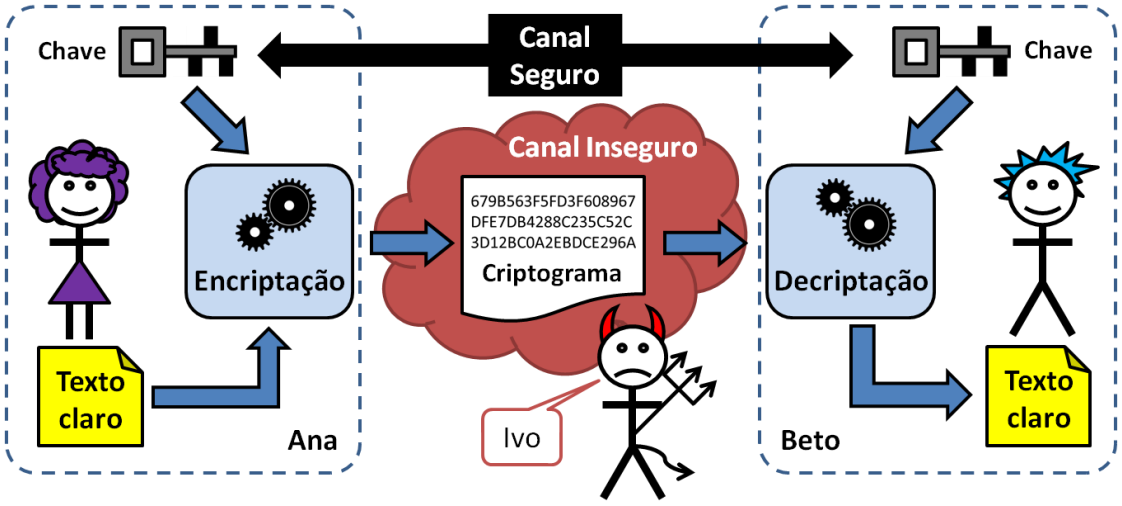


Figura 1: Sistema criptográfico.

1.2.2. Sistemas criptográficos

A Figura 1 mostra um sistema criptográfico e seus elementos fundamentais. Três personagens ilustram a figura: Ana, a remetente das mensagens; Beto, o destinatário das mensagens; e Ivo, o adversário com desejo de conhecer os segredos de Ana e Beto. As mensagens passam por um canal de comunicação inseguro e controlado por Ivo. O algoritmo criptográfico é usado para transformar o texto em claro (legível por qualquer um) em texto encriptado (o criptograma legível apenas por Ana e Beto) e vice-versa.

A chave criptográfica é o parâmetro de configuração do algoritmo que viabiliza a recuperação de um texto claro a partir do texto encriptado. Ana e Beto usam uma chave criptográfica conhecida apenas por eles e compartilhada (ou combinada) por um canal seguro diferenciado. Teoricamente, diz-se que a segurança do sistema criptográfico reside no segredo da chave e não no segredo do algoritmo criptográfico. Grosso modo, em sendo usado um algoritmo de boa reputação, a qualidade da implementação deste algoritmo e o tamanho da chave determinam a dificuldade em quebrar a encriptação da mensagem. A Figura 1 tem os seguintes passos:

1. Ana configura o algoritmo de encriptação com a chave compartilhada com Beto;
2. Ana passa o texto claro para o algoritmo e obtém o criptograma;
3. O criptograma é transmitido pelo canal inseguro e recebido por Beto;
4. Beto configura o algoritmo de decriptação com a chave compartilhada com Ana;
5. Beto decripta o criptograma recebido e obtém o texto claro original.

1.2.3. Ameaças comuns aos sistemas criptográficos

Encontrar vulnerabilidades em sistemas criptográficos, em vez de nas implementações destes sistemas, é uma tarefa complexa, pois normalmente os algoritmos criptográficos modernos são bem projetados, com segurança demonstrável, e submetidos ao escrutínio de pesquisadores por um bom período de tempo. Geralmente, o "teste do tempo" pode ser interpretado como evidência de robustez do algoritmo.

Em termos práticos, algoritmos criptográficos passam a ter valor a partir do momento em que são implementados, seja em software ou em hardware, para prover

confidencialidade, integridade, autenticidade e irrefutabilidade. Tradicionalmente, maior atenção tem sido dada à implementação confiável dos algoritmos criptográficos, uma vez que estas implementações podem expor problemas relacionados com o algoritmo subjacente, ou podem elas mesmas introduzir vulnerabilidades. Recentemente, tem crescido o interesse no uso correto dos algoritmos e suas implementações.

Uma implementação robusta de um sistema criptográfico é difícil de ser obtida, pois exige do desenvolvedor uma grande variedade de conhecimentos teóricos e práticos sobre criptografia, desenvolvimento de software seguro, arquitetura de computadores, compiladores, linguagens de programação, entre outras áreas da Ciência da Computação. Mesmo que o desenvolvedor possua esse tipo de conhecimento amplo e ao mesmo tempo profundo, defeitos de implementação ainda podem ocorrer.

Por causa destas dificuldades, em vez de tentar encontrar falhas nos algoritmos criptográficos, é mais fácil e prático para um adversário (Ivo) procurar por falhas não apenas nas implementações criptográficas, mas também, e às vezes principalmente, nos outros componentes dos sistemas criptográficos, como por exemplo, as camadas de software que encapsulam ou utilizam as implementações criptográficas.

Um ataque simples (porém, quase sempre impraticável) realizado por Ivo contra sistemas criptográficos é aquele conhecido como ataque de força bruta, no qual todas as possibilidades de chaves criptográficas são testadas na tentativa de decriptar corretamente o criptograma. Em geral, chaves longas são mais seguras, pois possuem um número maior de possibilidades. Vale observar que todos os outros ataques listados a seguir são facilitados se o primeiro ataque for bem sucedido e a chave secreta ou privada for comprometida (descoberta, adivinhada, ou deduzida). Ivo pode atacar um sistema criptográfico (por exemplo, um canal de comunicação protegido com criptografia) das seguintes maneiras:

1. Realizando um ataque de força bruta contra as chaves. Neste ataque, todas as chaves válidas possíveis são testadas na decriptação de um criptograma, para uma mensagem conhecida, até que a chave correta seja encontrada;
2. "Grampeando" o canal. Grampear um canal aberto é relativamente simples, pois basta ler a informação em trânsito. Por outro lado, para grampear um canal seguro é preciso não somente ler o criptograma em trânsito, mas também decriptá-lo. Para tal, seria necessário conhecer a chave de decriptação;
3. Reenviando mensagens antigas. Este ataque é possível se as mensagens não são unicamente identificadas (por exemplo, com carimbos temporais – *timestamps*) ou não possuem códigos de autenticação, ou ainda se as chaves não são trocadas periodicamente e com frequência adequada;
4. Personificando uma das partes comunicantes (Ana ou Beto). Ivo pode se fazer passar por Ana ou Beto pela substituição da chave de Ana/Beto pela sua própria, sem o conhecimento de Ana/Beto;
5. Assumindo o papel do intermediário (Man-in-the-Middle). Para este ataque, Ivo obtém as chaves de Ana e de Beto; personifica tanto Ana quanto Beto; intercepta os criptogramas de Ana/Beto para Beto/Ana; decripta estes criptogramas e os encripta novamente com sua própria chave de encriptação, antes de reenviá-los.

1.2.4. Tipos de sistemas criptográficos

Existem dois tipos de sistemas criptográficos, comumente conhecidos como criptografia de chave secreta (ou simétrica) e criptografia de chave pública (ou assimétrica). Na criptografia de chave secreta, uma única chave é usada para encriptar e decriptar a informação. Na criptografia de chave pública, duas chaves são necessárias. Uma chave é usada para encriptar; a outra chave, diferente, é usada para decriptar a informação. Estas duas chaves são matematicamente relacionadas e trabalham aos pares, de modo que o criptograma gerado com uma chave deve ser decriptado pela outra chave do par. Cada chave inverte o trabalho da outra e nenhuma pode ser usada sozinha em um sistema criptográfico. Nos sistemas de chave pública, uma das chaves do par é dita privada (a de decriptação), a outra é feita pública (a de encriptação).

1.2.5. Criptografia de chave secreta

Os sistemas criptográficos de chave secreta modernos possuem geralmente bom desempenho, mesmo em computadores considerados lentos. Com esta tecnologia, apenas uma chave é usada para encriptar e decriptar a informação. A Figura 2 ilustra os passos da encriptação com algoritmos simétricos de chave secreta, conforme a seguir:

1. Ana configura o algoritmo para o modo de encriptação com a chave secreta;
2. Ana alimenta o algoritmo com a mensagem original, o texto claro;
3. Ana encripta a mensagem e obtém o criptograma (mensagem encriptada).

Apenas a chave usada na encriptação pode decriptar corretamente a informação. Por isso, esta chave deve ser protegida e guardada em segredo; daí o nome de chave secreta. A Figura 2 também ilustra os passos da decriptação com chave secreta:

1. Beto configura o algoritmo para o modo de decriptação com a chave secreta;
2. Beto alimenta o algoritmo com a mensagem encriptada (criptograma);
3. Beto decripta a mensagem encriptada e obtém o texto claro original.

Teoricamente, este sistema criptográfico pode ser diretamente utilizado para encriptação bidirecional com a mesma chave. Porém, conforme tratado adiante neste texto, na prática, diversos detalhes de implementação dificultam a utilização segura da mesma chave para encriptação nas duas direções do canal de comunicação.

Na criptografia simétrica, a chave secreta deve ser conhecida por todos aqueles

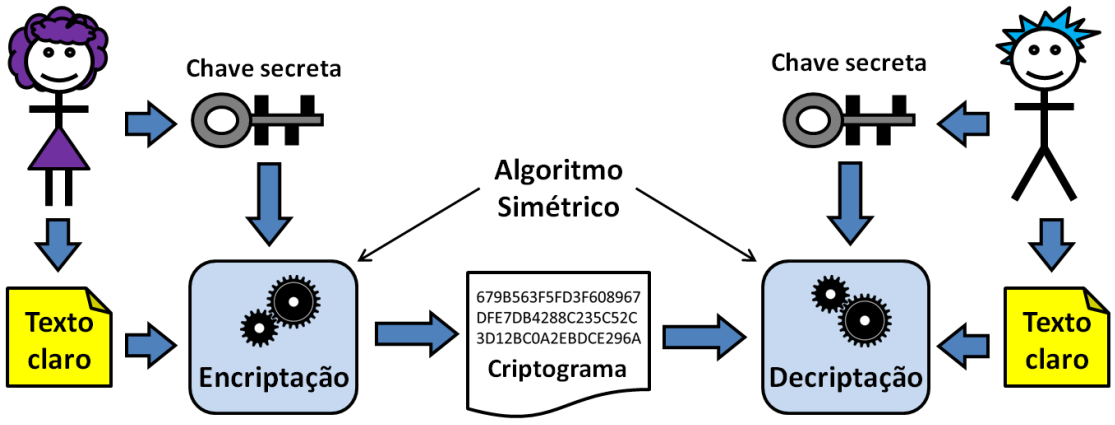


Figura 2: Sistema criptográfico simétrico.

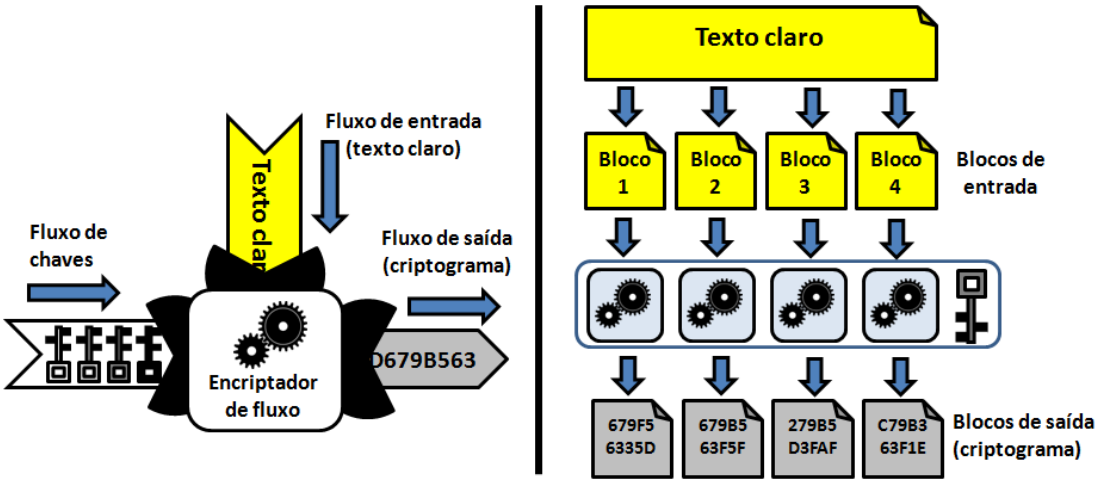


Figura 3: Encriptadores de fluxo (a esquerda) e de bloco (a direita).

que precisam decriptar a informação encriptada com ela. Mas como compartilhar a chave secreta sem que ela seja descoberta pelos adversários? Uma solução seria marcar um encontro secreto com todos que devem conhecer a chave. Porém, na Internet, isto não é fácil, afinal, usa-se a Internet quando encontros presenciais são difíceis. Sabe-se que a Internet é insegura por natureza (por isso, usar a criptografia), então não se pode simplesmente distribuir a chave secreta por e-mail ou mensagem de texto. A distribuição de chaves é um aspecto importante da criptografia de chave secreta. Apesar das dificuldades de distribuição de chaves, a criptografia de chave secreta é muito usada. Suas dificuldades aparentes podem ser contornadas pela combinação desta tecnologia com a criptografia de chave pública, originando sistemas criptográficos híbridos.

1.2.5.1. Encriptadores de fluxo e de bloco

Existem duas categorias de algoritmos simétricos (Figura 3): os encriptadores de fluxo e de bloco. Nos encriptadores de bloco, o texto claro é quebrado em blocos de bits de tamanho fixo. O encriptador trabalha sobre blocos e produz saídas em blocos também. O tamanho da chave criptográfica é geralmente um múltiplo do tamanho do bloco.

Os encriptadores de fluxo trabalham sobre sequências (de bits). A sequência ou fluxo de entrada é transformado continuamente na sequência ou fluxo de saída, bit a bit. É importante que a chave criptográfica seja uma sequência de bits pelo menos do mesmo tamanho do fluxo de entrada. Na prática, os bits da chave podem ser produzidos por um gerador de sequências de bits pseudoaleatórias, a partir de uma chave mestra de tamanho fixo.

1.2.6. Criptografia de chave pública

Devido à complexidade das operações matemáticas envolvidas, a criptografia de chave pública tradicional possui em geral um desempenho pior se comparada à criptografia de chave secreta, no mesmo hardware. A criptografia de chave pública utiliza duas chaves, que são relacionadas matematicamente e construídas para trabalharem juntas. Uma das chaves do par é dita a chave privada (pessoal) e é mantida em segredo, sendo conhecida

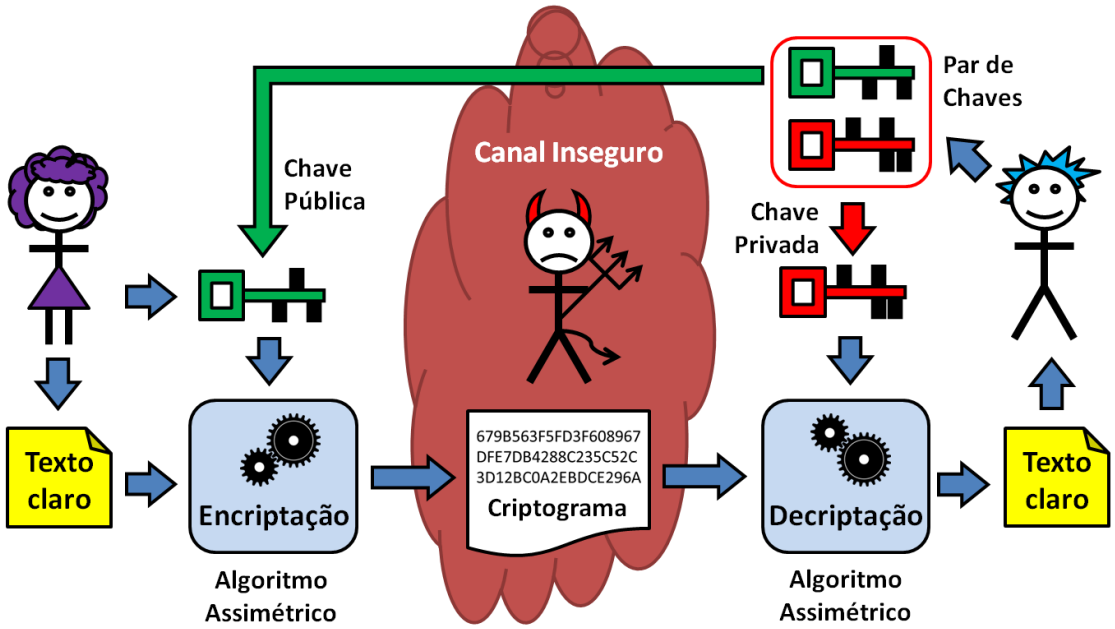


Figura 4: Sistema criptográfico assimétrico para sigilo.

apenas pelo dono do par de chaves. A outra chave do par é dita a chave pública por poder ser conhecida publicamente. A criptografia de chave pública pode ser usada para obter sigilo. Neste caso, a encriptação com a chave pública torna possível que qualquer um envie criptogramas para o dono da chave privada.

1.2.6.1. Encriptação para sigilo e privacidade

A Figura 4 ilustra um sistema criptográfico assimétrico para sigilo e seus elementos básicos. Mais uma vez, Ana, Beto e Ivo são os personagens. As mensagens de Ana para Beto são transmitidas por um canal inseguro, controlado por Ivo. Beto possui um par de chaves, uma chave pública e outra privada. Ana conhece a chave pública de Beto, mas somente o dono do par de chaves (Beto) conhece a chave privada (não há segredo compartilhado). A Figura 4 contém os passos a seguir:

1. Ana configura o algoritmo de encriptação com a chave pública de Beto;
2. Ana alimenta o algoritmo com a mensagem original (o texto claro);
3. O texto claro é encriptado e transmitido pelo canal inseguro para Beto;
4. Beto configura o algoritmo de decriptação com a sua própria chave privada;
5. O criptograma é decriptado e o texto claro original é obtido por Beto.

Analisando a Figura 4, observa-se que Ana envia uma mensagem privada para Beto. Para fazer isso, Ana encripta a mensagem usando a chave pública de Beto. Ana conhece a chave pública de Beto por que ela foi divulgada por Beto. Porém, o criptograma só pode ser decriptado pela chave privada de Beto; nem Ana pode fazê-lo.

Para obter comunicação segura bidirecional, basta acrescentar ao sistema criptográfico o mesmo processo no sentido oposto (de Beto para Ana), com outro par de chaves. Isto é, Beto usa a chave pública de Ana para enviar mensagens encriptadas para ela. Ana, ao receber a mensagem, usa sua própria chave privada pessoal para decriptar a mensagem enviada por Beto. A criptografia de chave pública é indispensável para a

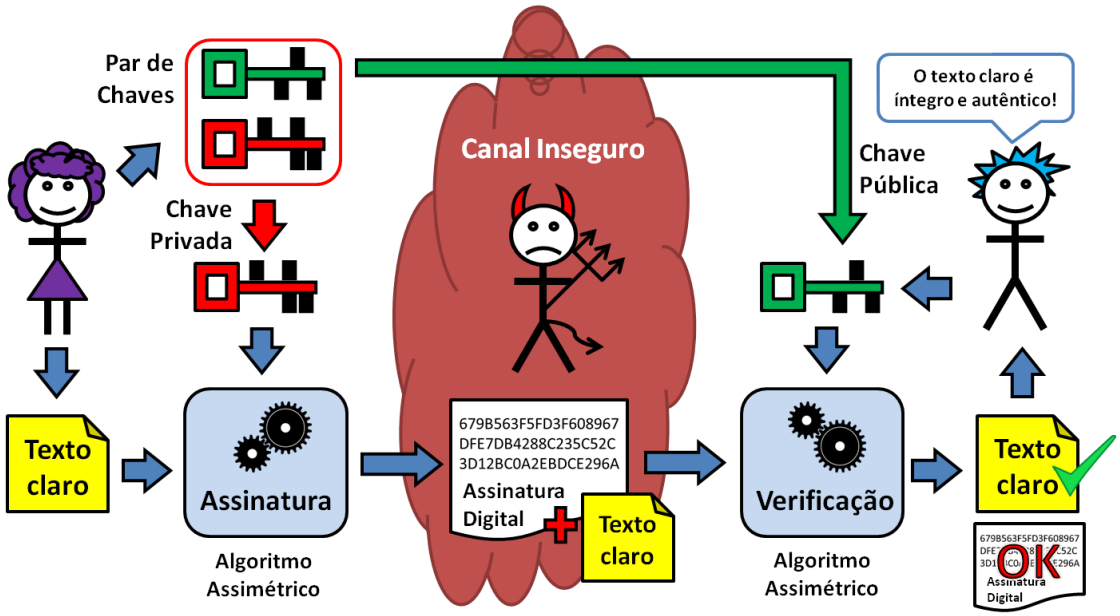


Figura 5: Sistema criptográfico assimétrico para autenticidade.

segurança da Internet, pois torna possível a comunicação privada em uma rede pública. A criptografia de chave pública é a base para outros dois serviços: a autenticação das partes e a verificação de integridade das mensagens.

1.2.6.2. Autenticação e irrefutabilidade

O uso da criptografia de chave pública para autenticação de mensagens é quase o oposto do uso para sigilo. A criptografia de chave pública para assinatura digital é usada para obter integridade, autenticidade e irrefutabilidade. Chama-se assinatura digital ao resultado de uma certa operação criptográfica com a chave privada sobre o texto claro. Neste caso, o dono da chave privada pode gerar mensagens assinadas, que podem ser verificadas por qualquer um que conheça a chave pública correspondente, portanto, sendo capaz de verificar a autenticidade da assinatura digital. Nem sempre a operação de assinatura é uma encriptação e a sua verificação é uma decriptação.

Visto que qualquer um de posse da chave pública pode "decriptar" a assinatura digital, ela não é mais secreta, mas possui outra propriedade: a irrefutabilidade. Isto é, quem quer que verifique a assinatura com a chave pública, sabe que ela foi produzida por uma chave privada exclusiva; logo, a mensagem não pode ter sido gerada por mais ninguém além do proprietário da chave privada.

Na Figura 5, Ana usa sua chave privada para assinar digitalmente uma mensagem para Beto. O texto claro e a assinatura digital são enviados por um canal inseguro e podem ser lidos por todos, por isso a mensagem não é secreta. Qualquer um que conheça a chave pública de Ana (todo mundo, inclusive Beto), pode verificar a assinatura digital. Ivo pode ler a mensagem, mas não pode falsificá-la, pois não conhece a chave privada de Ana.

O principal problema administrativo deste modelo de autenticação é justamente a confiança depositada na chave pública. Se a chave pública de alguém pode ser

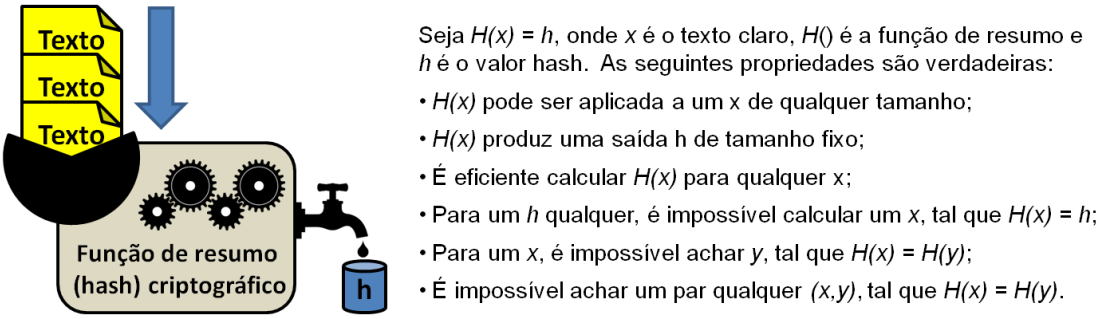


Figura 6: Funções de resumo (*hash*) criptográfico.

encontrada em qualquer lugar, então fica difícil saber se esta chave não foi corrompida ou substituída. O problema de garantir a autenticidade da chave pública é muito importante e, se não for solucionado satisfatoriamente, pode comprometer a confiança no sistema criptográfico. Uma maneira de validar chaves públicas é fazer com que elas sejam emitidas por Autoridades Certificadoras (AC) de uma Infraestrutura de Chaves Públicas (ICP), que torne possível a verificação da autenticidade de tais chaves.

1.2.6.3. Assinaturas digitais de tamanho fixo

A geração de assinaturas digitais com o mesmo tamanho do texto claro é ruim para a transmissão de dados. Além disso, a matemática envolvida na criptografia de chave pública é mais trabalhosa (complexa) e, por isso, não teria desempenho eficiente em processadores mais lentos. Por estas razões, normalmente, não é o texto claro inteiro que é assinado digitalmente, mas sim um resumo dele, que o identifique unicamente. Este identificador único é calculado por rotinas matemáticas chamadas de funções de resumo (ou *hash*) criptográfico, cujas propriedades são ilustradas na Figura 6.

Estas funções de *hash* geram uma sequência de bits, o valor do *hash*, que é único para o documento de entrada da função. O *hash* é muito menor que o documento original e geralmente tem um tamanho fixo de dezenas (algumas centenas) de bits. A função de *hash* é unidirecional porque não é reversível, isto é, não é possível recuperar o documento original a partir da sequência binária do *hash*. Além disso, idealmente, não existem dois documentos que geram o mesmo valor de *hash*.

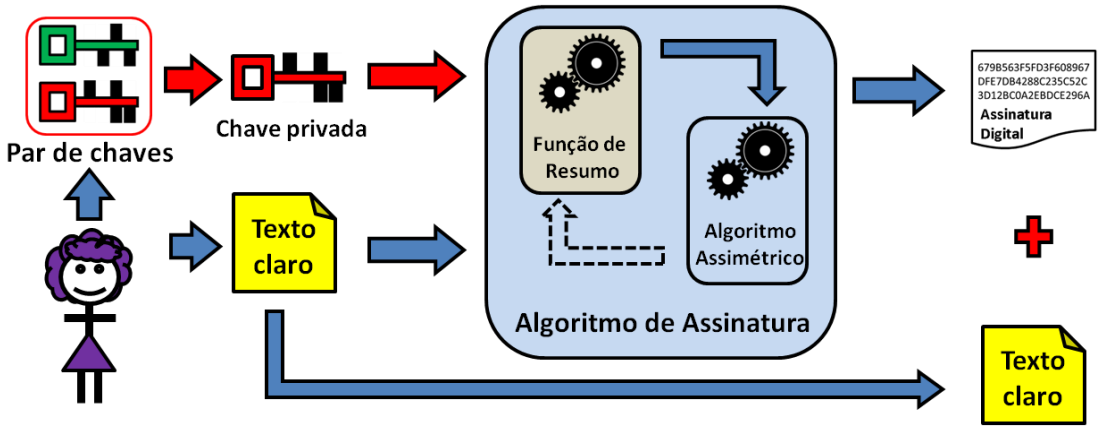


Figura 7: Assinaturas digitais de tamanho fixo.

Na Figura 7, Ana não assina o texto claro inteiro, pois ele pode ter um tamanho arbitrariamente grande. De fato, Ana usa um mecanismo de assinaturas digitais que calcula o *hash* do texto claro usando uma função de resumo criptográfico. O *hash* da mensagem é então assinado com a chave privada de Ana usando um mecanismo de assinatura digital. O modo de combinar a função de resumo e o algoritmo assimétrico é dependente de cada mecanismo de assinatura digital. Esta assinatura digital fixa é enviada junto com o texto claro original e está acessível para quem precisar verificar a autoria do texto claro.

A integridade do texto claro também está garantida, pois se ele tiver qualquer bit modificado, o *hash* calculado na verificação não corresponderá mais àquele da assinatura e a assinatura não será mais verificada com sucesso. Ana não pode mais negar (refutar) a autoria do texto claro, pois há uma assinatura digital feita com sua chave privada pessoal. Ninguém precisa da ajuda de Ana para verificar a autoria do documento, desde que a chave pública de Ana esteja amplamente disponível. Por isso, a assinatura digital é irrefutável.

1.2.7. Distribuição de chaves

A despeito da segurança oferecida pelas funções criptográficas para encriptação e assinatura digital, resta ainda um aspecto importante a ser considerado na análise de viabilidade de um sistema criptográfico: a distribuição de chaves. Distribuir as chaves na implantação de um sistema criptográfico significa oferecer a cada indivíduo participante do sistema todas as chaves necessárias para que ele consiga comunicar-se de modo seguro com qualquer outro participante. As tecnologias criptográficas simétricas e assimétricas possuem restrições diferentes quanto à distribuição de chaves.

Na criptografia simétrica, deve haver uma chave secreta compartilhada dentro de cada grupo de pessoas que deseja falar sem o conhecimento de outros. Por exemplo, para Ana e Beto trocarem segredos sem o conhecimento de Ivo, eles dois devem compartilhar uma chave secreta. A Figura 8 mostra a distribuição de chaves secretas entre pares para quatro indivíduos. Ana tem uma chave secreta para cada pessoa com que ela troca segredos sem que os outros saibam. Beto também quer ter seus segredos com cada pessoa em separado, por isso ele tem chaves secretas compartilhadas com cada participante. O mesmo vale para cada par de indivíduos.

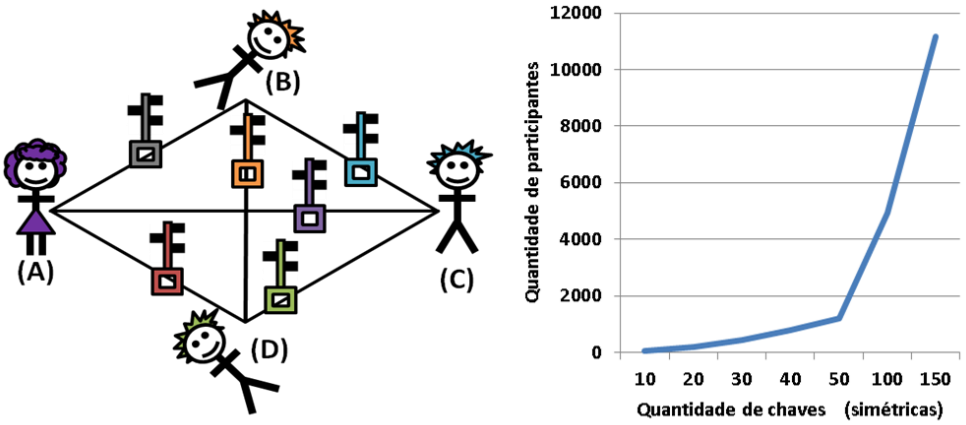


Figura 8: Distribuição de chaves simétricas e sua função de crescimento.

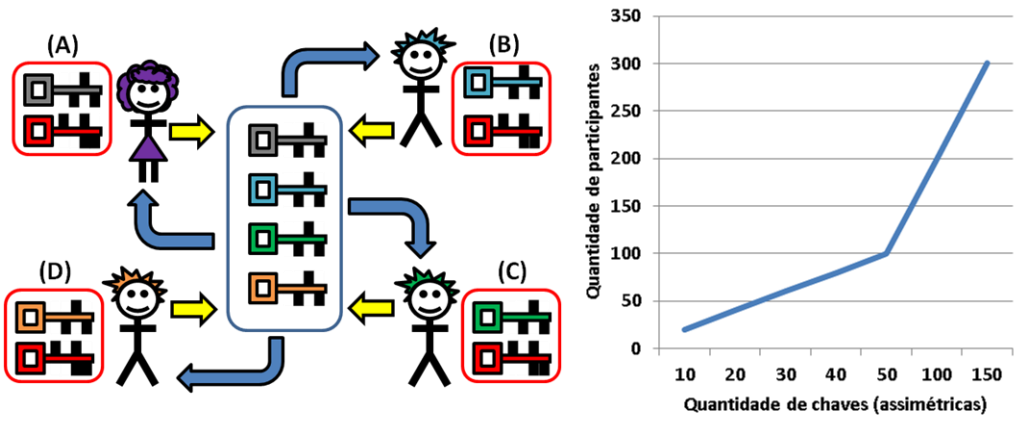


Figura 9: Distribuição de chaves assimétricas e sua função de crescimento.

Uma característica interessante dessa configuração é o sigilo mantido entre os pares. Cada par mantém seus segredos fora do alcance dos outros. O problema está na quantidade de chaves necessária para manter este nível alto de sigilo. Para um grupo de quatro pessoas, seis chaves são necessárias. Para cinco pessoas, dez chaves são necessárias, e assim por diante. Deste modo, este comportamento pode ser descrito por uma função quadrática. Isto é, a quantidade de chaves secretas cresce em função do quadrado da quantidade de pessoas do grupo.

O gráfico da Figura 8 mostra o crescimento da quantidade de chaves secretas em função do quadrado da quantidade dos participantes. Observa-se que para meros cem participantes, a quantidade de chaves secretas chega a alguns milhares, para cento e cinquenta participantes, tem-se mais de dez mil chaves e para duzentos participantes a quantidade de chaves quase chega a duzentos mil.

A criptografia assimétrica de chaves públicas simplifica muito o problema de distribuição de chaves. Cada participante só precisa ter o seu par de chaves, manter em segredo a chave privada e divulgar a chave pública. Pode haver um repositório global para todas as chaves públicas. A Figura 9 mostra a distribuição de chaves públicas para um grupo de quatro indivíduos. O gráfico da Figura 9 mostra o crescimento da quantidade de chaves (públicas e privadas) como uma função afim da quantidade de participantes. A quantidade de chaves é o dobro da quantidade de participantes.

1.2.8. Transporte de chaves e sistemas criptográficos híbridos

A criptografia de chave pública é recomendada para grupos grandes e ambientes dinâmicos e públicos. Por outro lado, a criptografia de chave secreta é recomendada para grupos pequenos e estáticos. A Tabela 1 compara as tecnologias criptográficas simétricas e assimétricas. Observa-se que as deficiências de uma tecnologia são complementadas pelas vantagens da outra. De fato, uma solução amplamente utilizada pelos protocolos de comunicação segura na Internet usa uma combinação das tecnologias simétrica e assimétrica, chamada de sistemas criptográficos híbridos.

Uma combinação interessante das tecnologias é aquela que oferece o desempenho superior da criptografia simétrica com a autenticação forte e a facilidade de distribuição de chaves da criptografia assimétrica. Tal combinação pode ser

implementada pelo uso de uma chave secreta encriptada por uma chave pública para transporte seguro. A criptografia assimétrica é usada como canal seguro para o compartilhamento da chave secreta. A Figura 10 mostra os passos da encriptação com a chave secreta em um sistema criptográfico híbrido:

1. Ana configura um algoritmo assimétrico com a chave pública de Beto;
2. Ana cria uma chave secreta e alimenta o algoritmo assimétrico com ela;
3. A chave secreta é encriptada pela chave pública de Beto;
4. Ana configura o algoritmo simétrico com a chave secreta;
5. Ana alimenta o algoritmo simétrico com um texto claro para Beto;
6. Ana envia o criptograma para Beto, junto com a chave secreta encriptada.

Durante uma comunicação segura, a sequência completa de passos para encriptação só é executada uma única vez, no início da comunicação, para compartilhar a chave secreta. A partir do segundo criptograma da conversa, basta que os passos 5 e 6 sejam executados. A tarefa de Beto é recuperar a chave secreta e manter a comunicação com Ana usando a chave secreta para proteger os dados.

De modo análogo, na decriptação, os passos para recuperar a chave secreta só são executados no início da comunicação. Observados os detalhes de implementação, depois que ambos (Ana e Beto) possuem a chave secreta, a comunicação pode ocorrer nos dois sentidos. Isto é, tanto de Ana para Beto, quanto de Beto para Ana.

Tabela 1: Comparação entre tecnologias criptográficas simétricas e assimétricas.

Criptografia Simétrica	Criptografia Assimétrica
Desempenho superior (mais rápida).	Desempenho inferior (mais lenta).
Não oferece autenticação forte, o segredo é compartilhado.	Autenticação forte com assinaturas digitais.
Não é irrefutável (a autoria pode ser negada).	Assinatura digital é irrefutável.
Distribuição de muitas chaves é trabalhosa.	Distribuição de muitas chaves é simplificada.

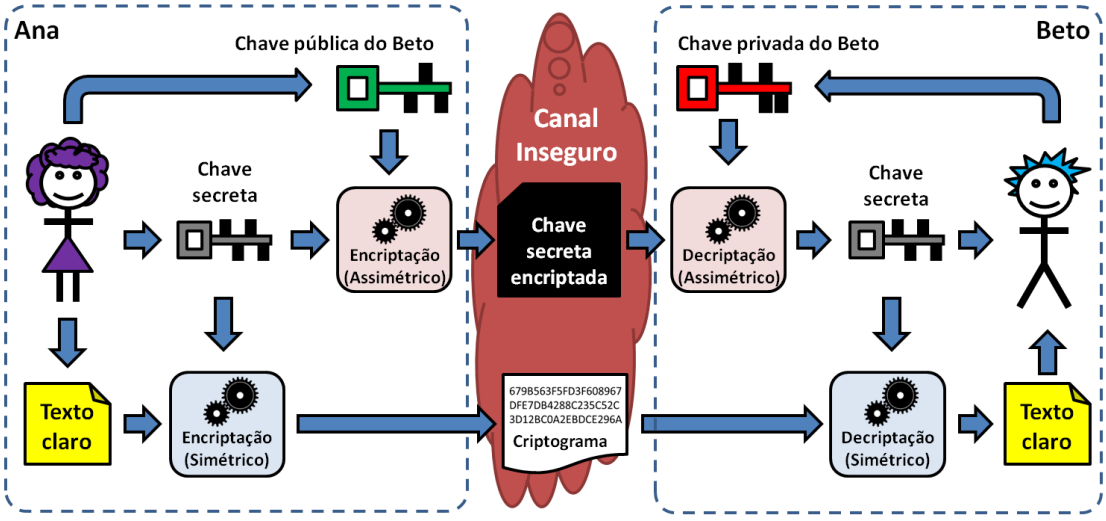


Figura 10: Sistema criptográfico híbrido e o transporte da chave de secreta.

1.2.9. Aspectos de gestão de chaves criptográficas

A gestão de chaves criptográficas é um tópico complexo e bastante suscetível a erros de construção, sendo muitas vezes a causa de vulnerabilidades graves em sistemas criptográficos. Esta seção aborda brevemente o armazenamento seguro de chaves em software, os métodos de acordo de chave e o ciclo de vida de chaves criptográficas.

1.2.9.1. Armazenamento seguro de chaves em software

O sigilo das chaves de encriptação e assinatura (privadas e secretas) é da maior importância para a segurança dos sistemas criptográficos. Nas implementações em software, sem auxílio de hardware de segurança, as chaves privadas e secretas devem ser guardadas de forma encriptada. A criptografia de proteção das chaves também necessita de chaves para encriptação, que por sua vez também precisam de proteção criptográfica, e assim por diante, levando a um problema aparentemente insolúvel.

A Criptografia Baseada em Senhas (*Password-Based Encryption* - PBE) resolve esta questão ao proporcionar os meios adequados para gerar uma chave criptográfica a partir de uma senha. A Figura 11 ilustra o funcionamento do método. O mecanismo de PBE é geralmente utilizado para proteger chaves logo que elas são criadas, para reduzir a exposição destas chaves. O mecanismo encapsula uma função de derivação de chaves (*Key Derivation Function* - KDF) e uma rotina de encriptação simétrica. Os passos de uso do PBE são os seguintes:

1. A PBE é configurada com uma senha (forte) e parâmetros de segurança;
2. A chave (privada ou secreta) a ser protegida é passada para a PBE;
3. Uma chave de proteção de chaves (CPC) é gerada pelo KDF a partir da senha;
4. A encriptação simétrica é configurada com a CPC;
5. A chave a ser protegida é encriptada e pode ser armazenada em um arquivo;
6. A CPC pode ser derivada novamente no futuro e por isto deve ser apagada.

A recuperação da chave encriptada é feita pela decriptação com a CPC derivada novamente a partir da senha. Vale ainda mencionar que as cópias em claro da chave a ser protegida devem ser apagadas do sistema, como uma boa prática de segurança. Mecanismos de verificação de integridade podem ser combinados ao método PBE.

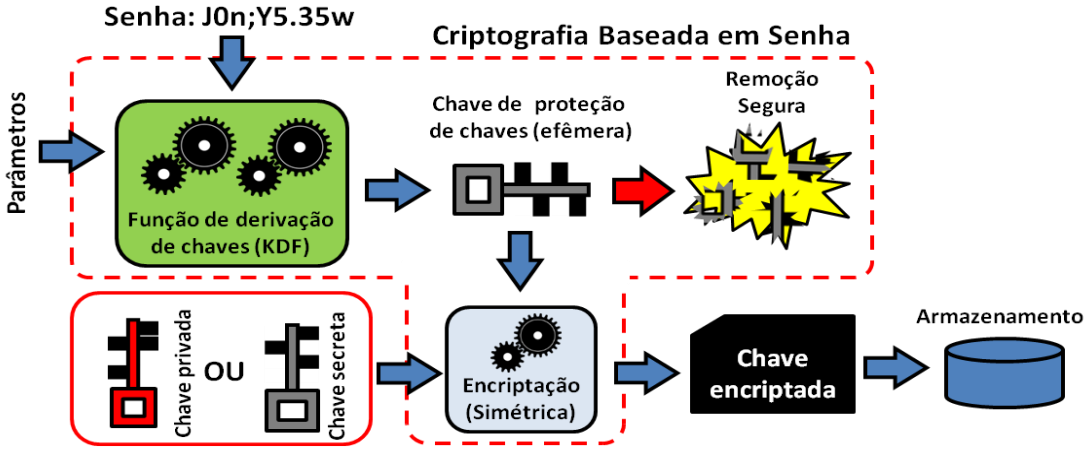


Figura 11: Armazenamento seguro de chaves com criptografia baseada em senhas.

1.2.9.1. Acordo de chaves

Há ocasiões em que entidades que nunca tiveram a oportunidade de compartilhar chaves criptográficas (por exemplo, nunca se encontram ou não se conhecem) precisam se comunicar em sigilo. Nestes casos, uma chave efêmera, usada apenas para algumas encriptações e decriptações decorrentes de uma conversa, pode ser gerada momentos antes do início da conversa. Os métodos de acordo de chaves são utilizados para combinar ou negociar uma chave secreta entre dois ou mais participantes usando um canal público. Uma característica interessante destes métodos é que o segredo compartilhado (a partir do qual a chave será derivada) é combinado pela troca de informações públicas por meio de um canal inseguro.

1.2.9.2. Ciclo de vida de chaves criptográficas

Uma chave criptográfica tem uma finalidade e está associada a uma entidade (pessoa ou sistema). A Figura 12 ilustra os estados e as fases da vida de uma chave criptográfica [28]. Uma chave possui seis estados, que são organizados em um ciclo de vida com quatro fases. Cada fase tem várias atividades específicas a serem realizadas. Em situações normais, uma chave passa a maior parte de sua vida útil no estado Ativado da fase Operacional. Uma chave desativada, que não está mais em operação, ainda pode ser utilizada para recuperar informação encriptada antiga.

O criptoperíodo é o período de tempo durante o qual as chaves para um determinado sistema permanecem em vigor e autorizadas para utilização. Uma chave sai de operação por dois motivos: o seu tempo de uso (criptoperíodo) terminou e ela será desativada, ou ela foi comprometida em algum incidente de segurança. A destruição de uma chave requer cuidados com a asseguração do apagamento ou remoção segura. Uma chave mal apagada ainda pode ser comprometida. Além disso, a destruição de uma chave comprometida não desfaz o comprometimento dos dados encriptados por ela, pois um segredo revelado nunca volta a ser um segredo de novo.

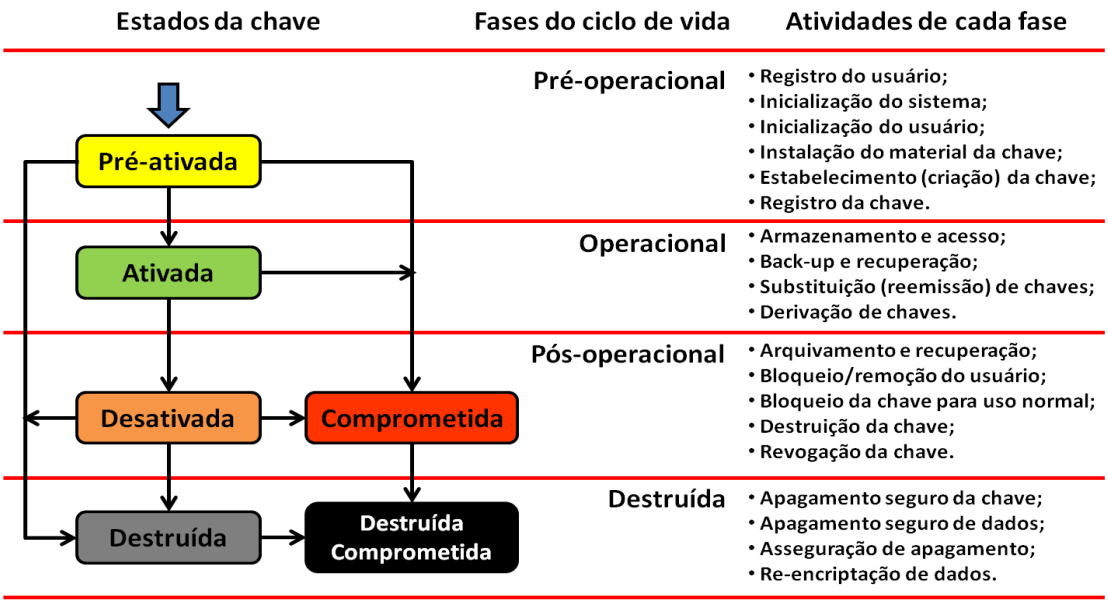


Figura 12: Estados e fases do ciclo de vida de uma chave criptográfica.

1.3. Como usar a criptografia: entendendo os bons usos

Esta seção aborda, por meio de (trechos de) programas em Java [38] e a biblioteca criptográfica BouncyCastle [39], os conceitos descritos na seção anterior. A bibliografia de apoio [10][11][12] explica aspectos específicos de programação da API criptográfica. Em particular, aqui são abordados os seguintes assuntos: encriptação/decriptação simétrica e assimétrica; modos de operação e suas propriedades; verificação de integridade e autenticação de mensagem; encriptação autenticada; criptografia baseada em senhas; transporte de chave simétrica com criptografia assimétrica; aleatoriedade e geração de números pseudoaleatórios; e distribuição e validação de certificados digitais. Ainda, os programas são inspirados nos padrões de projeto para criptografia [9].

1.3.1. O padrão de projeto de software criptográfico

A estrutura de um sistema criptográfico seque um padrão recorrente de arquitetura que foi primeiramente documento pelos autores em [9] e ficou conhecido como o padrão de projeto de software criptográfico. A Figura 13 mostra o diagrama de classes, em UML, de um sistema criptográfico simétrico para sigilo, em que Ana encripta e Beto decripta. O leitor deve ser capaz de abstrair a arquitetura de software da Figura 13 e visualizar estruturas semelhantes nos trechos de programas criptográficos mostrados a seguir.

Na figura, Ana e Beto usam a criptografia por meio de um aplicativo (App), em instanciações distintas, AnaApp e BetoApp, respectivamente. O App possui uma classe com estereótipo de controlador (CriptoCtrl) que é responsável por orquestrar os serviços criptográficos e a configuração de parâmetros de segurança juntamente com a lógica da aplicação. Por exemplo, no caso de Ana, o CriptoCtrl formata a mensagem m em um formato adequado para a criptografia, encripta o texto claro tc com a chave k_e e faz com que Beto receba o criptograma c. Beto, por sua vez, utiliza o seu orquestrador criptográfico para decriptar c com a chave k_d e converter o texto claro tc para a formatação requerida m. Ana e Beto utilizam instâncias diferentes da mesma classe Algoritmo (criptográfico).

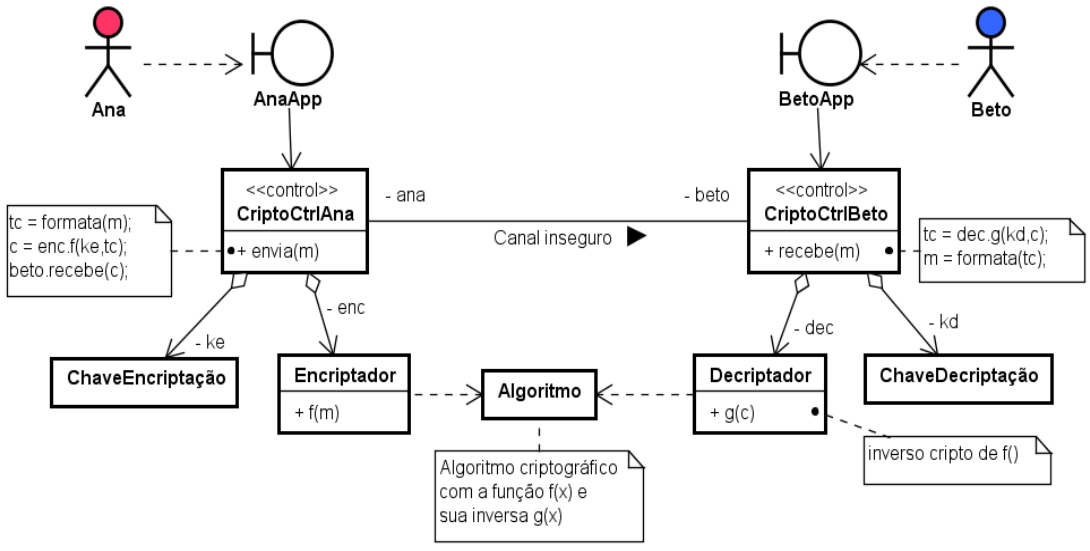


Figura 13: Sistema criptográfico simétrico para sigilo, em que Ana encripta e Beto decripta.

1.3.2. Encriptação e decriptação com chave secreta (simétrica)

O trecho de código mostrado no Programa 1 contém todos os elementos do sistema criptográfico simétrico usado por Ana e Beto. O Programa 1 não contém o código fonte completo, mas apenas o núcleo essencial para o sistema criptográfico. Trechos secundários, tais como importações de pacotes, declarações de variáveis auxiliares e apresentação de resultados, não são mostrados. Este estilo de apresentação será reproduzido no restante deste texto. O trecho de código do Programa 1 está dividido em três partes. A primeira parte, nas linhas de 01 a 06, contém as configurações comuns do sistema criptográfico. A segunda parte (linhas de 08 a 12) mostra o processo de encriptação pela Ana. A decriptação por Beto está nas linhas de 13 a 16.

As configurações comuns do sistema criptográfico são as seguintes. Na linha 02, o provedor criptográfico BouncyCastle (identificado adiante por "BC") é adicionado dinamicamente. A configuração dinâmica é mais flexível e recomendada nas instalações em que não é possível, sem modificações do sistema operacional, alterar as configurações da Máquina Virtual Java (JVM), como é o caso dos dispositivos móveis Android. Nas linhas 03 e 04, um gerador de chaves é instanciado com o algoritmo criptográfico AES, do provedor "BC", e inicializado para chaves de 256 bits. A chave criptográfica é gerada na linha 05. A linha 06 instancia o encriptador para o algoritmo AES como um encriptador de bloco no modo de operação CTR e *padding* PKCS#7 (os modos de operação e os mecanismos de *padding* são explicados adiante no texto).

O processo de encriptação começa na linha 09 com a configuração do encriptador para encriptação com a chave *k*. Na linha 11, os bytes do texto claro são encriptados pelo método `doFinal()`, produzindo o criptograma. O modo CTR requer um IV único, que não foi informado por Ana e por isto foi gerado automaticamente pelo encriptador e recuperado para uso da decriptação (linha 12). Na decriptação por Beto, o encriptador é configurado para decriptação com a chave compartilhada *k* e o mesmo IV usado na encriptação, linha 15. A decriptação (linha 16) devolve os bytes do texto claro e é direta pelo método `doFinal()`. O IV deve atender às propriedades do modo de operação CTR, sendo único e específico do criptograma, não devendo ser reutilizado com a mesma chave. Além do CTR, outros modos de operação comumente utilizados são o ECB, o CBC, o CFB e o OFB, detalhados adiante no texto.

Programa 1: Encriptação com criptografia simétrica com AES/CTR/PKCS7Padding.

```
01  // configurações do sistema criptográfico para Ana e Beto
02  Security.addProvider(new BouncyCastleProvider());
03  KeyGenerator g = KeyGenerator.getInstance("AES","BC");
04  g.init(256);
05  Key k = g.generateKey();
06  Cipher c = Cipher.getInstance("AES/CTR/PKCS7Padding", "BC");
07
08  // Encriptação pela Ana
09  c.init(Cipher.ENCRYPT_MODE, k);
10  byte[] textoclaroAna = "Testando o AES..".getBytes();
11  byte[] criptograma = c.doFinal(textoclaroAna);
12  byte[] iv = c.getIV();
13
14  // decriptação pelo Beto
15  c.init(Cipher.DECRYPT_MODE, k, new IvParameterSpec(iv));
16  byte[] textoclaroBeto = c.doFinal(criptograma);
```

1.3.3. Mecanismos de preenchimento de blocos incompletos

Os encriptadores de bloco trabalham sobre dados com tamanho igual a um múltiplo inteiro do tamanho do bloco. Já o texto claro tem tamanho livre. Para obter flexibilidade no texto claro, é necessário um mecanismo de preenchimento de blocos incompletos a fim de que o tamanho do texto claro seja múltiplo do tamanho do bloco. Chama-se de *padding* (preenchimento) o ato de completar a cadeia binária do texto claro para que ela tenha o tamanho múltiplo do tamanho do bloco do algoritmo de encriptação.

Os *paddings* PKCS#5, para blocos de 8 bytes, e PKCS#7, para blocos de 16 bytes, são calculados utilizando a mesma regra ilustrada a seguir para bloco de 16 bytes, em que M é o texto claro da mensagem, L é o tamanho de M em bytes, PM é a mensagem preenchida com o valor do *padding*, o caractere "|" é a concatenação e "resto" é o resto da divisão inteira de L por 16. Assim,

- Se o resto de L dividido por 16 é 15, PM = M | 0x01;
- Se o resto de L dividido por 16 é 14, PM = M | 0x0202;
- Se o resto de L dividido por 16 é 13, PM = M | 0x030303;
- Se o resto de L dividido por 16 é 12, PM = M | 0x04040404;
- Se o resto de L dividido por 16 é 11, PM = M | 0x0505050505;
- Se o resto de L dividido por 16 é 10, PM = M | 0x060606060606;
- Se o resto de L dividido por 16 é 09, PM = M | 0x07070707070707.

Assim por diante até a situação em que M tem um resto que é um bloco quase completo, exceto por um byte, e o resto de L dividido por 16 é 1. Nesta situação, a mensagem preenchida é PM = M | 0x0E0E0E0E0E0E0E0E0E0E0E0E0E0E0E. Finalmente, quando M tem tamanho múltiplo inteiro do bloco e o resto de L dividido por 16 é 0, temos PM = M | 0x0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F.

O trecho de código do Programa 2 mostra que o mecanismo de *padding* deve ser determinado no momento de instanciação do encriptador de bloco. Três mecanismos são ilustrados (PKCS#5, PKCS#7 e X9.23) além das opções sem *padding* e com *padding default*. Quando não há *padding*, somente blocos completos podem ser encriptados. Quando há *padding*, o criptograma é computado sobre o texto claro com *padding* e por isto, é maior que o texto claro original, podendo chegar a ter um bloco a mais.

Programa 2: Mecanismos de padding das cifras de bloco com PKCS5, PKCS7 e X9.23.

```
01  Cipher c1 = Cipher.getInstance("AES/ECB/NoPadding"   , "BC");
02  Cipher c2 = Cipher.getInstance("AES"                  , "BC");
03  Cipher c3 = Cipher.getInstance("AES/ECB/PKCS5Padding", "BC");
04  Cipher c4 = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
05  Cipher c5 = Cipher.getInstance("AES/ECB/X9.23Padding", "BC");
```

```
01  // configurações do sistema criptográfico
02  Chave AES de 128 bits     : 073815E64B5F3B04F793B09813908994
03  Um bloco de texto claro   : Testando o AES..
04  Texto claro em hexadecimal: 54657374616E646F206F204145532E2E
05
06  // resultados da encriptação com padding
07  Nenhum : A0DF739B83DD6A72EFB4F5941B2A9915
08  Default:A0DF739B83DD6A72EFB4F5941B2A9915F0FFEC381B1CF8466000A69620F61A36
09  PKCS5  :A0DF739B83DD6A72EFB4F5941B2A9915F0FFEC381B1CF8466000A69620F61A36
10  PKCS7  :A0DF739B83DD6A72EFB4F5941B2A9915F0FFEC381B1CF8466000A69620F61A36
11  X9.23  :A0DF739B83DD6A72EFB4F5941B2A9915C378DA269D95F861666C9ABEA33AC9C5
```

1.3.4. Modos de operação dos encriptadores de bloco e suas propriedades

Os modos de operação dos encriptadores de bloco combinam de maneiras diferentes o IV e o texto claro (ou o criptograma) com o algoritmo criptográfico e a chave, para produzir encadeamentos de blocos encriptados característicos de cada um deles. Assim, o mesmo trio texto claro, chave e IV, quando usados com modos de operação diferentes, resultam em criptogramas diferentes. Os modos de operação mais usados são ECB, CBC, CFB, OFB e CTR [26]. Os modos ECB e CBC são encriptadores de bloco típicos. Os modos CFB, OFB e CTR se comportam como encriptadores de fluxo. Os modos de operação são diferentes em relação à recuperação de erros e à perda de sincronismo.

A Saída 1 mostra o resultado sobre o texto decriptado da modificação de um bit nos criptogramas gerados com os modos ECB, CBC, CFB, OFB e CTR, a partir da mesma chave e IV. A modificação é simplesmente o XOR do byte 1 com o valor 0x01. As linhas de 01 a 03 mostram a chave, o IV em hexadecimal e o texto claro, que tem tamanho de exatamente dois blocos do AES (32 bytes). No modo ECB, cada bloco é independente dos anteriores e não influencia seus sucessores. Por isto, na linha 07, o bloco do bit corrompido é perdido e o bloco seguinte é preservado.

O modo CBC faz o XOR do bloco processado pelo algoritmo com o bloco anterior ou com o IV, se for o primeiro bloco. Por isto, na linha 11, não apenas o bloco corrompido é completamente perdido, mas também o byte do segundo bloco que corresponde ao byte corrompido do primeiro bloco. Nos modos CFB, OFB e CTR, um fluxo de chaves é gerado pela encriptação encadeada (de maneiras específicas) da sequência de IVs. O XOR entre o fluxo de chaves e o texto claro produz o criptograma, e vice-versa. Na linha 15, com CFB, apenas o byte corrompido do primeiro bloco é perdido, já o bloco seguinte inteiro é perdido. Os modos OFB e CTR são análogos, em que apenas o byte corrompido do primeiro bloco é perdido e mais nada.

Em relação à perda de sincronismo, no modo ECB, a perda de um bloco do criptograma não afeta a decriptação dos blocos seguintes. Nos modos CBC e CFB, a perda de um bloco inviabiliza a decriptação apenas do bloco seguinte. Nos modos OFB e CTR, a perda e um bloco inviabiliza a decriptação de todos os blocos seguintes.

Saída 1: Comportamento dos modos de operação quando 1 bit do criptograma é modificado.

```
01   Chave      : 0123456789ABCDEF0123456789ABCDEF
02   iv         : ABCDEF1234567890ABCDEF1234567890
03   Texto claro: Modo de operacaoModo de operacao
04
05   Teste 1: AES/ECB/NoPadding
06   Criptograma: D005B98ACDC054C81666DB6B2EDF8D8BD004B98ACDC054C81666DB6B2EDF8D8B
07   Texto claro: ??????????????Modo de operacao
08
09   Teste 2: AES/CBC/NoPadding
10   Criptograma: 4A7F495EE367615DFC107C6B1A5589C70940086079FDB9D307D044C2E017D8D7
11   Texto claro: ??????????????Mndo de operacao
12
13   Teste 3: AES/CFB/NoPadding
14   Criptograma: F8241BBA339DC359EFA5ACF0DDB177583DBD525C351AA7388B95ADBF9E001926
15   Texto claro: Mndo de operacao??????????????
16
17   Teste 4: AES/OFB/NoPadding
18   Criptograma: F8241BBA339DC359EFA5ACF0DDB17758858195019B5F22A3D4FC3366EDC9095F
19   Texto claro: Mndo de operacaoModo de operacao
20
21   Teste 5: AES/CTR/NoPadding
22   Criptograma: F8241BBA339DC359EFA5ACF0DDB1775861AD5E88AE385B452C01C82A18A68E33
23   Texto claro: Mndo de operacaoModo de operacao
```

1.3.5. Verificação de integridade e autenticação de mensagem

O uso de encriptação somente não é capaz de garantir que o criptograma não foi corrompido, maliciosamente modificado, ou até mesmo completamente substituído por Ivo, quando em trânsito de Ana até Beto. Um valor de *hash* calculado sobre o criptograma poderia ser enviado por Ana junto com o criptograma e verificado por Beto. Deste modo, as corrupções acidentais ou maliciosas do criptograma poderiam ser detectadas por Beto. Mas Ivo ainda seria capaz de substituir o criptograma e o *hash*. Este problema é resolvido pelo uso de uma *tag* de autenticação no lugar do *hash*. Um código de autenticação de mensagem (MAC) é um mecanismo criptográfico que produz um selo (*tag*) de autenticidade baseado em uma chave compartilhada entre Ana e Beto e, por isto, relativa apenas a eles. A função de MAC recebe como entrada a chave simétrica e a mensagem e, geralmente, utiliza em seu algoritmo funções de encriptação ou de *hash*, produzindo a *tag*. A verificação da *tag* é feita pela comparação da *tag* recebida com uma nova *tag* calculada no recebimento da mensagem.

O trecho de código do Programa 3 mostra a utilização combinada de MAC e encriptação simétrica e está divido em três partes. As linhas de 01 a 06 contêm as configurações comuns do sistema criptográfico. As linhas de 08 a 11 mostram a encriptação e a geração da *tag* de autenticação pela Ana. A decriptação com a verificação por Beto da *tag* de autenticação está nas linhas de 13 a 16. A linha 02 mostra, em hexadecimal, o segredo compartilhado por Ana e Beto. A linha 03 produz a chave AES a partir do segredo. A linha 04 identifica o MAC como "HMACSHA256" [30], uma função de MAC baseada no *hash* (HMAC) da família SHA de 256 bits [29], e cria a chave correspondente. As linhas 05 e 06 instanciam o AES/ECB e o MAC.

A encriptação com *tag* de autenticação da mensagem ocorre do seguinte modo. Na linha 08, o MAC e o encriptador são inicializados com suas chaves, o criptograma é gerado na linha 10 e, na linha 11, a *tag* é calculada sobre o criptograma. A decriptação por Beto com verificação da *tag* ocorre assim: na linha 14, o MAC e o decriptador são inicializados com suas chaves, o criptograma é decriptado na linha 15 e, na linha 16, a *tag* recebida é comparada à *tag* calculada. A segurança do HMAC depende da função de *hash* e é dada pela metade de seu tamanho em bits. O HMACSHA256 tem *tag* de 256 bits e segurança de 128 bits; por isto as chaves do MAC e do AES também têm 128 bits.

Programa 3: MAC e encriptação com HMACSHA256 e AES/ECB/PKCS7Padding, 128 bits.

```
01  // configurações do sistema criptográfico para Ana e Beto
02  byte[] k = U.x2b("0123456789ABCDEF0123456789ABCDEF");
03  SecretKeySpec sks1 = new SecretKeySpec(k, "AES");
04  SecretKeySpec sks2 = new SecretKeySpec(k, "HMACSHA256");
05  Cipher c = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
06  Mac m = Mac.getInstance("HMACSHA256", "BC");
07
08  // encriptação pela Ana com tag de autenticação da mensagem
09  m.init(sks2); c.init(Cipher.ENCRYPT_MODE, sks1);
10  byte[] criptograma = c.doFinal(textoclaroAna.getBytes());
11  byte[] tag = m.doFinal(criptograma);
12
13  // decriptação pelo Beto com verificação da tag
14  m.init(sks2); c.init(Cipher.DECRYPT_MODE, sks1);
15  byte[] textoclaroBeto = c.doFinal(criptograma);
16  boolean ok = MessageDigest.isEqual(m.doFinal(criptograma),tag);
```

1.3.6. Encriptação autenticada

A encriptação autenticada combina em uma única função criptográfica as funções de encriptação e de MAC. A união das duas funções em uma única rotina tem duas vantagens principais. A primeira é simplificar a programação do software criptográfico pelos programadores que, neste caso, utilizam uma API de mais alto nível. A segunda é evitar a combinação incorreta de encriptação e MAC (tratada na próxima seção). A encriptação autenticada não é apenas o encapsulamento das funções de encriptação e autenticação por uma API de mais alto nível. De fato, ela é uma função criptográfica nova que atende aos dois objetivos (encriptação e autenticação) simultaneamente. A função de encriptação autenticada é geralmente materializada por modos de operação específicos para encriptadores de bloco conhecidos, tais como os modos de operação GCM [27] do AES e CCM, aplicável a qualquer encriptador de bloco.

O trecho de código do Programa 4 mostra a encriptação autenticada com AES/GCM e está divido em três partes. As linhas de 01 a 04 contêm as configurações comuns do sistema criptográfico. As linhas de 06 a 09 mostram a encriptação autenticada com dados autenticados por Ana. A decriptação com a verificação por Beto está nas linhas de 11 a 15. A linha 02 já e conhecida e cria uma chave para o AES a partir de um segredo. A linha 03 constrói a estrutura para os parâmetros do modo GCM, o IV e o tamanho da *tag* de autenticação, com 128 bits (*tags* de 96 bits também são válidas). A linha 04 instancia o AES/GCM sem um *padding* explícito. O modo GCM é baseado no modo CTR e, por isto, segue as mesmas restrições do CTR para IVs únicos e não requer um *padding*, já que se comporta como um encriptador de fluxo.

O encriptador é inicializado com a chave e os parâmetros da encriptação autenticada na linha 07. A linha 08 mostra como dados adicionais que somente serão autenticados (AAD), mas não encriptados, são inseridos no encriptador pelo método `updateAAD()`. O criptograma é computado como de costume (método `doFinal()`) na linha 09. A *tag* de autenticação com 16 bytes é implicitamente anexada ao final do criptograma. Na linha 12, o encriptador e inicializado para decriptação autenticada e, na linha 13, ele recebe os dados adicionais autenticados, geralmente conhecidos por Ana e Beto. Na linha 14, a decriptação acontece com a verificação implícita e obrigatória da *tag*, que se for incorreta ou inválida, lança uma exceção especifica (linha 15) e impede a decriptação do criptograma, inviabilizando o uso de um texto claro não autenticado.

Programa 4: Encriptação autenticada com AES/GCM.

```
01  // configurações do sistema criptográfico para Ana e Beto
02  SecretKeySpec ks = new SecretKeySpec(k, "AES");
03  GCMParameterSpec gps = new GCMParameterSpec(128, iv);
04  Cipher c = Cipher.getInstance("AES/GCM/NoPadding", "BC");
05
06  // Encriptação pela Ana
07  c.init(Cipher.ENCRYPT_MODE, ks, gps);
08  c.updateAAD("AAD nao estah cifrado...".getBytes());
09  byte[] criptograma = c.doFinal(textoClaroAna.getBytes());
10
11  // decriptação pelo Beto
12  c.init(Cipher.DECRYPT_MODE, ks, gps);
13  c.updateAAD("AAD nao estah cifrado...".getBytes());
14  try {textoclaroBeto = c.doFinal(criptograma);}
15  catch (AEADBadTagException e) {ok = false;}
```

1.3.7. Criptografia baseada em senhas

Conforme já discutido anteriormente (Seção 1.2), a criptografia baseada em senhas (*Password-Based Encryption* - PBE) [21] oferece uma solução simples para um dos problemas mais comuns relacionados à gestão de chaves criptográficas em softwares criptográficos, a guarda segura de chaves criptográficas. A encriptação de chaves com chaves geradas a partir de senhas requer cuidados adicionais na escolha de senhas suficientemente fortes e outros parâmetros adequados. Em particular, uma senha ruim pode comprometer a segurança de todo o sistema criptográfico.

Senhas são normalmente menos seguras (menores, menos variadas, ou mais previsíveis) que chaves criptográficas. Por sito, a obtenção de uma chave boa a partir de uma senha qualquer (preferencialmente boa) requer decisões adequadas de segurança como, por exemplo, a escolha de uma boa função de derivação de chave (*Key Derivation Function* - KDF). A KDF utilizada pelo mecanismo PBE geralmente necessita, além da escolha de uma senha forte, da configuração de dois parâmetros de segurança: *(i)* um *nonce*, um número pseudoaleatório de uso único com a função de *salt* no KDF e *(ii)* um contador de iterações para a quantidade de vezes que a função criptográfica interna ao KDF será iterada sobre si mesma.

O trecho de código do Programa 5 mostra a utilização programática da API de PBE. Nas linhas 02 e 03, a senha e o *salt* são definidos. Na linha 04, a estrutura de parâmetros do PBE é instanciada e o contador é configurado para 2048. A boa prática de segurança estabelece um contador de pelo menos 1000 e um *salt* com pelo menos 32 bits [15]. As linhas 05 e 06 são bastante densas e estabelecem, de fato, a instância do PBE/KDF utilizado na derivação da chave criptográfica: um PBE sobre o *hash* SHA1 para uma chave do AES de 128 bits no modo CBC. Na linha 07, a chave é finalmente derivada de acordo com os parâmetros estabelecidos.

Uma vez tendo sido derivada, a chave pode ser utilizada normalmente conforme estabelecido no processo de derivação. No programa exemplo, a linha 08 cria um encriptador AES/CBC, que será utilizado por Ana para encriptação (linhas de 10 a 12) e por Beto para decriptação (linhas de 14 a 16) com a chave derivada. Neste caso, o segredo compartilhado entre Ana e Beto é a senha.

Programa 5: Criptografia baseada em senhas com PBEWithSHA1And128BitAES-CBC-BC.

```
01  // configurações do PBE comuns para Ana e Beto
02  char[] senha = "5enha!23".toCharArray();
03  byte[] salt = U.x2b("1234567890ABCDEF");
04  PBEKeySpec pbeks = new PBEKeySpec(senha, salt, 2048);
05  SecretKeyFactory skf = SecretKeyFactory.getInstance(
06                         "PBEWithSHA1And128BitAES-CBC-BC", "BC");
07  Key sk = skf.generateSecret(pbeks);
08  Cipher c = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
09
10  // Encriptação pela Ana
11  c.init(Cipher.ENCRYPT_MODE, sk);
12  byte[] criptograma = c.doFinal(textoclaroAna.getBytes());
13
14  // decriptação pelo Beto
15  c.init(Cipher.DECRYPT_MODE, sk);
16  byte[] textoclaroBeto = c.doFinal(criptograma);
```

1.3.8. Encriptação e decriptação com chave assimétrica

Tradicionalmente, o algoritmo criptográfico assimétrico mais conhecido e utilizado para encriptação é o RSA, cujo nome é formado pelas letras iniciais dos sobrenomes dos autores Ron Rivest, Adi Shamir e Leonard Adleman. Para promover segurança e interoperabilidade, o uso e a implementação do RSA devem obedecer a padrões internacionais. O documento PKCS#1 v2.0 [22] especifica o *Optimal Asymmetric Encryption Padding* (OAEP) como um mecanismo de *padding*, que transforma o RSA em um mecanismo de encriptação assimétrica aleatorizado chamado RSA-OAEP.

As chaves criptográficas do RSA são muito grandes se comparadas às chaves de até 256 bits dos algoritmos simétricos, como o AES. Atualmente, tamanhos de chave RSA considerados seguros são 2048 bits ou 3072 bits, com um nível de segurança comparável ao dos algoritmos simétricos de 256 bits. A aritmética mais complexa e os tamanhos de chave elevados impõem ao RSA restrições de desempenho e de espaço.

O RSA-OAEP limita o tamanho do texto claro que pode ser encriptado em uma única chamada da função. Este limite de tamanho está relacionado ao tamanho do corpo finito (e da chave) usado na aritmética modular subjacente ao algoritmo e pode ser determinado, em bytes, pela fórmula *(ks-2*hs)/8–2*, onde *ks* é o tamanho da chave RSA em bits e *hs* é o tamanho do *hash* em bits usado pelo *padding* OAEP. Por exemplo, o RSA-OAEP com chave de 2048 bits e *hash* de 256 bits pode encriptar de uma vez um texto claro com até 158 bytes (1264 bits). O criptograma terá o tamanho da chave, que neste exemplo é de 256 bytes.

O trecho de código do Programa 6 mostra a configuração e uso do RSA-OAEP. A linha 02 instancia um gerador de par de chaves para o RSA. Na linha 03, o gerador é configurado para chaves de 2048 bits e o par de chaves é criado na linha 04. Nas linhas 07 e 08, a estrutura de parâmetros OAEP é criada com a função de mascaramento MGF1, o *hash* SHA-256 e a fonte de números primos padrão. As linhas 09 e 10 instanciam o RSA-OAEP com SHA256 e MGF1, que será utilizado por Ana para encriptação com a chave pública de Beto (linhas 13 e 14) e por Beto para decriptação com sua chave privada (linhas 17 e 18). Aqui, Ana já conhecia a chave pública de Beto.

Programa 6: Encriptação e decriptação assimétricas com RSA-OAEP.

```
01  // Beto cria um par de chaves
02  KeyPairGenerator g = KeyPairGenerator.getInstance("RSA", "BC");
03  g.initialize(2048);
04  KeyPair kp = g.generateKeyPair();
05
06  // configurações comuns para Ana e Beto
07  OAEPParameterSpec OAEPps = new OAEPParameterSpec("SHA-256",
08     "MGF1",MGF1ParameterSpec.SHA256, PSource.PSpecified.DEFAULT);
09  Cipher c = Cipher.getInstance(
10                 "RSA/None/OAEPwithSHA256andMGF1Padding", "BC");
11
12  // Encriptação pela Ana com a chabe pública de Beto
13  c.init(Cipher.ENCRYPT_MODE, kp.getPublic(), OAEPps);
14  byte[] criptograma = c.doFinal(textoclaroAna.getBytes());
15
16  // Decriptação pelo Beto com sua chave privada
17  c.init(Cipher.DECRYPT_MODE, kp.getPrivate(), OAEPps);
18  byte[] textoclaroBeto = c.doFinal(criptograma);
```

1.3.9. Transporte de chave simétrica com encriptação assimétrica

As limitações de tamanho e de desempenho do RSA-OAEP podem ser minimizadas pela combinação deste algoritmo assimétrico com um algoritmo simétrico, como o AES, em um sistema criptográfico híbrido para transporte de chaves simétricas. O trecho de código do Programa 7 ilustra a encriptação de uma chave AES com RSA-OAEP.

O trecho de código é bastante semelhante aos anteriores, com as seguintes diferenças relevantes: as chaves RSA têm 3072 bits, o OAEP usa a função de *hash* SHA-512 e o AES está no modo CTR com *padding* PKCS#7 e chaves de 256 bits. Nesta configuração, o RSA encripta de uma única vez até 254 bytes, o que é mais do que suficiente para os 32 bytes da chave AES mais um bloco de *padding* PKCS#7.

Nas linhas 26 e 27, a chave secreta AES é encriptada por Ana com a chave RSA pública do Beto. Nas linhas 31 e 32, Beto decripta a chave secreta com sua chave RSA privada. A chave AES é restaurada (linha 35) e usada na decriptação (linhas 38 e 39).

Programa 7: Sistema criptográfico híbrido para transporte de chaves com RSA-OAEP e AES.

```
01  // Este é o par de chaves do Beto
02  KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA","BC");
03  kpg.initialize(3072);
04  KeyPair kp = kpg.generateKeyPair();
05
06  // configurações da criptografia assimétrica para Ana e Beto
07  OAEPParameterSpec OAEPps = new OAEPParameterSpec("SHA-512",
08     "MGF1",MGF1ParameterSpec.SHA512, PSource.PSpecified.DEFAULT);
09  Cipher x = Cipher.getInstance(
10                    "RSA/None/OAEPwithSHA512andMGF1Padding","BC");
11
12  // configurações da criptografia simétrica para Ana a e Beto
13  Cipher c = Cipher.getInstance("AES/CTR/PKCS7Padding", "BC");
14  IvParameterSpec ivps = new IvParameterSpec(iv);
15
16  // Chave secreta compartilhada entre Ana e Beto
17  KeyGenerator gAna = KeyGenerator.getInstance("AES", "BC");
18  gAna.init(256);
19  Key skAna = gAna.generateKey();
20
21  // Ana encripta alguma mensagem ...
22  c.init(Cipher.ENCRYPT_MODE, skAna, ivps);
23  byte[] criptograma = c.doFinal(textoclaroAna.getBytes());
24
25  // Encriptação por Ana da chave secreta com chave pública do Beto.
26  x.init(Cipher.ENCRYPT_MODE, kp.getPublic(), OAEPps);
27  byte[] chaveEncriptada = x.doFinal(skAna.getEncoded());
28
29  // Aqui começa a parte do Beto
30  // decriptação da chave secreta pelo Beto com a sua chave privada
31  x.init(Cipher.DECRYPT_MODE, kp.getPrivate(), OAEPps);
32  byte[] chaveBytes = x.doFinal(chaveEncriptada);
33
34  //recuperacao da chave secreta transportada
35  SecretKeySpec skBeto = new SecretKeySpec(chaveBytes, "AES");
36
37  //decriptando alguma coisa com a chave secreta
38  c.init(Cipher.DECRYPT_MODE, skBeto, ivps);
39  byte[] textoClaroBeto = c.doFinal(criptograma);
```

1.3.10. Assinaturas digitais e verificação de autenticidade

O trecho de código do Programa 8 mostra a utilização de dois sistemas criptográficos para assinaturas digitais. O primeiro é o RSA *Probabilistic Signature Scheme* (RSA-PSS), a padronização do RSA [22] para assinaturas digitais aleatorizadas. O segundo é o padrão norte-americano para assinaturas digitais sobre curvas elípticas, o ECDSA [25]. Os dois sistemas podem ser usados de modo alternativo, desde que a assinatura seja gerada e verificada pelo mesmo sistema.

O RSA-PSS foi configurado com SHA-256, MGF1 e chave de 3072 bits, produzindo assinaturas de 384 bytes. O ECDSA foi configurado com SHA-256 e a curva elíptica sobre corpo primo "`prime256v1`", produzindo uma assinatura digital de 70 bytes. A assinatura de tamanho reduzido e a maior velocidade de processamento, tanto na geração do par de chaves quanto na assinatura digital favorecem a tecnologia de curvas elípticas nas situações em que há restrições de espaço e de desempenho. O par de chaves do ECDSA também é consideravelmente menor em relação ao do RSA-PSS.

O par de chaves de Ana é gerado na linha 15. A geração de uma assinatura digital com a chave privada da Ana acontece nas linhas de 18 a 20. A verificação por Beto da autenticidade do documento com a assinatura de Ana, com a chave pública de Ana, ocorre nas linhas de 31 a 33. Beto já conhecia previamente a chave pública de Ana.

Programa 8: Dois algoritmos para assinaturas digitais: RSA-PSS e ECDSA.

```
01  // par de chaves de Ana e configurações do criptossistema
02  Signature sAna = null, vBeto = null;
03  KeyPairGenerator kpg = KeyPairGenerator.getInstance(alg,"BC");
04  switch (alg) {
05    case "RSA":
06      kpg.initialize(3072, new SecureRandom());
07      sAna = Signature.getInstance("SHA256withRSAandMGF1","BC");
08      break;
09   case "ECDSA":
10      ECGenParameterSpec ec= new ECGenParameterSpec("prime256v1");
11      kpg.initialize(ec, new SecureRandom());
12      sAna = Signature.getInstance("SHA256WithECDSA", "BC");
13      break;
14  }
15  KeyPair kpAna = kpg.generateKeyPair();
16
17  //Ana assina o documento
18  sAna.initSign(kpAna.getPrivate(), new SecureRandom());
19  assinadorAna.update(documento.getBytes);
20  byte[] assinatura = sAna.sign();
21
22  switch (alg) {
23    case "RSA":
24      vBeto = Signature.getInstance("SHA256withRSAandMGF1","BC");
25      break;
26    case "ECDSA":
27      vBeto = Signature.getInstance("SHA256WithECDSA", "BC");
28      break;
29  }
30  //Beto verifica a assinatura
31  vBeto.initVerify(kpAna.getPublic());
32  vBeto.update(documento.getBytes());
33  boolean ok = vBeto.verify(assinatura);
```

1.3.11. Aleatoriedade e geração de números pseudoaleatórios

O trecho de código do Programa 9 mostra a utilização de geradores de números pseudoaleatórios (*PseudoRandom Number Generators* - PRNGs). Em particular, o SHA1PRNG disponível no provedor criptográfico padrão do JDK é testado em relação a duas propriedades, a dispersão ou distribuição dos valores por um intervalo determinado e a previsibilidade da sequência pseudoaleatória gerada a partir de uma semente conhecida. Os PRNGs adequados para uso em sistemas criptográficos são ditos criptograficamente seguros. No Java, eles são objetos da classe SecureRandom.

Nas linhas de 02 a 04, as três instâncias do SHA1PRNG são criadas de modo independente. Nas linhas 06 a 08, as três instâncias produzem cada uma 100 valores inteiros no intervalo de 0 a 10 mil. O gráfico de dispersão da Figura 14(A) mostra os valores obtidos. Observa-se que os valores gerados pelas três instâncias estão distribuídos uniformemente pelo intervalo de 0 a 10 mil. Além disso, não houve repetição de valores nas sequências pseudoaleatórias. Este é o comportamento esperado de um PRNG criptograficamente útil. Testes mais rigorosos podem ser feitos [5].

Nas linhas 10 e 11, mais duas instâncias são criadas, mas desta vez elas são configuradas com a mesma semente pseudoaleatória (linha 12). Nas linhas 14 e 15, as duas instâncias produzem cada uma 100 valores inteiros no intervalo de 0 a 10 mil. O gráfico de dispersão da Figura 14(B) mostra os valores obtidos. Observa-se que as sequências pseudoaleatórias são idênticas. Para a mesma semente, o PRNG sempre produz a mesma sequência de valores. Este também é o comportamento esperado.

Programa 9: Geração de números pseudoaleatórios com SHA1PRNG.

```
01  // Teste de dispersão estatística
02  SecureRandom sr1 = SecureRandom.getInstance("SHA1PRNG", "SUN");
03  SecureRandom sr2 = SecureRandom.getInstance("SHA1PRNG", "SUN");
04  SecureRandom sr3 = SecureRandom.getInstance("SHA1PRNG", "SUN");
05  System.out.println("i , sr1 , sr2, sr3");
06  for (int i = 0; i < 100; i++) {
07    U.println(i+","+sr1.nextInt(10000)+","+sr2.nextInt(10000)+","
08                                          +sr3.nextInt(10000));}
09  //Teste de imprevisibilidade
10  SecureRandom sr4 = SecureRandom.getInstance("SHA1PRNG", "SUN");
11  SecureRandom sr5 = SecureRandom.getInstance("SHA1PRNG", "SUN");
12  byte[] s = sr4.generateSeed(32); sr4.setSeed(s); sr5.setSeed(s);
13  System.out.println("i , sr4 , sr5");
14  for (int i = 0; i < 100; i++) {
15    U.println(i+","+sr4.nextInt(10000)+","+sr5.nextInt(10000));}
```

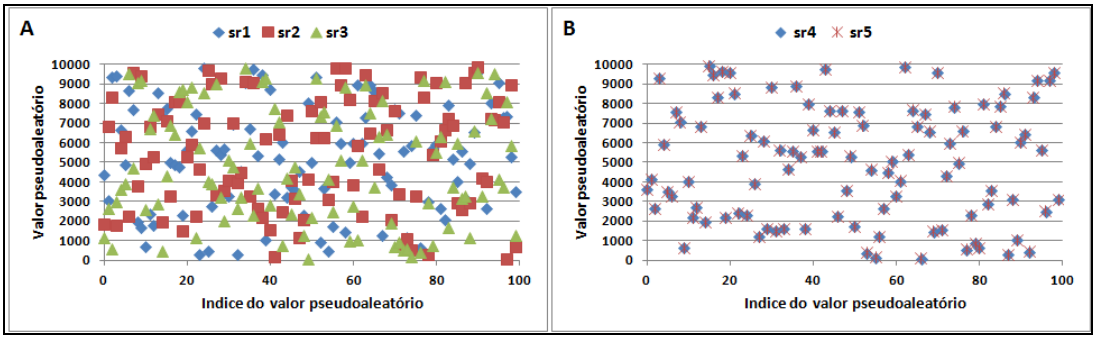


Figura 14: Testes do SHA1PRNG: (A) distribuição estatística e (B) imprevisibilidade.

1.3.12. Validação de certificados digitais

Certificação digital é um tema amplo e seu tratamento aprofundado tomaria todo o espaço deste texto, talvez merecendo até um capítulo exclusivo. Esta seção limita-se aos aspectos de validação de certificados digitais considerados boas práticas [7][8][28].

Um certificado digital de chave pública, ou simplesmente certificado, é um documento digital que dá como verdadeiro o vínculo entre uma chave pública autêntica e uma entidade cujo nome está no certificado. A veracidade do certificado é garantida por uma terceira parte confiável emissora do certificado, chamada de Autoridade Certificadora (CA). O certificado contém a assinatura digital da CA emissora e várias informações, tais como a chave pública, a identidade da entidade reconhecida pela CA, datas de início de uso e de validade (final de uso), etc. Por isto, o certificado é usado na verificação de que uma chave pública pertence a uma entidade e serve também como meio confiável para distribuição de chaves públicas.

A validação do certificado é realizada toda vez que a autenticidade da chave pública contida nele deve ser verificada. A assinatura da AC pode ser verificada por qualquer um com acesso à chave pública da AC, cujo certificado é amplamente disponível. Por exemplo, num caso bastante comum, uma AC pode emitir certificados de servidores web. Quando um software cliente HTTPS (browser) faz uma requisição para um servidor web protegido, o servidor responde com seu certificado digital. O software cliente valida o certificado do servidor verificando a assinatura da AC emissora sobre a chave pública do servidor e outros parâmetros do certificado. Se o cliente já não possuir a chave pública da AC, ele vai buscá-la (em um repositório de chaves públicas da AC). Se o certificado é válido, então o cliente sabe que o servidor é autêntico.

A validação do certificado (e da chave pública contida nele) abrange mais etapas do que a mera verificação da assinatura da AC. Além da verificação da assinatura, o software de verificação precisa verificar se o certificado não atingiu o final de seu período de validade, se não foi revogado e se o nome constante no certificado é o

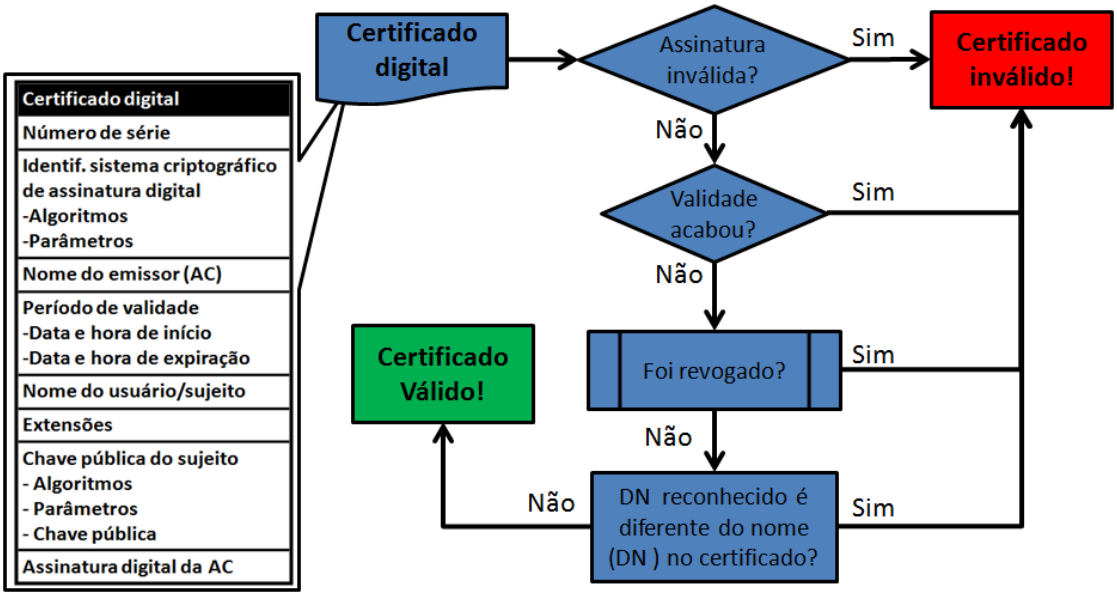


Figura 15: Fluxograma de validação de um certificado digital.

mesmo da parte que alega ser a dona da chave pública. O nome também deve ser obtido de um terceiro confiável, por exemplo, de um serviço de DNS, no caso de nomes de domínio. A validação do certificado é ilustrada no fluxograma da Figura 15.

A verificação da assinatura da AC em um certificado digital exige a chave pública da AC. Para ser confiável, a chave pública da AC deve estar contida em um certificado assinado por outra AC ou autoassinado, se for uma CA raiz. As verificações sucessivas de uma sequência de assinaturas constroem uma hierarquia de certificados. Os certificados na base da hierarquia são assinados pelas ACs de mais baixo nível, cujos certificados são assinados pelas ACs intermediárias, que têm seus certificados assinados pelas ACs de alto nível, cujos certificados são assinados pela AC raiz, que tem seus certificados autoassinados. A Figura 16 ilustra esta cadeia de certificação.

Uma AC revoga um certificado nas seguintes situações: quando ocorre erro na emissão do certificado (nome grafado errado), ou o certificado foi emitido para uso de um serviço e o portador não tem mais acesso a ele (demissão de um funcionário), ou a chave privada do portador foi comprometida, ou ainda a chave privada da AC foi comprometida (uma situação extrema). Uma Lista de Certificados Revogados (LCR) é o documento assinado digitalmente pela AC que lista o número de série de todos os certificados, ainda não expirados, que perderam a utilidade por algum dos motivos acima. Um software criptográfico pode consultar um serviço de LCR para receber atualizações periódicas, em intervalos regulares definidos por procedimentos, ou ainda consultar em tempo real se um certificado foi revogado ou não. Porém, a instabilidade de comunicação pode causar indisponibilidade do serviço de validação em tempo real.

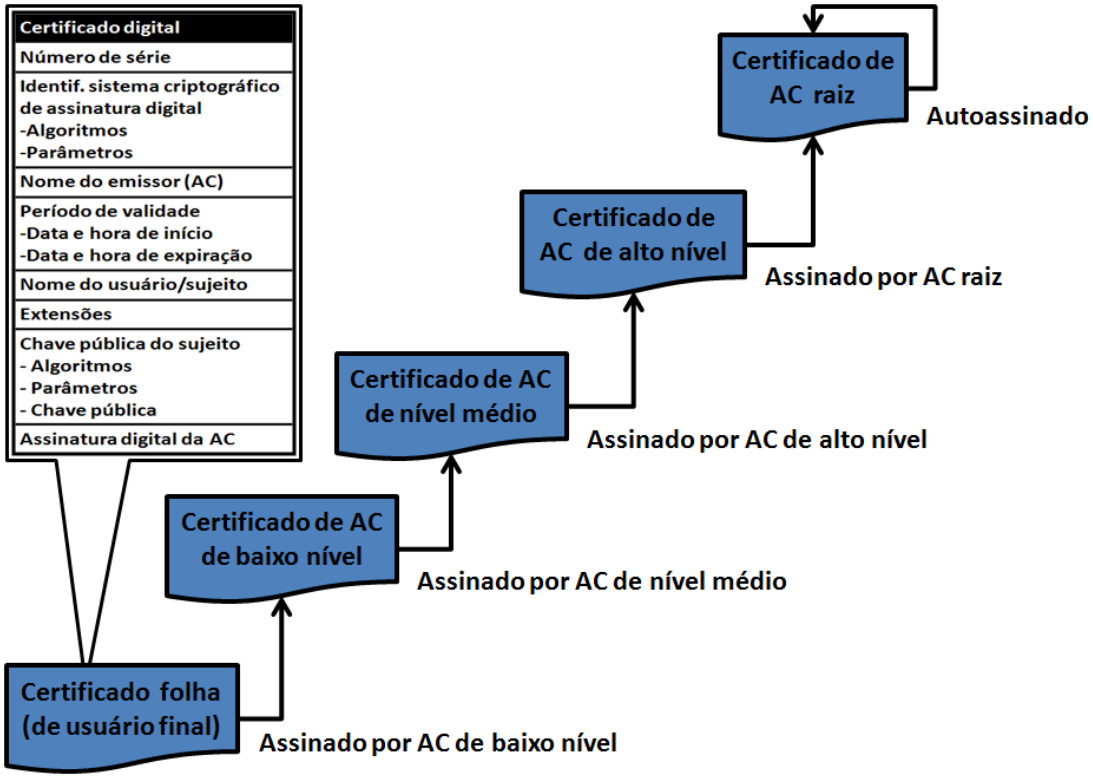


Figura 16: Cadeia hierárquica de certificação digital.

1.4. Como não usar a criptografia: reconhecendo os maus usos

Esta seção é organizada em torno dos maus usos de programação de criptografia que levam a vulnerabilidades em softwares criptográficos [13][17][18] e que têm sido estudados recentemente [14][15][20]. Os maus usos são exemplificados por casos reais e/ou ilustrados programaticamente por meio de programas em Java. Os maus usos comuns de criptografia tratados no curso são os seguintes: criptografia determinística simétrica e assimétrica, IVs fixos ou reutilizados, sementes fixas ou reutilizadas para PRNGs, troca indevida de modos de operação, combinação de integridade e encriptação, reutilização de chaves em encriptadores de fluxo, maleabilidade dos encriptadores de fluxo, *padding oracles* e validação incorreta de certificados digitais. A utilização de chaves fracas (pequenas) e de algoritmos obsoletos ou quebrados não é tratada aqui como um mau uso programático, mas sim uma configuração mal feita.

1.4.1. Criptografia determinística simétrica

A criptografia determinística simétrica é caracterizada pelo uso indiscriminado do modo de operação ECB juntamente com um encriptador simétrico, com ou sem *padding*. A utilização do modo ECB é considerada um mau uso por que, quando é utilizada inadvertidamente, pode levar a revelação indevida de informação pela identificação de padrões do texto claro no criptograma. Neste caso, a facilidade com que a criptografia determinística pode ser utilizada também favorece o seu mau uso.

Em Java, este mau uso pode ser identificado já na instanciação do algoritmo criptográfico, por exemplo, pelo método *getInstance(a)*, da classe *Cipher*, em que *a* é o nome do algoritmo. Há três opções para resolver o nome do algoritmo *a* com o modo ECB: *(i)* apenas o nome do algoritmo, por exemplo, *"AES"*; *(ii)* nome do algoritmo e modo sem *padding*, por exemplo, *"AES/ECB/NoPadding"*; e *(iii)* nome do algoritmo e modo com *padding*, por exemplo, *"AES/ECB/PKCS7Padding"*. Vale observar que a primeira opção leva naturalmente ao erro, uma vez que, na falta de uma escolha explícita feita pelo programador, o modo ECB é a opção padrão implícita. A Saída 2 mostra o resultado da encriptação do texto claro *"Deterministica.."* nas três opções de configuração do modo ECB, com o algoritmo AES e a mesma chave criptográfica. Na linha 04 pode ser observado que a encriptação do mesmo texto claro com os mesmos parâmetros de segurança produz o mesmo criptograma com *padding* que as outras opções de encriptação determinística (mostradas nas linhas 08 sem *padding*, e 11 com *padding* explícito). O algoritmo AES foi utilizado apenas como ilustração; o mesmo resultado seria obtido com outros encriptadores de bloco nas mesmas configurações de chave e modo de operação. Vale lembrar que o modo ECB não necessita de IV.

Saída 2: Criptografia determinística com AES no modo ECB.

```
01   Texto claro: Deterministica..
02   Chave   AES: C64539B4A9E992A077170C413FC02EB2
03
04   Encriptado com: AES
05   Criptograma: EAB196E34C4ED4B8C4261CACC243AC5E507D286A609456C69DA7CE68C844B89E
06
07   Encriptado com: AES/ECB/NoPadding
08   Criptograma: EAB196E34C4ED4B8C4261CACC243AC5E
09
10   Encriptado com: AES/ECB/PKCS7Padding
11   Criptograma: EAB196E34C4ED4B8C4261CACC243AC5E507D286A609456C69DA7CE68C844B89E
```

1.4.2. Criptografia determinística assimétrica

A criptografia determinística assimétrica é caracterizada pelo uso indiscriminado do algoritmo RSA na sua forma canônica, isto é, sem aleatorização. Analogamente ao modo ECB dos encriptadores simétricos de bloco, a utilização do algoritmo RSA sem *padding* aleatorizado é considerada um mau uso porque, quando é utilizada inadvertidamente, pode levar a revelação indevida de informação pela identificação de padrões do texto claro no criptograma. Mais uma vez, a facilidade com que a criptografia determinística pode ser utilizada também favorece o seu mau uso [17].

Em Java, este mau uso pode ser identificado já na instanciação do algoritmo criptográfico, por exemplo, pelo método *getInstance(a)*, da classe *Cipher*, em que *a* é o nome do algoritmo. Há três opções para resolver o nome do algoritmo *a* para o RSA canônico: *(i)* apenas o nome do algoritmo, por exemplo, *"RSA"*; *(ii)* nome do algoritmo e modo ECB sem *padding*, por exemplo, *"RSA/ECB/NoPadding"*; e *(iii)* nome do algoritmo, sem modo e sem *padding*, por exemplo, *"RSA/None/NoPadding"*. Vale observar que a primeira opção leva naturalmente ao erro, uma vez que, na falta de uma escolha explícita, o RSA canônico é a opção padrão implícita.

A Saída 3 mostra o resultado da encriptação do texto claro *"Cripto deterministica"* nas três opções de configuração do RSA canônico e a mesma chave criptográfica de 512 bits (uma chave pequena útil somente em exemplos). Este tamanho de chave foi escolhido apenas para facilitar a apresentação do resultado e não tem relação com este mau uso. Na linha 04 pode ser observado que a encriptação do mesmo texto claro, com a mesma chave, pelo algoritmo identificado por "RSA" produz o mesmo criptograma que as outras opções de encriptação determinística com RSA e são mostrados nas linhas 08, *"RSA/ECB/NoPadding"*, e 12, *"RSA/None/NoPadding"*. A Saída 3 também mostra as duas implementações de RSA aleatorizados disponíveis no provedor BouncyCastle: a versão mais antiga do padrão PKCS#1 identificada por "RSA/None/PKCS1Padding" e o PKCS#1 versão 2 conhecido como RSA-OAEP e identificado por "RSA/None/OAEPWithSHA1AndMGF1Padding". Ambos mostram criptogramas diferentes não apenas entre si, mas também dos anteriores.

Saída 3: Criptografia determinística assimétrica com RSA.

```
01   Texto claro: Cripto deterministica
02
03   Encriptado com: RSA
04   Criptograma: 6C77C92E46A4D7A110F3713D8B635BE7677E140CC607DEF342F10A8CD0AC00E7
05                E865439CF2501D6CDADAF8884CE4B61C6BA91E13225E5AB375A9DE2662059C3F
06
07   Encriptado com: RSA/ECB/NoPadding
08   Criptograma: 6C77C92E46A4D7A110F3713D8B635BE7677E140CC607DEF342F10A8CD0AC00E7
09                E865439CF2501D6CDADAF8884CE4B61C6BA91E13225E5AB375A9DE2662059C3F
10
11   Encriptado com: RSA/None/NoPadding
12   Criptograma: 6C77C92E46A4D7A110F3713D8B635BE7677E140CC607DEF342F10A8CD0AC00E7
13                E865439CF2501D6CDADAF8884CE4B61C6BA91E13225E5AB375A9DE2662059C3F
14
15   Encriptado com: RSA/None/PKCS1Padding
16   Criptograma: 6528EFA1281661E80D83728E5A1FEFDB55E45525DF9027C36784AF0BB375DB84
17                01767A45DD070B0A85FB31687785314A86C5DA89267259961EFF1D681891F08B
18
19   Encriptado com: RSA/None/OAEPWithSHA1AndMGF1Padding
20   Criptograma: 225EDA0DE625CF0F06400985FF1933F6F4457690F4513F4A0F4547B49A68F9C7
21                7CD600412BE0A4147D97DC030D3CAEF30FC47B8069A53540F818D36D61944ED6
```

1.4.3. IVs fixos ou reutilizados

O mau uso chamado de reutilização ou fixação dos Vetores de Inicialização (IVs) é caracterizado pela realização de mais de uma encriptação com o mesmo IV, para a mesma chave criptográfica. A repetição de IVs com a mesma chave é considerada um mau uso por que pode levar à identificação de padrões do texto claro no criptograma. Este mau uso é facilmente identificado em programas de computador quando um valor fixo para o IV está embutido no código fonte. Já ocorrências cuja identificação é mais desafiadora são aquelas em que o IV é obtido de modo indireto, por exemplo, de um arquivo ou banco de dados, ao ainda computado a partir de algoritmos determinísticos, como por exemplo, um PRNG de semente fixa.

A Saída 4 mostra o resultado da encriptação do texto claro *"Teste de IV fixoTeste de IV fixo"* de 32 bytes (dois blocos do AES), com a mesma chave e IV. Para cada modo de operação (CBC, OFB, CFB e CTR) foram feitas duas encriptações sem *padding*, para explicitar a repetição do padrão. As linhas 05 e 06 mostram a repetição do criptograma para o modo CBC. As linhas 10 e 11 mostram os criptogramas repetidos para o modo OFB. As repetições de criptograma para os modos CFB e CTR são mostradas nas linhas 14/15 e 18/19, respectivamente.

Um fato curioso dos encriptadores de bloco com comportamento de encriptador de fluxo (OFB, CFB e CTR) é que, para mesma chave e IV, o primeiro bloco do criptograma de cada um deles é idêntico ao primeiro bloco dos outros dois. Isto ocorre por que o criptograma do primeiro bloco é gerado simplesmente pelo XOR do texto claro com o IV encriptado com a chave, resultando sempre no mesmo valor quando há coincidência de parâmetros. Esta coincidência está marcada em vermelho na Saída 4.

Os modos de operação têm requisitos de geração e uso de IVs que precisam ser observados rigorosamente pelos programadores. Grosso modo, a boa prática dita que um IV nunca pode ser reutilizado com a mesma chave criptográfica no mesmo modo de operação. O modo CBC pode tolerar IVs pseudoaleatórios com pouquíssima chance de repetição. Os modos com comportamento de encriptadores de fluxo não toleram qualquer repetição de IV. A consequência da repetição é a revelação de mais informação do que padrões simples no criptograma.

Saída 4: IV fixo ou reutilizado por diversos modos de operação do AES.

```
01   Texto claro: Teste de IV fixoTeste de IV fixo
02   Chave      : 4A5349A8E49B4606746DC97BFB541384
03   IV fixo    : 0123456789ABCDEF0123456789ABCDEF
04
05   Encriptado com: AES/CBC/NoPadding
06   Criptograma1: 5064DB4A860AD38218AC16BE3E18344CBAD4FAE1E36AE256411E51577F5B7116
07   Criptograma2: 5064DB4A860AD38218AC16BE3E18344CBAD4FAE1E36AE256411E51577F5B7116
08
09   Encriptado com: AES/OFB/NoPadding
10   Criptograma1: 7ED81E3D99B68CC0CEB3CC19733087BFAB503D40DFE8F7F552D59356EC269EFC
11   Criptograma2: 7ED81E3D99B68CC0CEB3CC19733087BFAB503D40DFE8F7F552D59356EC269EFC
12
13   Encriptado com: AES/CFB/NoPadding
14   Criptograma1: 7ED81E3D99B68CC0CEB3CC19733087BF2D319CE0873444D1461EBD1B34952570
15   Criptograma2: 7ED81E3D99B68CC0CEB3CC19733087BF2D319CE0873444D1461EBD1B34952570
16
17   Encriptado com: AES/CTR/NoPadding
18   Criptograma1: 7ED81E3D99B68CC0CEB3CC19733087BFC9D6A25699F4877074723804A740086E
19   Criptograma2: 7ED81E3D99B68CC0CEB3CC19733087BFC9D6A25699F4877074723804A740086E
```

1.4.4. Sementes fixas ou reutilizadas para PRNGs

Já foi mostrado anteriormente na seção 1.3 que bons geradores de números pseudoaleatórios produzem sequências numéricas que se comportam, para a maioria dos usos práticos, como números aleatórios de fato. Porém, as sequências pseudoaleatórias são geradas por algoritmos determinísticos, os PRNGs, cujo fluxo de execução pode ser reproduzido a partir da repetição dos parâmetros de entrada. Quando PRNGs são utilizados na geração de IVs pseudoaleatórios, cuidados devem ser tomados para que não ocorra a repetição dos IVs por causa de uma configuração insegura como, por exemplo, o reuso de sementes fixas para o PRNG. Caso isto ocorra, um comportamento semelhante ao do reuso de IVs pode ser observado na geração dos criptogramas.

O Programa 10 ilustra a reutilização de IVs a partir da fixação de semente em um PRNG. O Programa 10 produz um resultado semelhante, mas não idêntico, ao da seção anterior, onde os IVs são reusados diretamente. As linhas 01 a 03 geram a chave AES de 128 bits. As linhas 04 e 05 criam um array com os quatro modos de operação CBC, OFB, CFB e CTR, sem *padding*. Para cada modo de operação, o laço iniciado na linha 06 faz o seguinte: cria duas instâncias (enc e dec) do AES no modo de operação em questão, cria um PRNG do tipo SHA1PRNG e o configura sempre com a mesma semente fixa, encripta o texto claro com a mesma chave e com o IV gerado internamente pelo PRNG, e decripta com a mesma chave e IV gerado pelo PRNG.

Visto que o SHA1PRNG tem na semente a única fonte de entropia para geração da sequência pseudoaleatória, a fixação da semente na linha 12 elimina a aleatoriedade da sequência, que será sempre a mesma. Este mau uso pode ser eliminado de duas maneiras: a primeira é simplesmente pela eliminação do comando *setSeed()* da linha 12, o que permitiria ao SHA1PRNG usar sementes diferentes em cada nova instância; a segunda é a substituição do SHA1PRNG por outro algoritmo que combine a semente com outras fontes de entropia, como é o caso do Windows-PRNG do sistema operacional Windows ou o /dev/urandom nos sistemas Linux. Uma terceira opção seria ainda combinar dois PRNGs, um para geração de semente e outro para geração da sequência pseudoaleatória.

Programa 10: Semente fixa ou reutilizada em PRNGs para geração de IVs.

```
01  KeyGenerator g = KeyGenerator.getInstance("AES", "BC");
02  g.init(128);
03  Key k = g.generateKey();
04  String[] aes = {"AES/CBC/NoPadding","AES/OFB/NoPadding",
05                  "AES/CFB/NoPadding","AES/CTR/NoPadding"};
06  for (int a = 0; a < aes.length; a++) {
07     Cipher enc = Cipher.getInstance(aes[a], "BC");
08     Cipher dec = Cipher.getInstance(aes[a], "BC");
09     byte[][] criptograma = new byte[2][];
10     for (int i = 0; i < 2; i++) {
11       SecureRandom sr = SecureRandom.getInstance("SHA1PRNG","SUN");
12       sr.setSeed(U.x2b("0123456789ABCDEF0123456789ABCDEF"));
13
14       enc.init(Cipher.ENCRYPT_MODE, k, sr);
15       criptograma[i] = enc.doFinal(textoClaroAna);
16
17       dec.init(Cipher.DECRYPT_MODE,k,new IvParameterSpec(enc.getIV()));
18       byte[] textoClaroBeto = dec.doFinal(criptograma[i]);
19  }}
```

1.4.5. Troca indevida de modos de operação

O mau uso por troca indevida de modos de operação é decorrente de dois mal-entendidos sobre a utilização de criptografia. O primeiro é a noção incorreta de que os modos de operação são intercambiáveis. Isto é, que um encriptador de fluxo como o AES/OFB poderia substituir (erroneamente), sem qualquer alteração no software criptográfico, um encriptador de bloco como o AES/CBC. O segundo mal-entendido está relacionado às consequências do reuso de chaves e IVs em encriptadores de fluxo.

Em um encriptador de fluxo, como AES/OFB, onde M é a mensagem em texto claro, C é o criptograma e o símbolo "^" é a operação lógica de OU-exclusivo (XOR), o fluxo de chaves K é obtido por uma cadeia de encriptações sucessivas do IV inicial. Neste encriptador de fluxo, obtém-se o criptograma como $K\text{^}M=C$ e a decriptação como $C\text{^}K=M$. No caso de dois criptogramas $C_1=K_1\text{^}M$ e $C_2=K_2\text{^}N$, obtém-se $C_1\text{^}C_2 = (K_1\text{^}M)\text{^}(K_1\text{^}N) = M\text{^}N$.

Se, por exemplo, as chaves são reusadas ($K_1 = K_2$) e o texto claro M possui partes fixas conhecidas, como é o caso das mensagens com cabeçalhos, então o texto claro N pode ser facilmente descoberto. A Saída 5 exemplifica a recuperação do texto claro N a partir da troca de um encriptador de bloco por um encriptador de fluxo em um sistema criptográfico que já estava mal configurado. Isto é, havia o reuso de IVs no modo CBC, cuja consequência é o reuso da chave no modo OFB. Na coluna da esquerda o texto claro é encriptado com AES/CBC, e na da direita com o AES/OFB.

No caso do CBC, a consequência do reuso de IVs para uma mesma chave pode ser menos prejudicial e geralmente pode passar despercebida por um longo tempo. Na coluna da esquerda da Saída 5, com modo CBC, a operação $C_0\text{^}C_1$ com M_0 conhecida não é capaz de revelar M_1 diretamente. Como pode ser observado pelo resultado da operação $C_0\text{^}C_1\text{^}M_0$ em caracteres não imprimíveis mostrados como "?". Por outro lado, na coluna da direita, com modo OFB, M_1 é revelada pela aplicação direta da operação $C_0\text{^}C_1\text{^}M_0$, que resulta em M_1. Este mau uso também pode ocorrer nos modos de operação CFB e CTR e encriptadores de fluxo verdadeiros como o RC4, podendo ser evitado pela utilização de um modo de encriptação autenticada, como o GCM com uma gestão rigorosa de IVs, conforme ilustrado na próxima seção.

Saída 5: Troca indevida de encriptador de bloco (AES/CBC) por um de fluxo (AES/OFB).

01	M[0] = "Troca a cifra de"	M[0] = "Troca a cifra de"
02	M[1] = "bloco por fluxo."	M[1] = "bloco por fluxo."
03		
04	K = 00112233445566778899AABBCCDDEEFF	K = 00112233445566778899AABBCCDDEEFF
05		
06	Encriptado com: AES/CBC/NoPadding	Encriptado com: AES/OFB/NoPadding
07		
08	C[0] =8C61DC63FF9682588910E6FF77866E58	C[0] =29CF058FFD0543D255888BC1A12F33DD
09		
10	iv[0]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C	iv[0]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
11		
12	C[1] =00D0625EC980B1FD965931601CE279C3	C[1] =1FD1058FF305529D44C18BDFB5773896
13		
14	iv[1]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C	iv[1]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
15		
16	C0^C1 = k^M0^K^M1 = M0^M1 =	C0^C1 = k^M0^K^M1 = M0^M1 =
17	=8CB1BE3D361633A51F49D79F6B64179B	=361E00000E00114F1149001E14580B4B
18		
19	M0^M1^M0 = M1 = ????????????????	M0^M1^M0 = M1 = bloco por fluxo.

1.4.6. Reutilização de chaves em encriptadores de fluxo

O reuso de chave em um encriptador de fluxo e a existência de partes fixas, como cabeçalhos, nos textos claros, podem viabilizar a descoberta de textos claros. Esta situação foi mostrada na seção anterior pelo reuso de IVs na geração do fluxo de chaves. Em um canal de comunicação bidirecional protegido por um encriptador de fluxo e uma chave secreta compartilhada, não basta que cada parte comunicante produza de modo único e imprevisível o seu IV. De fato, O IV deve ser único também no canal de comunicação e, por isto, combinado entre as partes para que não haja repetição. Em um caso recente [34], o mau uso de um encriptador de fluxo conhecido, o algoritmo RC4, em um protocolo de comunicação segura permitiu a decriptação de mensagens encriptadas trocadas entre dois usuários de um aplicativo de comunicação instantânea. A vulnerabilidade consistiu na reutilização do fluxo de chaves para encriptar mensagens nas duas direções do canal de comunicação protegido pelo encriptador de fluxo.

O trecho de código do Programa 11 mostra como Ivo pode descobrir o conteúdo de um criptograma de Beto a partir de uma mensagem indevidamente revelada por Ana em um sistema criptográfico mal configurado com repetição de IVs. As linhas de 01 a 07 contêm as configurações comuns a Ana e a Beto. Ana encripta nas linhas 09 e 10 e Beto encripta nas linhas 13 e 14. Ivo monitora a comunicação e lê os criptogramas de Ana e de Beto. Ivo (por exemplo, com engenharia social ou reversa) descobre o texto claro de Ana. Neste momento, Ivo usa as relações lógicas entre os criptogramas, a chave e os textos claros para recuperar o criptograma de Beto (linha 18).

Esta vulnerabilidade causada pelo mau uso do encriptador de fluxo pode ser evitada pela gestão compartilhada de IVs, em que cada parte comunicante gera o seu IV em um intervalo de valores diferente do utilizado pela outra parte, evitando deste modo a repetição do IV e o consequente reuso do fluxo de chaves. Por exemplo, em um IV de 128 bits, 1 bit pode identificar a parte comunicante (por exemplo, quem cria a chave usa o valor 1) e os 127 bits restantes podem ser incrementados por um contador. A chave deve ser trocada antes que qualquer lado atinja o limite do contador, evitando assim a repetição do IV para a mesma chave.

Programa 11: Reutilização de chaves em um encriptador de fluxo com AES/CTR.

```
01 // Configurações comuns para Ana e Beto
02 byte[][] M = {("Reuso de chave d").getBytes(),
03               ("a Cifra de fluxo").getBytes()};
04 byte[][] iv = {U.x2b("0123456789ABCDEF0123456789ABCDEF"),
05                U.x2b("0123456789ABCDEF0123456789ABCDEF")};
06 SecretKeySpec ks = new SecretKeySpec(k, "AES");
07 Cipher enc = Cipher.getInstance("AES/CTR/NoPadding", "BC");
08
09 // Ana encripta
10 enc.init(Cipher.ENCRYPT_MODE,ks,new IvParameterSpec(iv[0]));
11 C[0] = enc.doFinal(M[0]);
12
13 //Beto Encripta também
14 enc.init(Cipher.ENCRYPT_MODE, ks, new IvParameterSpec(iv[1]));
15 C[1] = enc.doFinal(M[1]);
16
17 // Ivo realiza o ataque
18 byte[] M0xorM1 = U.xor(C[0],C[1]); byte[] M1 = U.xor(M[0], M0xorM1);
```

```
01 //C0^C1 = k^M0^K^M1 = M0^M1 = 3345361A09520545440648071A10580B
02 //Resultado: M0^M1^M0 = M1 = a Cifra de fluxo
```

1.4.7. Maleabilidade indevida dos encriptadores de fluxo

Os criptogramas produzidos por sistemas criptográficos exclusivamente para sigilo, sem mecanismos de verificação de integridade e autenticidade, não possuem garantias de integridade e estão sujeitos a corrupções acidentais ou maliciosas. Em particular, os criptogramas gerados pelos encriptadores de fluxo são maleáveis e podem ser intencionalmente modificados de modo a produzir alterações controladas no texto claro.

Seja $C = M\hat{}K$ o criptograma de Ana obtido pelo XOR do fluxo de chaves K com o texto claro M. Aqui K foi bem construído e não é reutilizado. Ivo não precisa conhecer K. Porém, M possui estrutura e partes do conteúdo conhecidas por Ivo e, por isto, ele consegue realizar com precisão as modificações desejadas. Ivo deseja alterar M para o valor N a partir de mudanças controladas sobre C. Ivo calcula a mudança $X = M\hat{}N$. Ivo intercepta C em trânsito, calcula o criptograma modificado $Y = C\hat{}X$ e envia Y para Beto. Beto recebe Y e calcula $Y\hat{}K = (C\hat{}X)\hat{}K = (M\hat{}K)\hat{}(M\hat{}N)\hat{}K = (M\hat{}M)\hat{}(K\hat{}K)\hat{}N = 1\hat{}1\hat{}N = N$. Desta forma, Beto obtém apenas o texto claro falsificado N.

O trecho de código do Programa 12 mostra como criptogramas produzidos com o AES/CTR podem ser modificados, resultando em textos claros falsos, porém de conteúdo válido no contexto da aplicação. Ana e Beto usam o mesmo encriptador de fluxo da linha 02. A mensagem de Ana para Beto é composta de dois blocos de texto claro, na linha 05, em que Ana faz uma transferência de valor para Carlo. Nas linhas de 07 a 10 os blocos do criptograma são gerados conforme pretendido por Ana e Beto. Vale dizer que o IV do segundo bloco é incrementado de um, para evitar a repetição.

Nas linhas 14 a 17, Ivo realiza seu ataque. Primeiro, ele inclui seu nome na mensagem do seguinte modo. Nas linhas 14 e 15, Ivo calcula $X =$ *"Carlo"*^*" Ivo"* e $C_0 = C_0\hat{}X$. Em seguida, nas linhas 16 e 17, Ivo aumenta o valor da mensagem calculando $X =$ *"010.000,00"*^*" 100.998,54"* e, em seguida, $C_1 = C_1\hat{}X$. O resultado recebido por Beto são os blocos da mensagem falsa *"Ana para Ivo"* e *"Valor:100.998,54"*.

Programa 12: Maleabilidade dos encriptadores de fluxo com AES/CTR.

```
01 // Configurações comuns para Ana e Beto
02 Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");
03
04 // A mensagem é composta de dois blocos de texto claro
05 byte[][]M = {("Ana   para Carlo").getBytes(),
06              ("Valor:010.000,00").getBytes()};
07
08 for (int i= 0; i < M.length; i++){
09   c.init(Cipher.ENCRYPT_MODE,ks,new IvParameterSpec(iv[i]));
10   C[i] = c.doFinal(M[i]);
11 }
12
13 //Ivo realiza o ataque
14 X = U.xor("Ana   para Carlo".getBytes(),
15           "Ana   para   Ivo".getBytes()); C[0] = U.xor(C[0], X);
16 X = U.xor("Valor:010.000,00".getBytes(),
17           "Valor:100.998,54".getBytes()); C[1] = U.xor(C[1], X);
```

```
01 // Resultado: Ivo é o novo recebedor de um valor muito mais alto
02 // Resultado: N[0] =Ana   para   Ivo
03 // Resultado: N[1] =Valor:100.998,54
```

1.4.8. Combinação de integridade e encriptação

O mau uso descrito na seção anterior não pode ser evitado somente pela verificação de integridade da mensagem. De fato, *hashes* não são capazes de detectar a corrupção maliciosa de mensagens maleáveis. Esta ideia incorreta constitui outro mau uso da criptografia. O trecho de código do Programa 13 mostra como a combinação de encriptação com verificação de integridade por *hash* não é suficiente para proteger um criptograma maleável contra modificações intencionais e maliciosas. As linhas 01 a 04 contêm as configurações do sistema criptográfico para Ana e Beto. Neste exemplo vale ressaltar a função de resumo criptográfico SHA256 na linha 03. A encriptação é feita com AES/CTR.

Nas linhas 06 a 09, Ana inicializa as funções de *hash* e de encriptação, computa o criptograma do texto claro e calcula o *hash* do criptograma. Esta técnica é conhecida como *Encrypt-then-Hash* (EtH). Nas linhas 11 a 15, Ivo intercepta a mensagem de Ana para Beto, e se aproveita da maleabilidade do encriptador de fluxo para modificar o criptograma, de modo a incluir seu próprio nome no lugar do nome de Beto no texto claro. Além disso, Ivo recalcula o *hash* sobre o novo criptograma e o envia, junto com o novo criptograma, para Beto.

Nas linhas 17 a 20, ao receber a mensagem com o criptograma e o *hash*, Beto inicializa as funções de *hash* e de encriptação, calcula o *hash* sobre o criptograma e compara o *hash* calculado com o *hash* recebido. Uma vez que estes dois valores são iguais, Beto decripta o criptograma e obtém o texto claro. Neste exemplo, Beto foi incapaz de detectar a modificação maliciosa do texto claro feita por Ivo, uma vez que Ivo também substituiu o *hash* de verificação de integridade. Felizmente este mau uso é facilmente corrigido pela utilização de encriptação autenticada, em que a função de encriptação pode ser combinada com uma função de MAC, como por exemplo, o HMACSHA256, ou então substituída por uma única função de encriptação autenticada, como por exemplo, o AES/GCM.

Programa 13: Combinação de interidade e encriptação com SHA256 e AES/CTR.

```
01  // configurações do sistema criptográfico para Ana e Beto
02  Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");
03  MessageDigest md = MessageDigest.getInstance("SHA256", "BC");
04  byte[] textoclaroAna = "De Ana para Beto".getBytes();
05
06  //Ana calcula o hash do criptograma: Encrypt-then-Hash(EtH)
07  md.reset(); c.init(Cipher.ENCRYPT_MODE,sks,ivps);
08  byte[] criptograma = c.doFinal(textoclaroAna);
09  byte[] hash = md.digest(criptograma);
10
11  //Ataque: Ivo modifica o criptograma e recalcula o hash
12  X = U.xor("De Ana para Beto".getBytes(),
13            "De Ana para  Ivo".getBytes());
14  criptograma = U.xor(criptograma, X);
15  hash = md.digest(criptograma);
16
17  // decriptação pelo Beto com verificação da hash
18  md.reset(); c.init(Cipher.DECRYPT_MODE,sks,ivps);
19  if (MessageDigest.isEqual(md.digest(criptograma), hash))
20     { textoclaroBeto = c.doFinal(criptograma);}
```

1.4.9. Combinação manual de MAC e encriptação

A encriptação autenticada pode ser obtida manualmente pela combinação programática das funções de encriptação e autenticação. Em geral, existem três métodos para fazer a combinação das funções de MAC e encriptação, que se diferenciam em relação ao dado sobre o qual a *tag* de autenticação é calculada, conforme a seguir:

- Encrypt-then-MAC (EtM): encripta então autentica. Este método calcula a *tag* de autenticação sobre o criptograma e a *tag* não é encriptada;
- Encrypt-and-MAC (E&M): encripta e autentica. Este método calcula a *tag* de autenticação sobre o texto claro e a encriptação é feita somente sobre o texto claro e a *tag* não é encriptada;
- MAC-then-Encrypt (MtE): autentica então encripta. Este método calcula a *tag* de autenticação sobre o texto claro e depois encripta o texto claro e a *tag*.

O primeiro método, encripta então autentica, é considerado o mais seguro e foi exemplificado na seção anterior. Uma das vantagens do método EtM é que ele não revela informação sobre a integridade de formação do texto claro, uma vez que a *tag* foi calculada sobre o criptograma. Por isto, este método não é vulnerável ao ataque de *padding oracle*. A combinação manual pode ser substituída por uma única função de encriptação autenticada, como o AES/GCM. O trecho de código do Programa 14 mostra os outros dois métodos: encripta e autentica (E&M) e autentica então encripta (MtE). As configurações do sistema criptográfico estão nas linhas de 01 a 05. Duas chaves criptográficas são necessárias, uma para o AES/CTR e outra para o HMACSHA256.

O método encripta e autentica é mostrado nas linhas de 07 a 11. O criptograma é computado na linha 10 e a *tag* é calculada sobre o texto claro na linha 11. A verificação da *tag* requer a decriptação do criptograma. O método autentica então encripta é mostrado nas linhas de 13 a 18. A *tag* é calculada sobre o texto claro na linha 16, a *tag* é concatenada ao final do texto claro, linha 17, formando o pacote sobre o qual o criptograma é computado, na linha 18. A verificação da *tag* requer a decriptação do criptograma e a separação entre texto claro e *tag*. Nestes dois métodos, a verificação de integridade do texto claro é determinística e revela a repetição de um texto claro.

Programa 14: Encriptação autenticada manual com Encrypt-and-MAC e MAC-then-Encrypt.

```
01  //configurações do sistema criptográfico
02  SecretKeySpec sks1 = new SecretKeySpec(k, "AES");
03  SecretKeySpec sks2 = new SecretKeySpec(k, "HMACSHA256");
04  Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");
05  Mac m = Mac.getInstance("HMACSHA256", "BC");
06
07  // encripta e autentica: Encrypt-and-MAC (E&M)
08  m.init(sks2);
09  c.init(Cipher.ENCRYPT_MODE, sks1, new IvParameterSpec(iv));
10  criptograma = c.doFinal(textoclaroAna);
11  tag = m.doFinal(textoclaroAna);
12
13  // autentica entao encripta: MAC-then-Encrypt (MtE)
14  m.init(sks2);
15  c.init(Cipher.ENCRYPT_MODE, sks1, new IvParameterSpec(iv));
16  tag = m.doFinal(textoclaroAna);
17  byte[] pacote = Arrays.concatenate(textoclaroAna,tag);
18  criptograma = c.doFinal(pacote);
```

1.4.10. Tempo variável na verificação de *hashes* e MACs

Os maus usos criptográficos podem levar a vazamentos de informação por canais laterais (*side-channels* em inglês). Um canal lateral (de tempo) é alimentado pelas variações de tempo das computações sobre valores criptográficos. Por exemplo, quando a comparação de *hashes* ocorre em tempo variável, as pequenas variações de tempo na comparação revelam onde está a diferença entre valores comparados e podem ser utilizadas por Ivo para deduzir o valor de *hash* original. Para evitar este vazamento de informação, a comparação de *hashes* e MACs deve ser realizada em tempo constante e independente de onde ocorre a primeira diferença entre os valores comparados.

O trecho de código do Programa 15 mostra como é possível medir a diferença de tempo de comparação entre dois *hashes* de 64 bytes do SHA512. O laço da linha 05 percorre os 64 bytes do *hash* calculado, que é modificado no índice i (na linha 08). A comparação é feita pelo método Arrays.areEqual() que tem tempo variável. O resultado da tomada de tempo é mostrado na Figura 17, onde se percebe que o tempo de comparação aumenta com o índice i da primeira diferença entre valores comparados. Já os métodos MessageDigest.isEqual() e Arrays.ConstTimeAreEqual() oferecem tempos constantes, com vantagem de desempenho para o primeiro.

Programa 15: Tempo variável na verificação de hashes com SHA512.

```
01  MessageDigest md = MessageDigest.getInstance("SHA512", "BC");
02  boolean ok; long t1, t2; long t[] = new long[64];
03  byte[] hash1 = md.digest(textoClaro.getBytes());
04
05  for (int i = 0; i < t.length; i++) { // 64 bytes
06    md.reset();
07    byte[] hash2 = md.digest(textoClaro.getBytes());
08    hash2[i] = (byte) (hash2[i] ^ 0x01);
09    t1 = System.nanoTime();
10    ok = Arrays.areEqual(hash2, hash1);
11    t2 = System.nanoTime();
12    t[i] = t2 - t1;
13  }
```

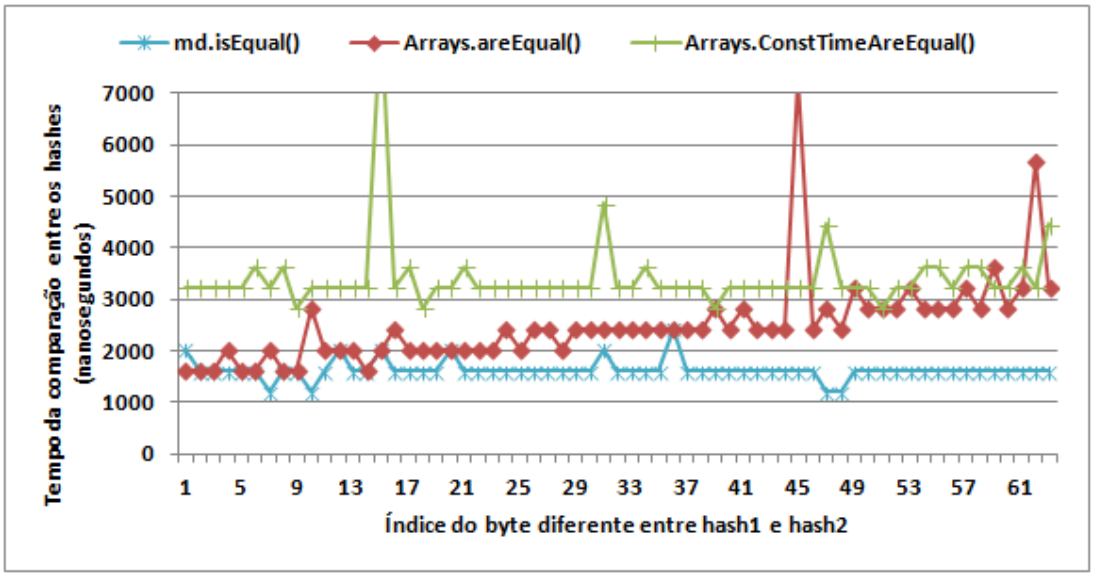


Figura 17: Comparação de *hashes* com tempo variável e com tempo constante.

1.4.11. *Padding oracles* do modo CBC

A vulnerabilidade de *padding oracle* (oráculo de preenchimento, em uma tradução livre) do modo CBC é um tipo de canal lateral conhecido pelos criptólogos desde 2002 [35], mas que só recentemente foi explorado em ataques reais a sistemas web [33][36]. Grosso modo, o *padding oracle* é um algoritmo que, ao receber um criptograma qualquer como entrada, retorna um único bit de informação sobre a validade do *padding* que completa o texto claro correspondente. Isto é, se o *padding* é válido, o oráculo retorna 1 ou *true* (verdadeiro); se ele não é válido, o oráculo retorna 0 ou *false* (falso). Ao usar estas respostas sobre a validade do texto claro, um adversário pode deduzir valores retornados pela função de decriptação.

Chama-se de vulnerabilidade do *padding oracle* a este vazamento de um bit da função de decriptação que, ao tentar decriptar qualquer criptograma, emitirá uma mensagem de erro informando quando o *padding* é inválido. Este vazamento de um bit pode ser utilizado iterativamente para descobrir todos os bytes de um texto claro, geralmente ao custo de apenas alguns milhares de iterações. Geralmente, a função de decriptação é executada mesmo que não retorne o texto claro para o adversário. Entretanto, ela deve informar se o texto claro tem um *padding* válido. Em situações extremas, o *padding oracle* serve não apenas como meio para ataques de decriptação de mensagens, mas também como bloco de construção para oráculos de encriptação.

O trecho de código do Programa 16 exemplifica um *padding oracle* simples. O oráculo recebe como entrada dois blocos de dados, o *iv* e o criptograma *c* (linha 01) e retorna *true* se a decriptação for bem sucedida. A decriptação de *c* é realizada (linha 06), mas não é retornada. Na linha 07, o oráculo trata a exceção de preenchimento ruim (`BadPaddingException`) do texto claro e retorna o valor *false* se ela ocorrer. Para este exemplo, todas as outras exceções são ignoradas. O valor lógico retornado na linha 12 é suficiente para a realização do ataque.

O diagrama de sequência da Figura 18 ilustra a implementação do ataque de *padding oracle* do modo CBC. Primeiro, o atacante Ivo obtém um criptograma e um IV conhecido e, iterativamente, modifica o byte menos significativo do IV e submete IV e *c* ao oráculo para descobrir o valor do *padding*. Em seguida, Ivo, mais uma vez pela manipulação do IV, modifica iterativamente o *padding* descoberto para decriptar, byte a byte, o criptograma, fazendo questionamentos repetidos ao oráculo. *Padding oracles* como este já foram encontrados em diversos softwares criptográficos [33][36].

Programa 16: *Padding oracle* do modo CBC.

```
01  public static boolean oracle(byte[] iv, byte[] c) {
02    boolean ok = true;
03    try {
04      Cipher c = Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
05      c.init(Cipher.DECRYPT_MODE, ks, new IvParameterSpec(iv));
06      c.doFinal(c); // ignora a saída do doFinal()!!!!
07    } catch (BadPaddingException e) { ok = false;
08    } catch (NoSuchAlgorithmException | NoSuchProviderException |
09             NoSuchPaddingException | InvalidKeyException |
10             InvalidAlgorithmParameterException |
11             IllegalBlockSizeException ex) { /* ignora!! */}
12  return ok;}
```

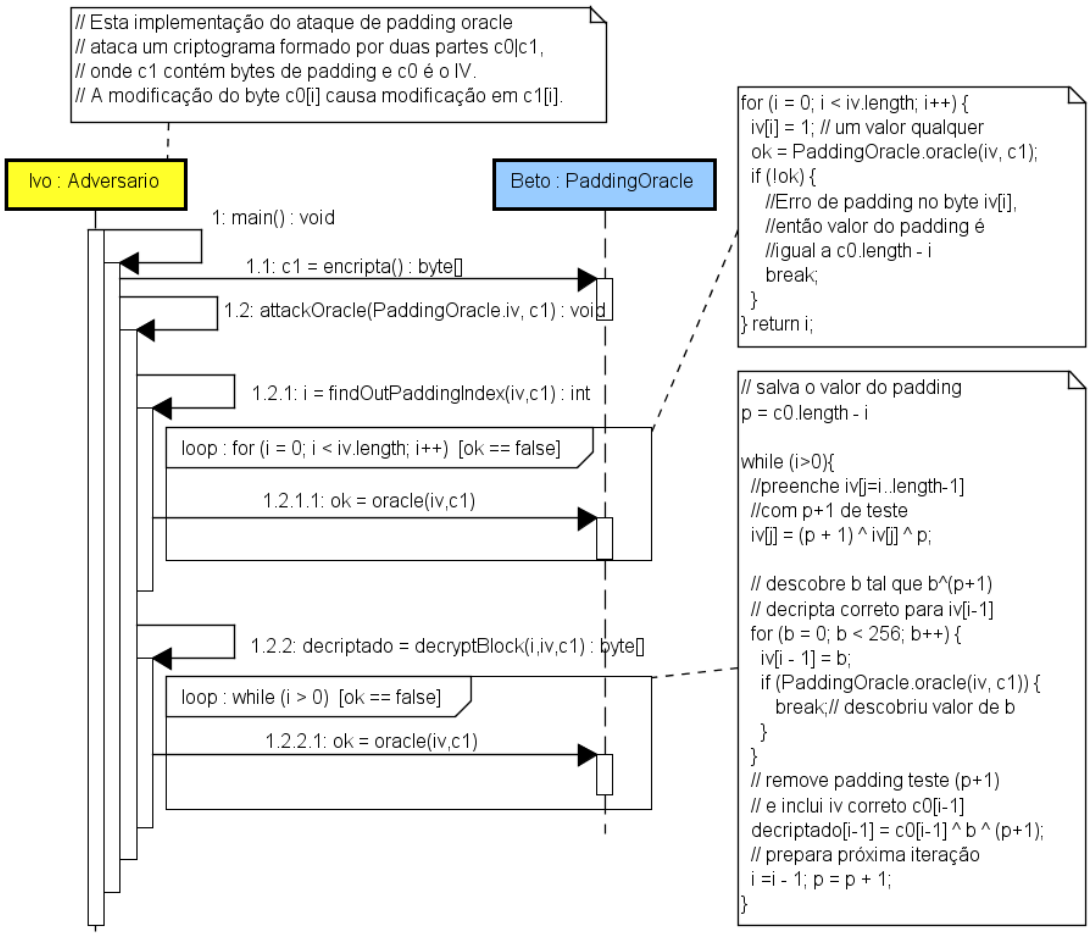


Figura 18: Diagrama de sequência (UML) do ataque de *padding oracle* do modo CBC.

Existem três maus usos criptográficos que podem ser diretamente associados às causas da vulnerabilidade de *padding oracle*:

- O uso de encriptação não autenticada em canais de comunicação, de modo que criptogramas injetados pelo adversário no canal sejam considerados válidos;
- Composição de encriptação e autenticação com os métodos MAC-then-Encrypt ou MAC-then-(pad-then)-Encrypt, de modo que a proteção de integridade só é aplicada sobre o texto claro (e o *padding*) e não sobre o criptograma, permitindo ao adversário a manipulação do criptograma para inferir a validade do *padding*;
- O uso de IVs fixos ou previsíveis com o modo CBC, de modo que ataques de decriptação possam ser montados a partir da previsão dos IVs.

Estes maus usos podem ser resolvidos pela adoção compulsória da autenticação em conjunto com a encriptação, que também protege a integridade. Ainda, ela precisa ser aplicada de maneira correta, sobre o criptograma completo, em vez de sobre o texto claro. Além disso, a função atômica de encriptação autenticada elimina a necessidade de combinação programática explícita de encriptação e autenticação, reduzindo a chance de erros. Finalmente, a última contramedida para eliminar este canal lateral é a inibição de mensagens de erro relacionadas à validação de integridade sobre o texto claro nas computações que fazem decriptação em protocolos de comunicação.

1.4.12. Validação incompleta de certificados digitais

A validação de certificados digitais em softwares criptográficos é relativamente bem estruturada e confiável, uma vez que as falhas de programação mais comuns no uso de certificados digitais já foram identificadas e solucionadas. Conforme discutido anteriormente, os quatro pontos mais importantes na validação do certificado digital são a confirmação do nome do sujeito ou usuário, a verificação da assinatura digital presente no certificado (e a consequente verificação da cadeia de certificação), a verificação da data de expiração e a presença do número do certificado em uma lista de revogação.

Em softwares criptográficos em geral, e nos aplicativos móveis em especial, existe uma variedade grande de bibliotecas para estabelecimento de conexões SSL/TLS. De acordo com estudos recentes [16][19], todas elas permitem que o programador desabilite um ou outro item na verificação do certificado, para fins de desenvolvimento. Porém, há risco da configuração desabilitada ser implantada acidentalmente em ambiente de produção. Em particular, a não verificação da assinatura ou da cadeia de certificação facilita o ataque do intermediário malicioso (*Man-in-The-Middle* – MiTM).

A vulnerabilidade Heartbleed [31] descoberta no código da extensão do TLS em abril de 2014 e presente no software criptográfico OpenSSL [40] é um exemplo de como defeitos comuns podem resultar em vulnerabilidades graves nos softwares criptográficos. A vulnerabilidade foi causada pela falta de verificação dos limites em um buffer em que o *nonce* é armazenado pelo protocolo Heartbeat. Na linguagem C, o tamanho do buffer é desvinculado da variável que registra este seu tamanho. A falta de consistência entre tamanho do buffer e a variável que registra o tamanho pode ser explorada por meio de leitura estendida de posições de memória além do tamanho real do buffer. Este defeito de programação insegura simples, quando presente em um protocolo de comunicação, como foi o caso do protocolo Heartbeat, possibilitou a leitura de regiões de memória do processo do OpenSSL, no servidor, em que eram mantidas diversas informações sensíveis tais como chaves de sessão, senhas de usuário e até chaves privadas de servidores, facilitando os ataques MiTM.

Outra vulnerabilidade recente [32] é mais um defeito do OpenSSL descoberto no sistema operacional iOS da Apple em fevereiro de 2014. O defeito consistiu de duas sentenças “goto fail” consecutivas e sem chaves delimitadoras colocadas depois de uma cláusula condicional (“if”). Este código resultou em um salto incondicional para a sentença rotulada de “fail”. Este defeito ocorreu em um ponto crítico do estabelecimento de sessão do SSL, na verificação de uma assinatura digital; por causa do defeito, passou a ser considerada sempre bem sucedida, mesmo na ocorrência de certificados inválidos ou assinatura inexistente, mais uma vez favorecendo o ataque MiTM.

Estas vulnerabilidades poderiam ter sido facilmente identificadas pela aplicação de técnicas comuns de auditoria de código fonte, manuais ou até automáticas. Por outro lado, as vulnerabilidades poderiam nunca ter ocorrido se linguagens de programação com tipos fortes tivessem sido utilizadas. Para ilustração, o software de infraestrutura de chaves públicas EJBCA [37], escrito em Java e que utiliza à biblioteca criptográfica BouncyCastle, também em Java, são imunes às vulnerabilidades de leitura de buffer além dos limites, que causaram a vulnerabilidade Heartbleed.

1.5. Por dentro de um encriptador de blocos

Este seção ilustra a construção de uma implementação do algoritmo AES conforme a especificação [23], em que são destacados os aspectos necessários para utilizá-la como um encriptador de blocos [26]. Além disso, são tratadas otimizações algorítmicas e de código, automação de testes funcionais [24] e análise de segurança da implementação.

1.5.1. O algoritmo AES

A estrutura de dados principal do AES é o *square*, uma matriz 4x4 de células de 1 byte cada, totalizando 128 bits. Todas as operações do AES são realizadas sobre esta matriz. Computações também são feitas sobre palavras de quatro bytes. Uma palavra é uma linha ou uma coluna da matriz. O conjunto de valores correntes da matriz é chamado de estado. Cada rodada do AES usa uma subchave de 128 bits derivada da chave principal por meio de um mecanismo de expansão de chaves. O AES trabalha com três tamanhos de chaves: 128, 192 e 256 bits. O laço principal do AES possui 10,12 ou 14 rodadas, conforme o tamanho da chave. O Programa 17 mostra a função de encriptação do AES, com as operações sobre a estrutura de estado: adição da chave da rodada, a substituição de bytes, a rotação de linhas, e a mistura de colunas. A decriptação é análoga, com as operações inversas.

1.5.2. Modos de operação e *padding*

A implementação crua do AES é de pouca utilidade prática. A fim de torná-la útil é necessário acrescentar as construções necessárias para torná-la um encriptador de bloco: modos de operação e esquema de *padding* do texto claro, para que este tenha o tamanho de um múltiplo do tamanho do bloco do algoritmo de encriptação. Em um encriptador de bloco, a operação de encriptação insere o *padding* e em seguida encripta com o modo de operação determinado. A operação de decriptação decripta com o modo de operação correspondente e em seguida remove o *padding*, conforme visto no Programa 18 (linhas 01 a 15). As linhas 17 a 55 mostram os modos de operação ECB (17 a 25), CBC (27 a 41) e CTR (43 a 54) [26]. O *padding* PKCS#5/PKCS#7 [21] está a partir da linha 58.

Programa 17: Função de encriptação do AES.

```
01  public void encrypt(byte[] in, byte[] out) {
02    wCount = 0;
03    byte[][] state = new byte[4][Nb];
04    Util.toState(state, in);
05    addRoundKey(state);
06    for (int round = 1; round < Nr; round++) {
07      subBytes(state);
08      shiftRows(state);
09      mixColumns(state);
10      addRoundKey(state);
11    }
12    subBytes(state);
13    shiftRows(state);
14    addRoundKey(state);
15    Util.fromState(out, state);
16  }
```

Programa 18: Modos de operação e *padding* de um encriptador de bloco.

```
01  public byte[] doCipher(byte[] in){
02    byte[] inPadded = null, outPadded = null, out = null;
03    if (padding){
04      if(enc==ENCRYPT) {inPadded = doPadding(in);}
05      if(enc==DECRYPT) {inPadded = in;}
06    } else { inPadded = in;}
07    outPadded = new byte[inPadded.length];
08         if (this.opMode == ECB) {doECB(inPadded,outPadded);}
09    else if (this.opMode == CBC) {doCBC(inPadded,outPadded);}
10    else if (this.opMode == CTR) {doCTR(inPadded,outPadded);}
11    if (padding){
12      if (enc == ENCRYPT) { out = outPadded; }
13      if (enc == DECRYPT) { out = undoPadding(outPadded);}
14    } else { out = outPadded;} return out;
15  }
16
17  private void doECB(byte[] in, byte[] out) {
18    int numBlocks = (int)(in.length/blockSize);
19    for (int i = 0; i < numBlocks; i++) {
20      int offset = i * blockSize;
21      for (int j=0; j<blockSize; j++) { inBlock[j] = in[offset+j];}
22      if      (enc == ENCRYPT) { aes.encrypt(inBlock, outBlock);}
23      else if (enc == DECRYPT) { aes.decrypt(inBlock, outBlock);}
24      for (int j=0; j<blockSize; j++) {out[offset+j] = outBlock[j];}
25  }}
26
27  private void doCBC(byte[] in, byte[] out) {
28    int numBlocks = (int)(in.length/blockSize);
29    byte[] inX = new byte[blockSize], outX = new byte[blockSize];
30    for (int i=0; i<numBlocks; i++) {
31      int offset = i * blockSize;
32      for (int j=0; j<blockSize; j++) {inBlock[j] = in[offset+j];}
33      if (i == 0) {copy(iv,outX);}
34      if (enc == ENCRYPT) {
35        xor(inBlock,outX,inX);
36        aes.encrypt(inX,outBlock);
37        copy(outBlock,outX);}
38      if (enc == DECRYPT) {
39        aes.decrypt(inBlock, inX);
40        xor(inX,outX,outBlock);
41        copy(inBlock,outX)   ;}
42      for (int j=0; j<blockSize; j++) {out[offset+j] = outBlock[j];}
43  }}
44
45  private void doCTR(byte[] in, byte[] out) {
46    int numBlocks = (int)(in.length/blockSize), sizeCTR = blockSize/2;
47    byte[] inCTR = new byte[blockSize], outCTR = new byte[blockSize];
48    for (int i = 0; i < numBlocks; i++) {
49      int offset = i * blockSize;
50      for (int j=0; j<blockSize; j++) {inBlock[j] = in[offset+j];}
51      if (i == 0) {copy(iv,inCTR);}
52      else {standardIncrement(inCTR,sizeCTR);}
53      if (enc == ENCRYPT || enc == DECRYPT)
54      { aes.encrypt(inCTR, outCTR); xor(outCTR,inBlock,outBlock);}
55      for(int j=0; j<blockSize; j++) {out[offset+j] = outBlock[j];}
56  }}
57
58  private byte[] doPadding(byte[] in){
59    byte pad[] = {16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1};
60    byte val   = (byte) (in.length%blockSize);
61    int numBlocks = (int)(in.length/blockSize);
62    byte[] padded = new byte[(numBlocks+1)*blockSize];
63    for(int i=0; i<in.length; i++){padded[i] = in[i];}
64    for(int i=in.length; i<padded.length; i++){padded[i]=(byte)pad[val];}
65    return padded;
66  }
67  private byte[] undoPadding(byte[] padded){
68    byte[] out = new byte[padded.length - padded[padded.length-1]];
69    for(int i=0; i<out.length; i++){ out[i]=padded[i];} return out;}
```

Programa 19: Dois métodos de multiplicação.

```
01   public byte FFMul(byte a, byte b) {
02     byte aa = a, bb = b, r = 0, t;
03     while (aa!=0){
04       if ((aa&1) != 0) {r = (byte)(r^bb);}
05       t = (byte) (bb&0x80); bb =(byte)(bb << 1);
06       if (t != 0) {bb =(byte) (bb^0x1b);}
07       aa =(byte)((aa & 0xff) >> 1);
08     }
09     return r;
10   }
11   public byte FFMulFast(byte a, byte b) {
12    int t = 0;
13    if (a == 0 || b == 0) { return 0;}
14    t = (L[(a & 0xff)] & 0xff) + (L[(b & 0xff)] & 0xff);
15    if (t > 255) { t = t - 255;}
16    return E[(t & 0xff)];
17   }
```

1.5.3. Otimização da implementação do algoritmo AES

A implementação do AES sofreu otimizações algorítmicas e de código. A substituição do método de multiplicação utilizado pelas operações de encriptação e decriptação foi a que resultou no maior ganho de desempenho, pois baixou a complexidade do algoritmo. O método convencional, com deslocamentos de bits, foi substituído pelo método mais eficiente baseado em *lookup* de tabelas. O Programa 19 mostra os dois métodos, o mais lento (linhas 1 a 10) e o mais rápido (linhas 11 a 17).

Outras otimizações realizadas foram a desenrolagem de laços dentro do fluxo de controle do algoritmo e a pré-computação das tabelas. Nestas otimizações de código os ganhos foram mais modestos. A Figura 19 mostra o ganho de desempenho acumulado com as otimizações combinadas: sem otimização, otimização algorítmica (multiplicação rápida) e otimizações de código. A figura mostra os tempos, em milissegundos, de encriptação e decriptação para três implementações, nos três tamanhos de chave. A otimização algorítmica ofereceu a maior contribuição no desempenho final. A decriptação é mais lenta por causa da inversão modular na decriptação.

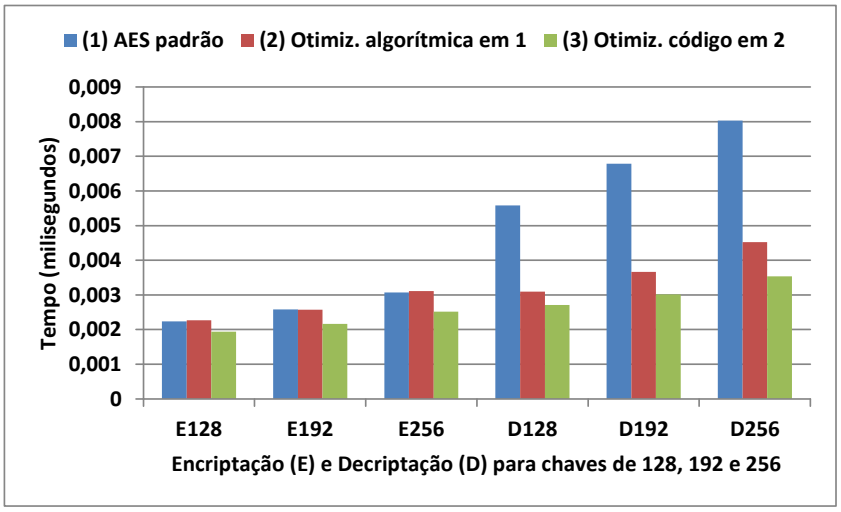


Figura 19: Tempos de encriptação e decriptação para três implementações do AES.

1.5.4. Validação funcional com vetores de teste

A validação funcional de implementações criptográficas pode ser realizada com o auxílio de vetores de teste criptográficos, que são conjuntos de dados construídos com o objetivo de avaliar a correção de implementações criptográficas em relação às normas e especificações. A correção funcional de uma implementação criptográfica é premissa para a segurança desta implementação, uma vez que uma implementação incorreta não oferece segurança. Porém, a validação não oferece uma medida da segurança da implementação. Além de ajudar a determinar a conformidade com as especificações, a validação pode detectar falhas de implementação, incluindo problemas com ponteiros, a alocação insuficiente de memória, tratamento incorreto de erros e comportamentos anômalos diversos na implementação avaliada.

Devido ao grande volume de dados envolvidos, os vetores de teste são utilizados em testes automáticos das implementações criptográficas. Para que a validação seja viável, o software criptográfico deve permitir ao software de validação ter o controle sobre os parâmetros de entrada necessários aos testes, por exemplo, por meio de uma interface de programação (API). Se um software criptográfico não permite o controle dos parâmetros de entrada, os testes não podem ser realizados satisfatoriamente.

Existem diversas bases ou vetores de teste para validação de implementações de algoritmos criptográficos. Um conjunto de vetores de boa reputação é o do NIST CAVP [24]. Os vetores de testes devem ter uma amostragem estatística, senão exaustiva, que permita pelo menos a cobertura completa dos fluxos de controle do algoritmo. Quanto melhor a cobertura da validação, maior será a confiança na correção da implementação. Além disto, a validação de implementações contra amostras diversas e atualizadas é uma boa prática de teste de software. A Figura 20 conceitua a validação de algoritmos com vetores de teste criptográficos. Vale observar que em se tratando de algoritmos não padronizados, geralmente o próprio criptólogo disponibiliza os vetores de teste.

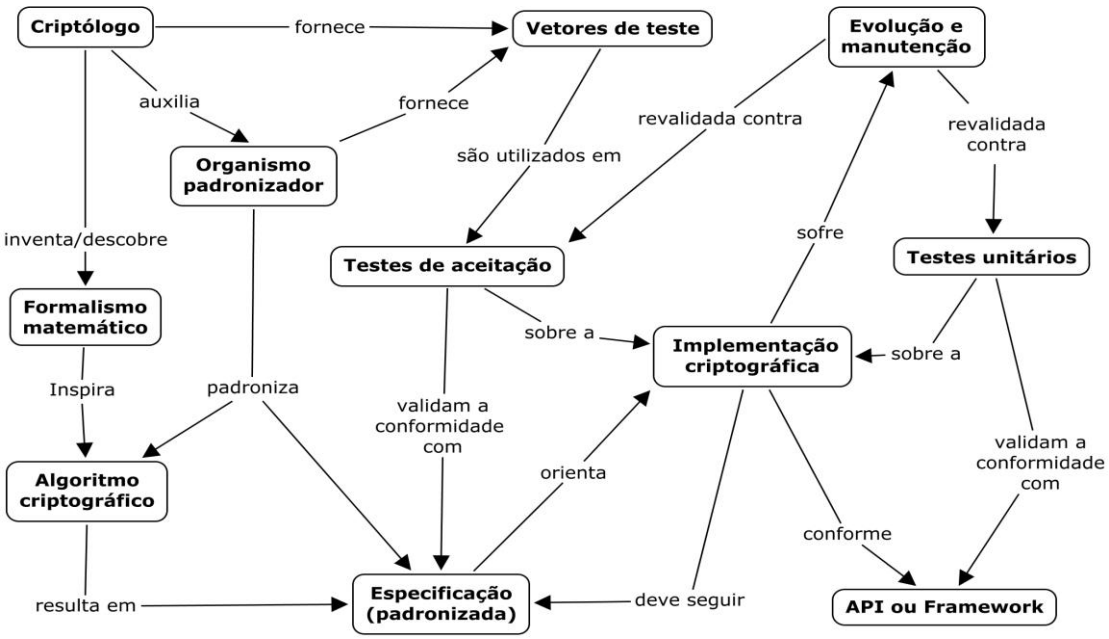


Figura 20: O conceito de validação de algoritmos criptográficos com vetores de teste.

1.5.5. Análise de segurança da implementação

Neste ponto, supõe-se que a implementação tenha passado pelos testes de validação e possa ser considerada, em certa medida, aderente à sua especificação com uma confiança justificada pela cobertura ou abrangência da validação. Isto, porém, não significa que a implementação seja segura. Em particular, a implementação está sujeita às vulnerabilidades dos sistemas criptográficos e de computação em que será utilizada.

Conforme visto anteriormente, existem diversas maneiras de usar mal um encriptador de blocos. Por exemplo, o modo ECB é determinístico e está sujeito à repetição de padrões, enquanto o modo CBC está sujeito ao mau uso de IVs e, combinado ao *padding* PKCS#7, é vulnerável ao ataque por canal lateral conhecido como *padding oracle*. Já o modo CTR é seriamente afetado pelo reuso de IVs.

Além de variações do tempo de execução, ataques por canais laterais também usam monitoramento do consumo de energia, uso da memória, emanação eletromagnética, tratamento de erros, entre outras [5]. Como já vimos, as diferenças de tempo durante as computações com chaves criptográficas podem levar a ataques de canal lateral, assim como as diferenças de tempo entre o processamento bem sucedido e o tratamento de erros e exceções. Uma implementação criptográfica deve evitar os vazamentos por canais laterais. Entretanto, nem sempre é possível obter uma solução efetiva somente em software, sem o apoio do hardware subjacente ou das outras camadas de software intermediárias.

Em Java, as modificações no código fonte feitas para eliminar canais laterais podem ser desfeitas pela JVM durante o processo de compilação em tempo real (*Just-in-Time Compilation* - JiTC). Este processo pode não apenas desfazer proteções, como também acrescentar vulnerabilidades relacionadas aos canais laterais. Este é o caso das construções em código fonte para computações em tempo constante.

Finalmente, implementações do AES são particularmente vulneráveis aos ataques por canais laterais sobre os acessos à memória *cache* dos processadores [5]. Duas variações deste tipo de ataque são: a que correlaciona o tempo de execução aos acertos e faltas na *cache*; e a que monitora a sequência de acertos e faltas pelo consumo de energia da CPU [6]. A viabilidade destes ataques muitas vezes requer a execução concorrente de código malicioso para o monitoramento do sistema alvo. Na implementação em Java do AES, a pré-computação e carga antecipada das tabelas seriam uma medida de proteção contra os ataques de *cache*, se não fossem potencialmente desfeitas pelas diversas transformações de código até a sua execução em um hardware específico.

1.6. Considerações finais

Este texto é um primeiro passo na direção da construção de um arcabouço que dê ao programador não especialista a confiança necessária para a codificação segura de métodos criptográficos. Como todos sabemos, o provimento de segurança é uma tarefa sem fim, sob constante ameaça de novas tecnologias e do talento de adversários mais ou menos bem intencionados. Por isso mesmo este texto não pode pretender ser completo e estático. Mais do que os elementos específicos (contramedidas, recomendações, etc), nosso objetivo foi despertar o leitor para a necessidade de implementações cuidadosas de métodos criptográficos, ilustrando com exemplos a diferença entre um código real e aqueles encontrados em livros-textos da área. Esperamos que o leitor programador use este texto como um primeiro recurso para a sua atividade, e que desfrute da sua leitura tanto quanto desfrutamos da sua escrita.

A elaboração deste texto foi desafiadora porque exigiu não apenas os conhecimentos técnicos em criptografia e programação, mas também a habilidade para apresentar os conceitos do modo mais adequado ao público alvo. Um senso de utilidade prática esteve presente durante toda a elaboração do texto. Por isto, muitas vezes, quando foram tomadas decisões de compromisso entre complexidade do conteúdo e facilidade de apresentação, a última sempre foi preferida em detrimento da primeira. A elaboração de exemplos práticos dos maus usos mais comuns em criptografia pode auxiliar programadores em geral a melhorar a segurança de seus softwares. Em um desdobramento interessante, a coleção de maus usos contida neste texto pode dar origem a uma lista de verificação, de apoio à inspeção de código fonte, com grande utilidade em verificações de segurança realizadas sobre softwares criptográficos.

Como era de se esperar em um texto curto (cerca de 50 páginas apenas) para um assunto abrangente e complexo, houve grandes omissões propositais. As ausências mais evidentes, motivadas pelas faltas tanto de espaço no texto quanto de tempo para elaborar uma apresentação adequada, são os exemplos programáticos de acordo de chaves e de validação de certificados digitais. Houve casos em que não foram encontradas, na infraestrutura de software escolhida, implementações fáceis de usar em demonstrações, como foi o caso do acordo de chaves com Diffie-Hellman autenticado.

Além disso, há ainda diversas práticas (boas e ruins) de programação criptográfica que não foram incluídas e poderão, em conjunto às omissões supracitadas, dar origem a um segundo volume deste texto. Citam-se os seguintes assuntos de interesse: a escolha de parâmetros para geração de chaves para algoritmos assimétricos, a utilização programática de curvas elípticas, os bons usos de encriptação determinística em buscas sobre dados cifrados e práticas boas e ruins em certificação digital. Naturalmente, a listagem não pode ser exaustiva.

Finalmente, um último comentário sobre a bibliografia utilizada na elaboração deste texto. Além de respaldar os conceitos apresentados no texto, a bibliografia foi escolhida e organizada para servir como material de leitura para o leitor interessado em aprofundar os estudos. Assim, a lista bibliográfica está dividida nos seguintes tópicos: *(i)* livros texto sobre criptografia moderna, *(ii)* aspectos de implementação criptográfica, *(iii)* programação criptográfica, *(iv)* maus usos da criptografia, *(v)* normas e padrões, *(vi)* vulnerabilidades e ataques e *(vii)* softwares criptográficos. Espera-se, com esta organização, facilitar a navegação do leitor pelo tema e também os estudos futuros.

1.7. Agradecimentos

Alexandre Braga agradece à UNICAMP pelo apoio institucional e financeiro e ao CPqD pelo apoio institucional às atividades acadêmicas de seus funcionários. Ricardo Dahab agradece a UNICAMP e a Universidade de Waterloo pelo apoio institucional, e à Fapesp, Capes e CNPq por auxílios à pesquisa.

1.8. Referências

1.8.1. Criptografia moderna

[1] A. Menezes, P. Van Oorschot, and S. Vanstone, Handbook of applied cryptography. CRC press, 1996.

[2] J. Katz and Y. Lindell, "Introduction to Modern Cryptography," 2006.

[3] W. Mao, *Modern cryptography: theory and practice*. 2003.

[4] W. Stallings, Cryptography and network security, principles and practices. 2003.

1.8.2. Aspectos de implementação criptográfica

[5] C. Koç, About Cryptographic Engineering. 2009.

[6] N. Ferguson, B. Schneier, and T. Kohno, Cryptography Engineering: Design Principles and Practical Applications. Wiley, 2011.

[7] P. Gutmann, "Everything you Never Wanted to Know about PKI but were Forced to Find Out." [Online] https://www.cs.auckland.ac.nz/~pgut001/pubs/pkituto rial.pdf.

[8] P. Gutmann, "Godzilla crypto tutorial - Part 2, Key Management and Certificates." [Online]. Available: https://www.cs.auckland.ac.nz/~pgut001/tutorial/index.html.

1.8.3. Programação criptográfica

[9] A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software, Chapter 16 in Pattern Languages of Program Design 4 (N. Harrison, B. Foote, and," in Also in Procs. of PLoP, 1999.

[10] CryptoWorkshop and BouncyCastle, The Cryptoworkshop Guide to Java Cryptography and the Bouncy Castle APIs. 2013.

[11] D. Hook, Beginning cryptography with Java. John Wiley & Sons, 2005.

[12] T. S. Denis, Cryptography for Developers. Syngress Publishing, 2006.

1.8.4. Maus usos da criptografia

[13] B. Schneier, "Cryptographic design vulnerabilities," Comp., Sept., pp.29–33, 1998.

[14] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, "Why Does Cryptographic Software Fail?: A Case Study and Open Problems," in 5th Asia-Pacific Workshop on Systems, 2014, pp. 7:1–7:7.

[15] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, “An empirical study of cryptographic misuse in android applications,” ACM SIGSAC conference on Computer & communications security - CCS ’13, pp. 73–84, 2013.

[16] M. Georgiev, S. Iyengar, and S. Jana, “The most dangerous code in the world: validating SSL certificates in non-browser software,” in Proceedings of the 2012 ACM conf. on Computer and comm.. security - CCS ’12 (2012), 2012, pp. 38–49.

[17] P. Gutmann, “Lessons Learned in Implementing and Deploying Crypto Software,” Usenix Security Symposium, 2002.

[18] R. Anderson, “Why cryptosystems fail,” in 1st ACM conference on Computer and communications security, 1993, pp. 215–227.

[19] S. Fahl, M. Harbach, and T. Muders, “Why Eve and Mallory love Android: An analysis of Android SSL (in) security,” in ACM conference on Computer and communications security, 2012, pp. 50–61.

[20] S. Shuai, D. Guowei, G. Tao, Y. Tianchang, and S. Chenjie, “Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications,” in IEEE 12th Intl. Conf. Dependable, Autonomic and Secure Computing, 2014, pp. 75–80.

1.8.5. Normas e padrões

[21] B. Kaliski, “PKCS #5: Password-Based Cryptography Specification Version 2.0 (RFC 2898).” [Online]. Available: http://tools.ietf.org/html/rfc2898.

[22] J. Jonsson and Burt Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” RSA Laboratories, 2003. [Online]. Available: https://tools.ietf.org/html/rfc3447.

[23] NIST, “Advanced Encryption Standard (AES),” NIST FIPS PUB 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[24] NIST, “Cryptographic Algorithm Validation Program (CAVP).” [Online]. Available: csrc.nist.gov/groups/STM/cavp/index.html.

[25] NIST, “Digital Signature Standard (DSS),” NIST FIPS PUB 186-4, 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[26] NIST, “Recommendation for Block Cipher Modes of Operation,” NIST SP 800-38A,2001.[Online]. At: csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[27] NIST, “Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC,” NIST SP 800-38D, 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[28] NIST, “Recommendation for Key Management – Part 1: General (Revision 3),” NIST SP 800-57, 2012. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

[29] NIST, “Secure Hash Standard (SHS),” NIST FIPS PUB 180-4, 2015. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[30] NIST, “The Keyed-Hash Message Authentication Code (HMAC),” NIST FIPS PUB 198-1, 2008. [Online]. Available: http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf.

1.8.6. Vulnerabilidades e ataques

[31] “The Heartbleed Bug.” [Online]. Available: http://heartbleed.com/.

[32] Apple’s SSL/TLS ‘Goto fail’ bug.” [Online]. Available: www.imperialviolet.org/2014/02/22/applebug.html.

[33] J. Rizzo and T. Duong, “Practical padding oracle attacks,” Proceedings of the 4th USENIX conference on Offensive technologies (2010), pp. 1–9, 2010.

[34] Piercing Through WhatsApp’s Encryption.” [Online]. Available: https://blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption/.

[35] S. Vaudenay, “Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS...,” Advances in Cryptology—EUROCRYPT 2002, no. 1, 2002.

[36] T. Duong and J. Rizzo, “Cryptography in the Web: The Case of Cryptographic Design Flaws in ASP.NET,” IEEE Security and Privacy, pp. 481–489, 2011.

1.8.7. Softwares criptográficos

[37] “EJBCA - Open source PKI and CA.” [Online]. Available: www.ejbca.org.

[38] “Java Cryptography Architecture (JCA) Reference Guide.” [Online]. Available: docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html.

[39] “The Legion of the Bouncy Castle.” [Online]. Avail.: http://www.bouncycastle.org.

[40] “OpenSSL Cryptography and SSL/TLS toolkit.” [Online]. Avail.: OpenSSL.org.

Capítulo 2

Segurança em Mobile Crowd Sensing

Joélisson Joaquim de V. Laurido e Eduardo Luzeiro Feitosa

Abstract

Mobile Crowd Sensing (MCS) applications allow that participants or users to collect (using different sensors on your mobile devices - microphone, camera, GPS, etc.) and share information in order to assist other users in decision-making (based on third-party opinion) or inform about different events. Examples of MCS applications include coverage of instantly news, traffic congestion, weather disasters, air pollution, parking spaces, among others. Once the operation of MCS applications involves human or mechanical participation (cars, drones, sensors, etc.) to collect data that will be used by end users, the privacy and safety of the participants, not to mention the reliability of generated and received information are important issues that need investigation. In this context, this chapter aims to provide understanding of the environment of MCS applications, focusing mainly on issues related to privacy, security and reliability of the information.

Resumo

Aplicações de Mobile Crowd Sensing (MCS) são aquelas onde os participantes ou usuários coletam (através dos diferentes sensores em seu dispositivo móvel - microfone, câmera, GPS, entre outros) e compartilham informações com o intuito de auxiliar outros usuários na tomada de decisões (baseada na opinião de terceiros) ou informar sobre os mais diversos acontecimentos. Entre exemplos comuns de aplicações de MCS pode-se citar a cobertura de notícias instantaneamente, informações sobre pontos de congestionamento, desastres climáticos, poluição do ar, buracos em vias públicas, vagas em estacionamentos, entre outros. Uma vez que o funcionamento de aplicações MCS envolve a participação humana ou mecânica (automóveis, drones, sensores, entre outros) para coletar os dados que serão usados por usuários finais, a privacidade e a segurança dos participantes, sem falar na confiabilidade das informações geradas e recebidas são questões importantes que precisam investigadas. Neste contexto, este Capítulo objetiva fornecer entendimento sobre o ambiente de aplicações de Mobile Crowd Sensing, focando principalmente nas questões relacionados a privacidade, a segurança e a confiabilidade das informações.

2.1. Introdução

Há algum tempo, o processo de sensoriamento passou a ser utilizado na gestão de cidades, através da monitorização de áreas urbanas e da observação da dinâmica das comunidades, visando a fornecer aos gestores informações essenciais para a tomada de decisão sobre os mais variados assuntos. Atualmente, esse processo de gestão é parte essencial da vida nas cidades. Por exemplo, o sensoriamento de fatores ambientais permite que autoridades ou agências obtenham dados e informem a população sobre condições de tráfego, poluição sonora, poluição do ar, qualidade da água, segurança pública, entre outros assuntos, informando o que acontece, quando acontece e o que fazer quando algo acontecer. Mas como fazer esse sensoriamento?

As atuais e tradicionais técnicas de sensoriamento, tais como as redes de sensores sem fio (RSSF), vêm sendo muito aproveitadas para adquirir as "condições" do mundo real. Contudo, as redes de sensores comerciais nunca foram implantadas com sucesso no mundo real, devido a problemas como a cobertura insuficiente, falta de escalabilidade e, principalmente, o custo de instalação e manutenção.

Graças ao exponencial crescimento do poder computacional e da quantidade e disponibilidade de sensores embutidos nos dispositivos móveis usados diariamente, é possível,nos dias atuais, que o cidadão comum possa monitorar aspectos sobre a cidade, sua comunidade ou uma região e, assim, contribuir de alguma forma para melhorar e/ou auxiliar a vida das outras pessoas. Entre os exemplos óbvios estão o nível de ruído na cidade e o fluxo de tráfego. Também é possível incluir situações que interferem na qualidade de vida, como buracos em vias públicas e estradas, os quais podem ser detectados usando dispositivos móveis.

Recentemente, um novo paradigma de sensoriamento vem tirando proveito dessa vasta gama de recursos para monitorar e compartilhar informações de interesses comum, coletadas através dos sensores embutidos nos dispositivos móveis, o que acaba por auxiliar as pessoas na tomada de decisões. Denominado de *Mobile Crowd Sensing*, ou simplesmente MCS, esse paradigma parte do princípio de que é possível (e até fácil em certo ponto) utilizar os mais variados sensores (câmeras, acelerômetros, sistemas de posicionamento global - GPS, sensores de temperatura, entre muitos outros) presentes em *smartphones*, dispositivos de IoT (I*nternet of Thing*), tocadores de músicas, consoles de jogos (Wii e XboX Kinect, por exemplo) e veículos (GPS, computadores de bordo) para coletar informações relevantes para uma cidade ou uma comunidade.

As aplicações MCS vêm sendo utilizadas para o monitoramento do ruído nas cidades [Maisonneuve et al. 2010, Rana et al. 2010], poluição ambiental [IBM 2010] e fenômenos climáticos [Thepvilojanapong et al. 2010], medição da densidade demográfica [Weppner and Lukowicz 2013], cenários de emergências [Ludwig et al. 2015], medicina [Lane et al. 2010], anomalias no tráfego [Pan et al. 2013, Ganti et al. 2011] e até mesmo detectar terremotos [Minson et al. 2015].

Um aspecto importante é que as aplicações MCS vêm ganhando popularidade com a criação de diversos sistemas e aplicativos e, consequentemente, conquistando o envolvimento de mais e mais pessoas, redes e grupo de colaboradores. Em conjunto com outras tecnologias e plataformas, como rede sociais e armazenamento em nuvem,

as MCS têm desempenhado um papel imprescindível na integração entre pessoas que buscam informações ajustadas para tomar decisões que possam ajudá-las.

O diferencial de MCS em relação a outras abordagens de sensoriamento com participação de usuários reside no fato de que toda ação de registro (coleta de dados e informações) parte do dispositivo móvel do usuário participante do serviço, conectado a qualquer rede de acesso à Internet, muitas vezes sem a necessidade de registro em tempo real [Wang et al. 2013]. Assim, o custo de implantação e utilização torna-se bastante reduzido, uma vez que não se faz necessário construir uma infraestrutura específica, como ocorre em sensores convencionas. Serviços e aplicações de MCS podem usar a infraestrutura das próprias operadoras de telefonia móveis bem como redes domésticas e/ou empresarias através de WiFi e 3G Além disso, quanto maior o número de participantes mais tarefas podem ser disponibilizadas e mais áreas pode ser cobertas [Tuncay et al. 2012], o que acaba por aumentar a confiabilidade das informações, permitindo que que elas sejam mais facilmente aceitas.

Por outro lado, o uso de MCS também traz seus riscos. O principal deles é a privacidade [Ganti et al. 2011, Lane et al. 2010, Kong et al. 2015]. O cenário a seguir exemplifica melhor o problema. Considerando um sistema MCS cujo objetivo é recolher imagens ou vídeos curtos sobre irregularidades no trânsito em diferentes partes de uma cidade, qualquer pessoa que deseja participar precisa, primeiramente, enviar uma consulta ao servidor de aplicativos para descobrir o conjunto de locais próximos a ela que ainda necessitam de coleta de dados. Se o participante não estiver disposto a divulgar sua identidade, o servidor pode até remover a identificação (ID) da consulta, mas ainda precisa saber as informações de localização do participante, a fim de responder à consulta. Assim, devido a essa forte correlação entre os participantes e seus movimentos, um servidor malicioso pode identificar um participante através de sua informação de localização.

Diante do exposto, o objetivo deste Capítulo é introduzir o leitor no paradigma de *Mobile Crowd Sensing*. A ideia é apresentar os principais conceitos e definições sobre o tema. Contudo, o foco principal é voltado para a questão de segurança (privacidade e confiabilidade) das informações coletadas e transmitidas pelos participantes de MCS. Além de possibilitar a compreensão dos problemas de segurança em MCS e apresentar as soluções existentes, este Capítulo também identifica oportunidades de estudos na área, bem como aponta alguns trabalhos futuros.

Para alcançar os objetivos propostos, o restante deste Capítulo está organizado da seguinte forma. A Seção 2.2 caracteriza *Mobile Crowd Sensing*, apresentando conceitos, definições e características únicas, bem como discutindo a evolução dos esquemas de sensoriamento participativo até chegar às MCS e apresentando exemplos de aplicações reais. A Seção 2.3 trata dos aspectos de segurança em MCS, onde são discutidos os problemas de privacidade dos participantes e de confiabilidade dos dados. Os problemas causados e as técnicas empregadas como contramedidas a esses problemas são relatadas. A Seção 2.4 apresenta algumas soluções (arquiteturas, aplicações, bibliotecas e ferramentas) existentes que levam em consideração a segurança em MCS ou permitem que sejam empregadas nos aspectos de segurança em MCS. A Seção 2.5, por sua vez, aponta algumas questões em aberto. Por fim, a Seção 2.6 apresenta os comentários finais sobre o tema.

2.2. Mobile Crowd Sensing

Como já mencionado na Seção anterior, *Mobile Crowd Sensing* (MCS) apresenta um novo paradigma de sensoriamento baseado no poder dos dispositivos móveis. Ele tira proveito do grande número de dispositivos que "acompanham" um usuário (telefones celulares, dispositivos portáteis e veículos inteligentes) e de sua mobilidade para adquirir conhecimento local e compartilhar esse conhecimento dentro de uma esfera social.

Esta Seção discute o processo evolutivo do sensoriamento, traz definições sobre tais aplicações, enumera os elementos arquiteturais de sistemas MCS, apresenta as características únicas que o paradigma proporciona e termina exemplificando algumas aplicações MCS.

2.2.1. Da Sabedoria das Multidões a MCS

Para entender o paradigma de MCS, é preciso, primeiramente, compreender a ideia de resolver problemas usando populações como fonte de informação (definido em inglês como *crowd-powered problem-solving*).

Em seu livro "The Wisdom of Crowd" (Sabedoria das Multidões), Surowiecki [Surowiecki 2005] revela que a agregação de dados ou informações a partir de um grupo de pessoas resulta, muitas vezes, em decisões melhores do que as feitas por um único indivíduo . Na mesma linha de raciocínio da sabedoria das multidões, existe o conceito de inteligência coletiva, um tipo de "saber compartilhado" oriundo da colaboração de muitas pessoas em suas diversidades. Segundo Levy [Levy and da Costa 1993], a inteligência coletiva é aquela distribuída por toda parte, na qual todo o saber está na humanidade, já que ninguém sabe tudo, porém todos sabem alguma coisa.

Com base nos conceitos de sabedoria das multidões e inteligência coletiva, duas formas de aproveitar a participação de multidões, intrinsecamente ligados a MCS. A primeira delas é *Crowdsourcing*. Definido por Jeff Howe e Mark Robinson em [Howe 2006], *Crowdsourcing é o ato de uma empresa ou instituição, tendo uma função uma vez realizada por funcionários, terceirizá-la a uma rede indefinida (e geralmente grande) de pessoas sob a forma de um convite aberto*. *Crowdsourcing* é um modelo de produção que utiliza a inteligência coletiva, a cultura colaborativa e a formação de comunidades para solucionar problemas, criar conteúdo ou buscar inovação. Um exemplo típico é a Wikipedia, a maior enciclopédia do mundo, criada por milhares de contribuidores, de forma colaborativa.

A segunda forma é o *Sensoriamento Participativo* [Burke et al. 2006], que sustenta a ideia que cidadãos e dispositivos móveis podem formar redes de sensores participativas para aquisição e compartilhamento de conhecimento local, enfatizando a participação explícita do usuário. *Sensoriamento Participativo* requer um ativo envolvimento dos indivíduos para contribuir com dados sensoriados (uma foto, um relato sobre uma via pública interditada) de acordo com o fenômeno monitorado.

Embora as diferenças entre *Crowdsourcing*, *Sensoriamento Participativo* e MCS possam não ser facilmente percebidas, elas existem. Todas as três formas de sensoriamento assumem que existe a participação de multidões, uma vez que quanto maior número de participantes, mais dados podem ser sensoriados, maiores áreas podem ser

cobertas e o que aumenta a confiabilidade das informações coletadas.

Contudo, além da coleta de dados, MCS também permite (e algumas vezes exige) que o participante atue no processamento das informações coletadas. Além disso, MCS aceita a participação oportunista [Lane et al. 2010], onde é necessário apenas que o aplicativo capture os dados através dos sensores disponíveis, e a participação mecânica (carros inteligentes, drones, entre outros) na atividade de coleta [Konidala et al. 2013].

A Figura 2.1 ilustra a abrangência de MCS em comparação aos conceitos de sabedoria das multidões, *crowdsourcing* e sensoriamento participativo.

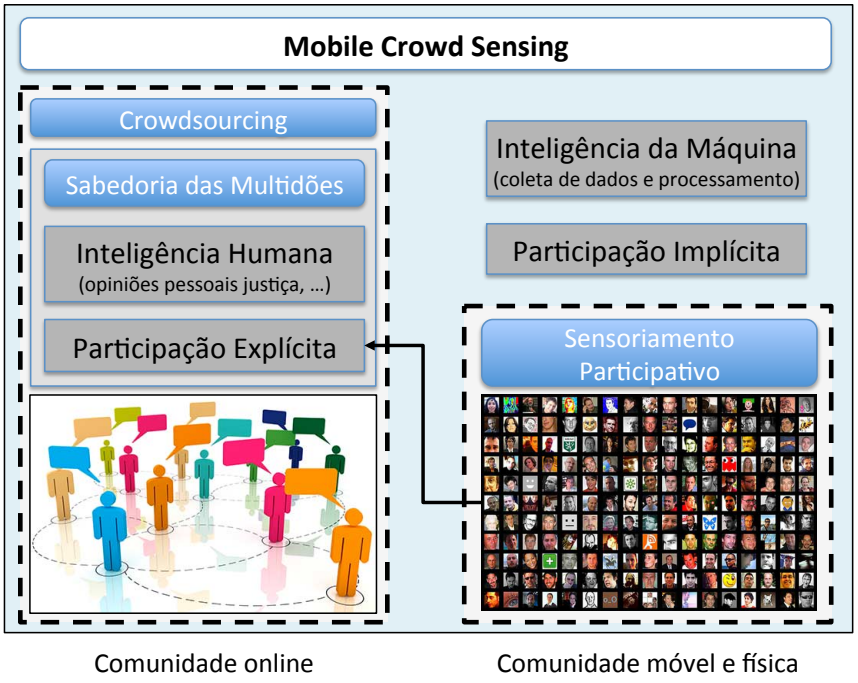


Figura 2.1. Comparação entre MCS e os conceitos relacionados. Adaptado de [Guo et al. 2015]

Percebe-se na Figura 2.1 que enquanto a sabedoria das multidões e *crowdsourcing* contam apenas com a inteligência humana, o sensoriamento participativo e MCS exploram uma fusão da inteligência humana e da máquina.

Segundo Hu et al. [Hu et al. 2013], em comparação com sistemas e redes de sensores fixos, com grande complexidade estrutural e logística, MCS possui vantagens como:

- **Generalidade**, uma vez que atendem a diversidade de sistemas operacionais e hardwares, ignorando nuances e generalizando o desenvolvimento de aplicações necessárias e populares;
- **Escalabilidade**, pois não limitam a quantidade de usuários;
- **Mobilidade de conteúdo**, já que as informações coletadas podem ser passadas por diversos canais (redes sociais, canais de rádio, jornais online e televisão, por exemplo), antes de serem acessadas/vistas, não havendo barreiras para as informações transmitidas.

2.2.2. Definições

Como já mencionado neste Capítulo, *Mobile Crowd Sensing* (MCS) apresenta um novo paradigma de sensoriamento baseado no poder dos dispositivos móveis.

Para Ganti et al. [Ganti et al. 2011], os pesquisadores responsáveis por cunharem o termo, *Mobile Crowd Sensing se refere a uma variada e ampla gama de modelos de sensoriamento no qual indivíduos com dispositivos computacionais e sensores são capazes de coletar e contribuir com dados valiosos para diferentes aplicações*. Em linhas gerais, MCS tira proveito do grande número de dispositivos que "acompanham" um usuário (telefones celulares, dispositivos portáteis e veículos inteligentes) e de sua mobilidade para adquirir conhecimento local e compartilhar esse conhecimento dentro de uma esfera social. A informação recolhida mais a fusão dos dados coletados (que acontece com o apoio da computação em nuvem), torna MCS uma plataforma versátil que muitas vezes pode substituir as infraestruturas de sensoriamento estáticos, permitindo uma ampla variedade de aplicações.

Guo et al. [Guo et al. 2015] definem MCS como "*um novo paradigma de detecção que capacita os cidadãos comuns à contribuir com dados sensoriados ou gerados a partir de seus dispositivos móveis, agregados e fundidos na nuvem para a extração de inteligência da multidão, prestando serviços centrados nas pessoas*". Já autores como [Dimov 2014] definem MCS como sendo um novo modelo de negócios do conceito de *crowdsourcing*, uma vez que permite que um grande número de dispositivos móveis sejam usados não somente para troca de informações entre seus usuários, mas também para atividades que podem ter um grande impacto social.

Quanto ao que pode ser sensoriado com MCS, Ganti et al. [Ganti et al. 2011] afirmam que tudo depende do fenômeno que se está medindo. Eles sugerem a seguinte classificação:

- **Ambiental**, onde fenômenos de natureza ambiental são os alvos. Exemplos incluem o nível e qualidade da água em rios [IBM 2010, Minkman et al. 2015], a medição dos níveis de poluição em cidades [Dutta et al. 2009, Leonardi et al. 2014] e o monitoramento de vida selvagem em seu habitat [Mediated Spaces, Inc 2015].

- **Infraestrutura** envolve a medição em larga escala de fenômenos relacionados a infraestrutura pública. Exemplos incluem a disponibilidade de vagas em estacionamentos [Mathur et al. 2010], a condição de rodovias e estradas, problemas com aparelhos públicos (hidrantes, iluminação, entre outros) e o monitoramento do trânsito em tempo real [Mohan et al. 2008, Google 2015].

- **Social**, onde os participantes coletam e compartilham informações entre si. Exemplos incluem dados sobre os exercícios [Reddy et al. 2007] que realizaram durante o dia ou quanto andaram de bicicleta [Eisenman et al. 2007].

2.2.3. Componentes Arquiteturais

Embora não exista uma arquitetura formal, visto que ela depende do que se quer sensoriar, a Figura 2.2 mostra uma arquitetura genérica de MCS.

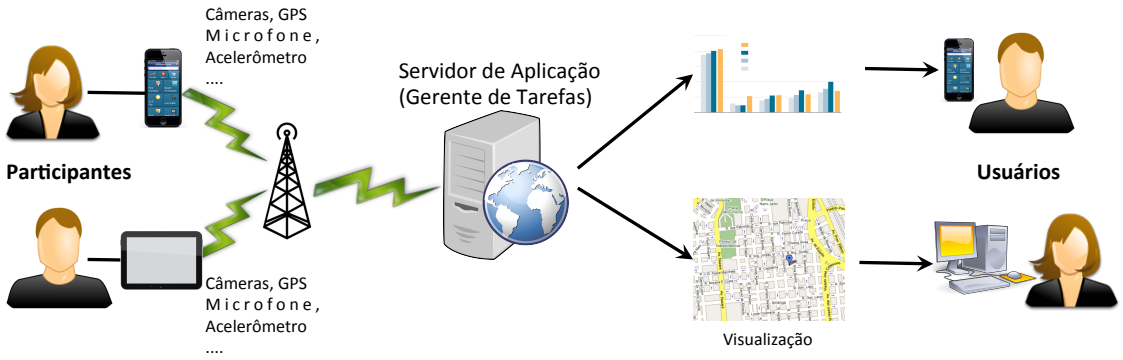


Figura 2.2. Exemplo de arquitetura geral em MCS. Adaptado de [Pournajaf et al. 2014]

Entretanto, vários autores [Pournajaf et al. 2014, Ganti et al. 2011] concordam que qualquer sistema MCS deve possuir pelos menos três componentes chave:

- **Participantes**: São as entidades, pessoas, que usam algum tipo de sensor para obter ou medir a informação requisitada sobre um elemento de interesse;
- **Aplicações ou Usuários Finais**: São as entidades que requisitam dados através de tarefas e então utilizam a informação coletada pelos participantes;
- **Servidor de Aplicação**: Também chamado de gerentes de tarefas, são as entidades responsáveis pela distribuição de tarefas aos participantes que conseguem atender aos requisitos das aplicações. O modo de distribuição de tarefas do servidor de aplicação varia de acordo com as tarefas e os participantes, e pode ser categorizado em:
  - **Centralizado**: Um servidor central fornece aos participantes diferentes tarefas para executar. Por exemplo, em um aplicativo que mede o entusiasmo de festas (*party thermometer*), um servidor central pode escolher um conjunto de participantes, pedindo que eles classificam uma determinada festa. Uma questão importante do modelo centralizado é ter um único ponto de falha para interações entre participantes e aplicações;
  - **Descentralizado**: Cada participante pode se tornar um gerente de tarefa e decidir ou executar uma tarefa ou passá-la para outros participantes que possam estar melhor adaptados para cumprí-la. Esta decisão pode ser baseada em certos atributos de outros participantes, como localização, habilidades ou hardware disponível no dispositivo. Um bom exemplo é o modelo de recrutamento descentralizado proposto em [Tuncay et al. 2012], que notifica os participantes qualificados de uma atividade de sensoriamento próxima;
  - **Híbrido**: Neste esquema, um servidor central e um conjunto de participantes atuam na construção do núcleo de gerenciamento de tarefas. Um bom exemplo é o esquema de bolha [Lu et al. 2010], que requer um servidor central para manter o controle das tarefas de sensoriamento, que são alocadas principalmente de forma descentralizada.

2.2.4. Ciclo de Vida de MCS

De acordo com Zhang et al. [Zhang et al. 2014a], o ciclo de vida em MCS consiste: (1) na criação de aplicativos de acordo com os requisitos; (2) na atribuição de tarefas de sensoriamento para os participantes; (3) na execução da tarefa (sensoriamento, computação e upload) no dispositivo móvel do participante, e (4) coleta e processamento dos dados enviados pelos participantes (Figura 2.3).

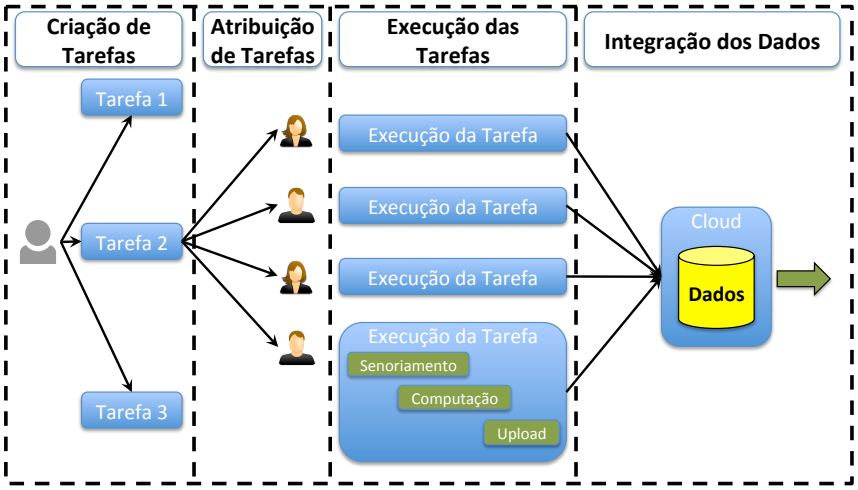


Figura 2.3. Ciclo de vida em MCS. Adaptado de [Zhang et al. 2014a]

As funcionalidades fundamentais de cada fase são descritos como:

- **Criação de tarefas**: O organizador/gerenciador MCS define uma tarefa de acordo com as necessidades estabelecidas e elabora uma aplicação MCS que deve ser fornecida aos participantes.

- **Atribuição de tarefas**: Após a tarefa MCS e a aplicação móvel serem criadas, a próxima fase é atribuir a tarefa, ou seja, recrutar participantes e lhes atribuir tarefas de sensoriamento individuais que são supostamente para serem executadas no dispositivo móvel de cada participante. Encontrar participantes suficientes e adequados para o sensoriamento é a questão central neste estágio.

- **Execução de tarefas individuais**: Depois de receber a tarefa, o participante deve tentar terminá-la dentro do período pré-definido. Esta fase pode ser dividida em 3 sub-fases - sensoriamento, computação e *upload* dos dados.

- **Integração de dados**: Esta etapa recebe os fluxos de dados coletados de todos os participantes como entrada, agrega-os e fornece aos usuários finais no formato adequado. Para algumas aplicações MCS [Sherchan et al. 2012], o processamento de dados nesta fase é bastante simples. Um servidor armazena os dados centrais e fornece interfaces para usuários consultarem e partilharem os dados. Enquanto outros aplicativos MCS [Mun et al. 2009, Rachuri et al. 2011] empregam algoritmos complicados para integrar dados e extrair alto nível de inteligência coletiva a partir dos dados brutos.

2.2.5. Características Únicas

De acordo com Guo et al. [Guo et al. 2015], MCS possui quatro características chave que a diferenciam das outras abordagens. São elas.

Sensoriamento baseado em Multidões

Em comparação com redes de sensores tradicionais, a diferença fundamental de MCS é o envolvimento de multidões para o sensoriamento em grande escala, o que oferece duas grandes vantagens. A primeira é aproveitar a capacidade de sensoriamento nos dispositivos dos participantes bem como a infraestrutura de comunicação existente, facilitando a implantação e reduzindo custos. A segunda é aproveitar a mobilidade inerente dos usuários de dispositivos móveis, o que oferece uma cobertura espaço-temporal sem precedentes comparada às implementações de redes de sensores estáticas.

Além disso, o sensoriamento através de multidões de MCS oferece novos recursos. O primeiro deles é o **modo de geração de dados**. MCS pode reunir (sensoriar) dados de dois modos diferentes:

1. **Sensoriamento Móvel**, onde o processo de coleta é baseado no contexto individual, ou seja, o sensoriamento é feito a partir dos dispositivos dos indivíduos [Lane et al. 2010];

2. **Redes Sociais Móveis** (*Mobile Social Networks* ou MSN), onde os dados postados pelos usuários em serviços de redes sociais contribuem, em larga escala, como outra fonte de dados.

A combinação desses dois modos é uma característica única de MCS.

O segundo é o **estilo de sensoriamento**. Ganti et al. [Ganti et al. 2011] afirmam que o sensoriamento em MCS pode ser classificado em **participativo** e **oportunista**, dependendo dos requisitos das aplicações e dos recursos do dispositivo. O sensoriamento é participativo, também chamado de **explicito**, quando a tarefa de sensoriamento é informada pelo usuários. Em outras palavras, a geração de dados (sensoriamento) é feita a partir do dispositivo do usuário de forma individual ou baseada no contexto. Já o sensoriamento oportunista, como o próprio nome diz, tira proveito das interações online envolvendo elementos físicos (por exemplo, *check-in* em lugares) ocorridas em redes sociais móveis como fonte de dados. Assim, os dados coletados são enviados pelos usuários de serviços de redes sociais móveis. No entanto, os dados são usados para um segundo propósito (o objetivo principal é a interação social online), e, portanto, ela é realizada de modo **implícito**. Devido essa caracterização de dois estilos de sensoriamento, Guo et al. [Guo et al. 2015] afirmam que MCS apresenta uma nova dimensão: a consciência do usuário na realização do sensoriamento.

O terceiro é a **organização de voluntários**. Os participantes podem ser cidadãos auto-organizados com diferentes níveis de envolvimento organizacional, variando de totais estranhos, grupos pouco organizados de vizinhos que enfrentam um problema comum até grupos bem-organizados de ativistas previamente existentes [Maisonneuve et al. 2010]. Os voluntários podem ser organizados em:

1. *Grupos*, pessoas organizadas de forma oportunista (tipicamente vizinhos) que desejam abordar de forma colaborativa um problema enfrentado por todos;

2. *Comunidades*, onde as pessoas que vivem em uma determinada área se unem por causa de seus interesses comuns;

3. *Urbanos*, onde qualquer cidadão (na sua maioria estranhos) pode participar da atividade de sensoriamento em escala urbana.

Sistema Centrado no Usuário

Como já mencionado neste Capítulo, MCS destaca-se pela capacidade de integrar a inteligência humana e da máquina. Contudo, para motivar a participação plena dos usuários bem como melhorar sua experiência de uso, sistemas MCS são desenvolvidos centrados no usuário (*user-centric*).

Embora vital, tal fato traz alguns problemas. O primeiro deles é como **motivar os usuários a participarem**? A primeira vista, a promessa de ganho financeiro é um método importante e bastante usado como incentivo. Contudo, o simples fato do entretenimento pode ser motivador em muitas situações, mesmo quando não há perspectiva de ganhos financeiros [Ganti et al. 2011]. As pessoas também podem ser motivadas a participar por razões éticas e sociais, tais como a socialização com outros ou o reconhecimento. Vale lembra que os usuários tem noção de que ao usar seus dispositivos e sensores estarão consumindo seus próprios recursos (energia e dados móveis, por exemplo).

Outro problema é a **segurança e privacidade do usuário**. O compartilhamento de dados pessoais em sistemas MCS pode levantar preocupações quanto a privacidade. Para motivar a participação do usuário, é essencial que novas técnicas para proteção da privacidade do usuário sejam elaboradas, permitindo que seus dispositivos possam contribuir de forma confiável. De forma particular, a definição de segurança e privacidade pode e precisa evoluir nos sistemas MCS, já que informações pessoais podem não ser obtidas diretamente, mas sim inferidas a partir de dados agregados. Por exemplo, o fato de que um objeto com uma etiqueta RFID poder ser identificado exclusivamente e rastreado de volta até seu utilizador pode trazer muitos problemas de privacidade [Acampora et al. 2013].

Requisitos de Comunicação

O sucesso de qualquer solução MCS depende de recursos de comunicação e de conexões heterogêneas temporárias que permitam a coleta eficiente dos dados sensoriados. Embora aplicações e sistemas MCS possam ter arquiteturas de comunicação diferentes, grande parte delas baseai-se em quatro pontos.

O primeiro é a **conexão de rede heterogênea**. Os atuais dispositivos móveis são geralmente equipados com múltiplas interfaces e tecnologias de comunicação sem fio (por exemplo, GSM, Wi-Fi e Bluetooth). Enquanto as interfaces GSM e WiFi podem fornecer conectividade com uma infraestrutura de comunicação pré-existente, Bluetooth ou Wi-Fi podem fornecer conexão de curto alcance entre dispositivos móveis e redes oportunistas auto-organizadas para compartilhar dados [Conti and Kumar 2010, Guo et al. 2013].

O segundo é a **topologia de rede e mobilidade humana**. A mobilidade dos dispositivos móveis e seus donos não só fornece uma boa cobertura de sensoriamento para

MCS, mas também traz desafios para as comunicações:

- A topologia da rede evolui ao longo do tempo, o que torna difícil encontrar rotas estáveis entre dispositivos móveis;
- Os protocolos de roteamento tradicionais, projetados para redes sem fio estáticas, não conseguem lidar com topologias altamente dinâmicas para cumprir as tarefas básicas de comunicação em MCS (especialmente para implementações ad hoc puras);
- Uma vez que a mobilidade humana desempenha um papel importante na dinâmica e no comportamento de sistemas MCS, os esforços em pesquisa sobre mobilidade humana [Karamshuk et al. 2011, Clementi et al. 2013] precisam avançar.

O terceiro é o **serviço tolerante a interrupções**. Em algumas aplicações MCS, os dados sensoriados não precisam ser transmitidos em tempo real ou com garantias de completude e precisão. Portanto, esses sistemas podem tirar proveito das redes tolerantes a interrupção [Gao and Cao 2011], que só dependem de conectividade de rede intermitente e têm custo de implantação muito menor. Vale lembrar que muitos dispositivos móveis não possuem garantias de conexão o tempo todo devido à fraca cobertura de rede (por exemplo, a força fraca de sinal devido a interferências ou nenhum sinal em uma área rural), restrição de energia (por exemplo, bateria fraca), ou preferências do usuário (por exemplo, telefone desligado em uma reunião) [Ma et al. 2014].

O quarto e último é a **exigência de alta escalabilidade**. Uma vez que MCS baseia-se em dados de sensoriamento de um grande volume de usuários móveis, a escalabilidade é claramente um requisito básico e um desafio fundamental para os sistemas de comunicação subjacentes. Para alcançar a escalabilidade suficiente, os protocolos de comunicação MCS e arquiteturas de rede são geralmente altamente distribuídas e descentralizadas. Tais soluções podem também melhorar a robustez do sistema global MCS. Além disso, o projeto eficiente de energia tem de ser considerado, devido aos recursos limitados de energia de cada dispositivo individual e do grande número de dispositivos no sistema.

Processamento de Dados e Inteligência na Extração

Uma vez que um dos objectivo de MCS é extrair inteligência de alto nível a partir de um grande volume de entradas, a participação humana no processo de sensoriamento traz algumas incertezas para os sistemas MCS.

Uma delas é a **baixa qualidade dos dados**, geralmente definida como o grau de como se encaixam os dados para utilização em operações, tomada de decisão e planejamento. Por exemplo, os participantes anônimos podem enviar dados incorretos, de baixa qualidade ou até mesmo falsos [Zhang et al. 2014b, Wang et al. 2011a]. Além disso, dados oriundos de pessoas diferentes podem ser redundantes ou inconsistentes ou o mesmo sensor pode sentir o mesmo evento sob diferentes condições (por exemplo, detecção de ruídos em ambiente com o celular no bolso ou na mão). Portanto, a seleção de dados é muitas vezes necessária para melhorar a qualidade dos dados, devendo ser explorados métodos de filtragem de faltas, estimativa de qualidade, incentivo ao participante especialista, entre outros.

Outra questão é a **mineração de dados heterogêneos**. Os dados em MCS são obtidos em comunidades físicas e virtuais tanto de forma offline quanto online. Diferentes comunidades representam maneiras distintas de interação (por exemplo, comentários, transferências online, localização offline) e conter conhecimento diferente (por exemplo, a amizade em comunidades online, os padrões de movimento em comunidades offline). Portanto, o problema é como associar, de forma eficaz, dados com diferenças espaciais.

2.2.6. Exemplos de Aplicações

Para melhor apresentar o paradigma, esta Seção descreve algumas aplicações voltadas para MCS.

CreekWatch

Desenvolvido pela IBM Almaden Research Center, o Creek Watch [IBM 2010] visa monitorar o níveis das águas de riachos, baseando-se em imagens coletadas por participantes. O aplicativo permite o uso do smartphone (somente versões para IOS estão disponíveis) para marcar uma posição (usando GPS), tirar fotos daquele ponto e informar observações perceptíveis pelo usuário como o nível da água, o fluxo e a presença de lixo. A Figura 2.4 ilustra alguns passos para o uso do Creek Watch.

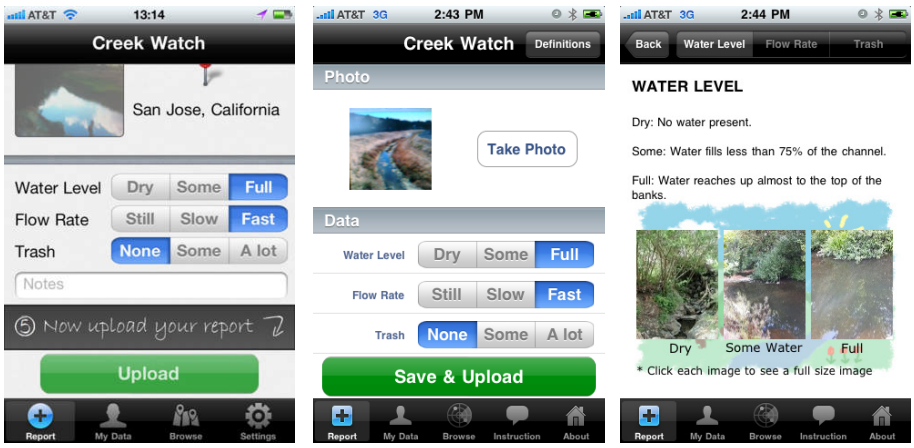


Figura 2.4. Exemplo de uso do Creek Watch

Os dados coletados são automaticamente enviados para o servidor central ou armazenados no próprio dispositivo se não houver acesso a Internet. Todas as informações podem ser acessadas e baixadas em `http://creekwatch.org`. De acordo com a IBM, o Creek Watch já conta com mais de 4.000 usuários em 25 países.

NoiseTube

NoiseTube é uma nova abordagem para a avaliação da poluição sonora que envolve o público em geral [Maisonneuve et al. 2009]. O objetivo é transformar telefones celulares equipados com GPS em sensores de ruído que permitam aos cidadãos medir sua exposição pessoal ao ruído no seu ambiente quotidiano.

Cada usuário pode contribuir compartilhando suas medições geolocalizadas bem como anotações pessoais para produzir um mapa de ruído coletivo. Uma vez que os dados coletados são enviados ao servidor, qualquer usuário pode ver suas próprias contribuições

e a de outros através do Web site ou visualizá-las usando o Google Earth. Essa maior possibilidade de exibição faz com que o NoiseTube forneca dados para convencer e auxiliar autoridades na tomada de decisão sobre o problema da poluição sonora. NoiseTube usa o conceito de translucidez social, que consiste em fazer os participantes e suas atividades se tornarem visíveis um para o outro.

A versão inicial do NoiseTube foi escrita em Java, focada em smartphones com sistema operacional Symbian/S60 e testada em um dispositivo Nokia N95 com 8GB. Além disso, um receptor de GPS externo (conectado via Bluetooth) foi utilizado. Atualmente, está disponível para plataformas iOS, Android e smartphones com Java ME. O servidor é implementado using Ruby on Rails, MySQL, Google Maps e Google Earth.

A Figura 2.5 ilustra algumas interfaces do NoiseTube, que representam a coleta e a visualização das medições.

Figura 2.5. Exemplo de interfaces do NoiseTube

Nericell

O Nericell é uma aplicação MCS voltada para monitorar o tráfego em ruas, avenidas e rodovias em tempo real, mas de forma oportunista [Mohan et al. 2008]. Desenvolvida pela Microsoft Research, a ideia é que quando um usuário participante coloca seu telefone no bolso e começa a dirigir, a aplicação irá automaticamente monitorar as condições da estrada e do trânsito, transmitindo certas informações para um serviço na nuvem para a agregação e geração de relatórios.

Um aspecto interessante da Nericell é o uso de vários sensores presentes em dispositivos móveis, tais como bluetooth, rádio celular, microfone, acelerómetro e GPS. Segundo os criadores, um sensoriamento rico e diversificado é fundamental no contexto das cidades, onde as condições das estradas tendem a ser variáveis (buracos), existem vários tipos de veículos (2 rodas, 3 rodas, carros, caminhões) e o fluxo de tráfego pode ser caótico (por exemplo, vários motorista buzinando em um congestionamento). Assim, por exemplo, o acelerômetro pode ser usado para detectar buracos e o microfone para detectar buzinas, ajudando a determinar a condição de trafegabilidade.

Além da questão do sensoriamento, Nericell também resolve certos aspectos técni-

cos como consumo de energia. Sobre consumo de energia, a aplicação emprega o conceito de sensoriamento por gatilho, onde um sensor relativamente barato em termos de energia é usado para desencadear a operação de um sensor de maior consumo energético, quando necessário. Por exemplo, o acelerômetro é utilizado para detectar uma alta incidência de frenagens, o que provoca então o sensoriamento baseado no microfone para verificar se há buzinas.

Waze

Waze é um popular sistema de navegação que usa *Crowdsensing* para oferecer informações de tráfego em tempo quase real. Criado em 2008, o Waze registrava aproximadamente mais de 50 milhões de usuários em 2013, ano em que foi comprado pelo Google. Waze recolhe periodicamente dados do GPS em dispositivos móveis e os utiliza para calcular a velocidade do veículo. Assim, pode fornecer informações úteis sobre as condições de tráfego em diferentes áreas. O sistema também oferece aos seus usuários alertas pré-definidos informando incidentes como engarrafamentos e pontos de verificação da polícia, bem como informações sobre as condições de tráfego. Também é possível usar subcategorias de incidentes para melhor especificá-los, por exemplo, "engarrafamento pesado" em vez de apenas engarrafamento.

A Figura 2.6 ilustra algumas interfaces do Waze.

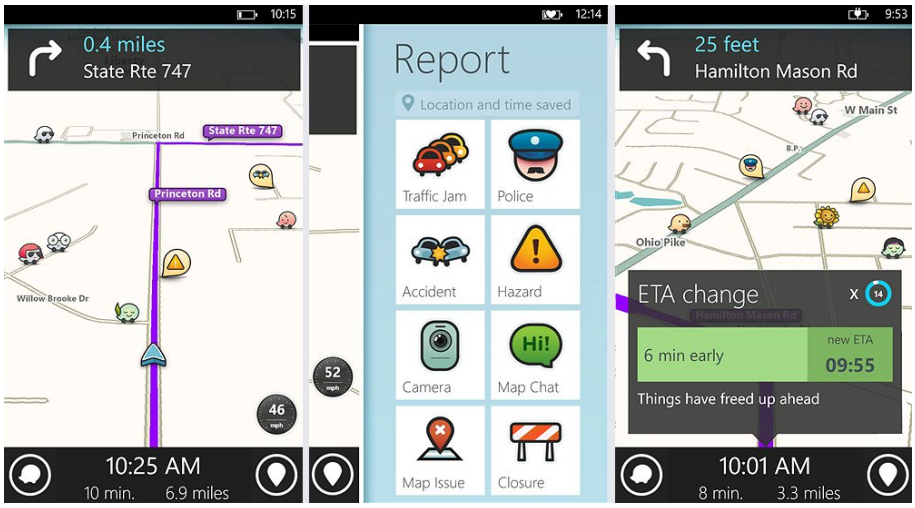


Figura 2.6. Exemplo de interfaces do Waze

mCrowd

mCrowd é uma plataforma de *Crowdsourcing*, desenvolvida em [Yan et al. 2009], que permite aos usuários móveis postar e trabalhar em tarefas relacionadas com o sensoriamento móvel distribuído. Ela permite que os usuários de iPhones possam utilizar, de forma plena, os diferentes sensores para participar e realizar tarefas de *Crowdsourcing*, incluindo a coleta de imagens com ciência de localização (geolocalização), marcação de imagens, monitoramento do tráfego rodoviário, entre outros. Obviamente, para incentivar a participação, é oferecida uma compensação financeira para cada usuário. Além de facilitar o *Crowdsourcing*, mCrowd torna viável que os usuários usem serviços populares como

Amazon Mechanical Turk (`https://www.mturk.com/mturk/welcome`) (Mturk) e ChaCha (`http://www.chacha.com`), através de uma interface única, simplificando assim sua participação no *Crowdsourcing*.

Por ser um aplicativo interativo e de fácil manuseio, o tratamento com as tarefas é um processo simples. Primeiro, as tarefas de interesse são postadas e em seguida os usuários capazes de executá-las são convocados para atuar. Diferentes serviços de *Crowndsourcing* podem ser usados, como Mtrurk ou ChaCha, para postar uma imagem ou texto. Ao acessar o mCrowd, o usuário pode escolher uma categoria de tarefas e postá-la no Mturk através de um mecanismo de busca. Após enviar a tarefa, uma lista com as tarefas é montada e a recompensa associada a tarefa.

A Figura 2.7 mostra a interface de usuário do mCrowd.

(a) Visão do Solicitante (b) Exibição de Tarefas (c) Resultados

Figura 2.7. Interfaces do mCrowd. Os usuários que desejam iniciar uma tarefa postam a partir da (a) visão do solicitante, as tarefas postadas são mostrados na exibição de tarefas (b) e os resultados apresentados pelos participantes são mostradas em resultados (c). Fonte: [Yan et al. 2009].

mCrowd foi estruturado para suportar quatro tipo de tarefas: marcação de imagem, captura de Imagem, consultas textuais e consultas baseadas em localização. Na marcação de imagem, os objetos de interesse são etiquetados ou marcados para serem consumidos pelos usuários, por exemplo, marcar um determinado objeto em uma imagem. Já na captura de imagens, a imagem é capturada quando há interesse em um determinado local, como parques de diversões. Consultas textuais são destinadas a obter as opiniões dos usuários como, por exemplo, quais os melhores restaurantes, pontos turísticos, entre outros. Consultas baseadas em localização buscam informações sobre locais mais próximos de um determinado ponto (por exemplo, restaurantes, floriculturas, entre outros).

2.3. Privacidade e Confiabilidade em Mobile Crowd Sensing

Esta Seção discute os dois principais problemas de segurança enfrentados por soluções, sistemas e aplicações MCS: a privacidade do usuário e a confiabilidade dos dados.

Em relação a privacidade, o problema é a possibilidade da divulgação de informações privadas (identidades, endereços físicos e na Internet, rotas, estilo de vida, entre muitas outras), uma vez que aplicações MCS gerenciam grandes volumes de informação que tipicamente são e devem ser tornadas públicas.

Já quanto a confiabilidade dos dados, o problema é que não existem garantias de que ou os dados inseridos pelos participantes são reais ou que as informações mantidas nos servidores ainda são reais.

2.3.1. Privacidade

O que é privacidade? Na área do Direito Civil, conceitua-se privacidade como a "faculdade que tem cada indivíduo de obstar a intromissão de estranhos em sua vida privada e familiar, assim como de impedir-lhes o acesso a informações sobre a privacidade de cada um, e também impedir que sejam divulgadas informações sobre esta área da manifestação existencial do ser humano" [Vieira and Alves 2014]. Em outras palavras, privacidade é o direito de cada indivíduo de manter e controlar o conjunto de informações que o cerca, podendo decidir se, quando, por que e por quem essas informações podem ser obtidas e usadas. Esse conjunto de informações incluem desde o seu modo de vida, as relações familiares e afetivas, os segredos, os pensamentos, os hábitos, os fatos e até mesmo os planos de futuro.

No paradigma MCS, devido suas características únicas, a privacidade envolve o direito do usuário e/ou participante de permanecer livre de intrusos e autônomo. A privacidade em MCS traz preocupações com a divulgação direta da identidade dos participantes bem como com a divulgação de atributos sensíveis, incluindo locais (por exemplo, endereço residencial ou do trabalho) e outras informações privadas como atividades pessoais ou condições (por exemplo, estilo de vida ou doença) que permitam inferir sobre a identidade dos participantes [Pournajaf et al. 2014].

Do ponto de vista dos participantes, as ameaças à privacidade podem ocorrer quando:

1. O participante recebe uma tarefa específica e compartilha suas preferências durante a atribuição dessa tarefa ou notifica o servidor que aceitou a tarefa. Neste momento, alguns atributos como a localização, o tempo, os tipos de tarefas que o participante está interessado ou alguns atributos do sensor que ele dispõe podem ser revelados. Embora seja possível argumentar que estas informações por si só podem não violar a privacidade, o fato é que elas podem permitir que alguém (atacante, adversário) possa rastrear as tarefas selecionadas pelo participante e, consequentemente, revelar sua identidade ou outros atributos sensíveis [Shin et al. 2011]. Alguns dos atributos que podem ser usados para rastrear os participantes são seus IDs, endereços IP ou outras informações de rede.

2. O participante realiza ou participa de tarefas espaciais com frequência. Essa repetição pode levar à divulgação de atributos sensíveis, como endereço residencial ou, eventualmente, sua identificação através de ataques de inferência [Krumm 2007]. Nestes tipos de tarefas, mesmo que o participante esteja usando o aplicativo de forma anônima, sua trajetória pode revelar locais sensíveis.

3. No processo de definição de tarefas, uma entidade maliciosa (aplicativo ou uma entidade separada) cria tarefas que impõem limitações estritas sobre os atributos do participante ou o dispositivo que está carregando (por exemplo, exigindo um estilo de vida especial ou um tipo de sensor raro para se qualificar para a tarefa). Este ataque pode resultar na divulgação da identidade ou outros atributos sensíveis do participante que aceita uma tarefa tão rigorosa [Shin et al. 2011].

4. No processo de distribuição de tarefas, uma entidade maliciosa (aplicativo, uma entidade separada ou outro participante) pode compartilhar tarefas para um conjunto limitado de participantes para ser capaz de aprender seus atributos ou rastreá-los [Shin et al. 2011] (por exemplo, empurrar ou atribuir uma tarefa a apenas um participante).

5. Várias aplicações ou usuários finais podem conspirar para conectar-se a informação dos participantes, com a finalidade de desanonimização. O usuário final malintencionado pode criar várias aplicações em uma tentativa de coletar mais dados privados.

De acordo com Christin et al. [Christin et al. 2011], as principais informações reveladas quando ocorrem quebras na privacidade dos participantes são:

- **Tempo e Localização**: A divulgação desses dois tipos de dados tem si mostrado altamente sensível à privacidade dos participantes, uma vez que permitem inferir ou de fato obter endereços residências e do local de trabalho, bem como as rotinas e hábitos dos participantes [Shilton 2009]. Por exemplo, visitas frequentes a hospitais podem permitir que os empregadores infiram sobre a condição médica de seus empregados, assim como a participação em eventos políticos pode fornecer informações sobre os pontos de vista políticos dos utilizadores [Liu 2007]. Em resumo, sem qualquer mecanismo de protecção, a divulgação de informações de localização pode levar à graves consequências sociais [Shilton 2009]. Além disso, as ameaças resultantes do rastreamento de localização e tempo não se limitam a aplicações onde é necessária autenticação. Mesmo no caso de contribuições anônimas, registros de localização podem ser analisados para deduzir a identidade dos participantes com base na localização de sua residência e consultas as listas telefônicas [Mun et al. 2009].

- **Amostras de Som**: Além inferir identidades e preferências a partir de dados espaço-temporais (tempo e localização), a "imagem" dos usuários pode ser refinada com amostras de outras modalidades de sensoriamento. Um bom exemplo são as amostras de som registradas intencionalmente pelos participantes ou capturadas automaticamente pelos dispositivos móveis. Enquanto os participantes podem facilmente preservar a sua privacidade somente gravando eventos não-sensíveis no primeiro caso, os dispositivos móveis, de forma eficaz, se comportam como espiões inteligentes no caso de gravações automatizados. Mesmo em locais públicos, o reconhecimento de padrões sonoros característicos, que são exclusivos para determinados eventos e locais, podem permitir que adversários determinem o contexto atual de um participante.

- **Fotos e Vídeos**: O conteúdo de imagens e vídeos gravados também é susceptível a revelar informações pessoais sobre os participantes e seu ambiente. Por exemplo, a aplicação SensoDiet [Reddy et al. 2007] marca as fotos tiradas das refeições consumidas e não existe qualquer contramedida para esconder os rostos das pessoas que compartilham a refeição com os participantes. Em todos os cenários em que a câmera é orientada longe do participante, rostos de outras pessoas na vizinhança são possivelmente capturados nas imagens, e, portanto, as conclusões sobre o número e a identidade das relações sociais do participante podem ser feitas. A publicação de imagens capturadas pode levar a consequências semelhantes como as em redes sociais online, como o Facebook. Semelhante a gravações de som, o contexto do utilizador atual e o ambiente em volta também podem ser extraídos a partir dos dados do sensor. Por exemplo, imagens que mostram pontos de interesse podem facilmente estabelecer a presença do participante nesses locais.

- **Acelerômetro**: É fácil pensar que as leituras de acelerômetros presentes em dispositivos móveis são os dados menos importantes no que diz respeito a revelar informações pessoais sobre os participantes. No entanto, essa hipótese nem sempre é verdadeira e muitas vezes pode servir como uma falsa sensação de segurança. Por exemplo, se o celular está na cintura, informações sobre a marcha, a forma de caminhar, correr, entre outras, podem gerar possíveis indicações sobre a identidade de um usuário [Derawi et al. 2010]. Assim, empregadores podem querer verificar se seus funcionários estão realmente trabalhando durante suas horas de trabalho.

- **Dados ambientais**: Dados ambientais (registro de partículas, das concentrações de gás ou pressão barométrica) podem não representar uma ameaça direta a privacidade dos participantes. No entanto, esses registros, quando combinados com informações secundárias, tais como temperatura, podem revelar a localização dos participantes a um nível de granularidade melhor (um quarto dentro de um edifício) do que as obtidas via GPS ou outros serviços de localização.

- **Dados biométricos**: Da mesma forma que dados de um sensor biométrico podem ser usados para o diagnóstico do estado fisiológico do usuário, atacantes e adversários podem identificar anomalias de saúde ou doenças com base nos dados capturados. Informações médicas vazadas podem ser utilizadas por companhias de seguros de saúde ou empregadores para revogar contratos, se um agravamento das condições fisiológicas dos participantes é identificado.

Como forma de manter a privacidade dos usuários, e ao mesmo tempo disponibilizar as informações coletadas, várias técnicas e métodos de segurança são empregados. São elas [Christin et al. 2011]:

2.3.1.1. Preferências do Usuário

Permitir aos participantes expressarem suas preferências de privacidade é uma técnica que visa controlar o processo de coleta de dados junto ao dono do dispositivo sensor, evitando assim danos à privacidade dos participantes. Além de simples, também permite a aplicação em diferentes graus.

Das et al. [Das et al. 2010] propuseram um esquema binário (acesso completo aos dados do sensor ou nenhum acesso) que pode ser estendido através da introdução de níveis intermediários adicionais. Por exemplo, os participantes podem decidir ativar seletivamente as medições do sensor, dependendo de uma variedade de fatores, como a presença em locais sensíveis (casa ou escritório), e de pessoas de seu convívio social (presença de amigos ou familiares). A seleção desses fatores permite aos participantes expressamente indicar quais tipos de informação eles estão aptos e felizes a coletar em diferentes contextos.

Os autores argumentam que um esquema com granularidade mais fina nas preferências de sensoriamento pode permitir aos participantes ajustar tanto a atividade quanto o tempo de atuação. Por exemplo, amostras podem ser recolhidas a cada hora ao invés de a cada 15s ou as informações de localização podem ser capturadas em diferentes graus de granularidade.

Embora essa técnica não forneça claramente um *trade-off* entre privacidade e dados divulgados, ela obviamente melhora a aceitação global de uma solução ao oferecer granularidade de configurações de privacidade para os participantes. Vale ressaltar que embora algumas soluções permitem que os usuários desativem totalmente sua função sensora [Miluzzo et al. 2008, Shilton et al. 2008], isso é de pouca utilidade para MCS.

2.3.1.2. Distribuição de Tarefas Anonimamente

A coleta de dados de um sensor é geralmente desencadeada por meio de tarefas que especificam as modalidades de sensoriamento (por exemplo, regiões de interesse, critérios a cumprir para iniciar a captura, sensores usados e frequência de amostragem) com base nos requisitos da aplicação [Reddy et al. 2009]. Assim, as tarefas são distribuídas para os dispositivos que satisfaçam os requisitos das tarefas.

Como explicado na Seção 2.2.2, a distribuição de tarefas pode ocorrer de forma centralizada - através de um servidor de tarefas ou aplicativos, que seleciona os dispositivos apropriados com base em critérios pré-determinados para otimizar o processo de sensoriamento, de forma descentralizada, onde os próprios dispositivos, de forma autônoma, gerenciam e distribuem tarefas ou de forma híbrida. Em todas essas três abordagens, os processos de contato e envio das tarefas podem colocar a privacidade dos participantes em perigo de várias maneiras, uma vez que baixar tarefas fornece informações ao servidor de tarefas sobre a localização dos participantes, com data e hora precisas. Além disso, a natureza das tarefas fornece dicas sobre os dispositivos utilizados.

Como forma de proteger os participantes, foram propostos alguns mecanismos para assegurar o anonimato e a privacidade de localização:

- Transferência das tarefas somente em locais densamente povoadas [Shin et al. 2011], já que a alta densidade de pessoas presentes nesses locais torna a identificação dos participantes pelo servidor difícil e, portanto, esconde suas identidades.
- Autenticação baseada em atributos [Kapadia et al. 2009], onde ao invés de usar a sua identidade, os participantes podem usar credenciais baseadas em criptografia

que comprovam sua participação em um determinado grupo (por exemplo, os alunos matriculados no clube de ciclismo da universidade);

- Esquemas de roteamento para preservação da privacidade de localização, que visam esconder a localização dos participantes [Kapadia et al. 2009] utilizando rotas específicas.

2.3.1.3. Anonimato

Segundo o Dicionário Aurélio, anonimato significa: (1) qualidade do que é anônimo; (2) sistema de escrever anonimamente, sem se identificar; e (3) o mesmo que anonimado. De forma geral, anonimato é a qualidade ou condição de permanecer anônimo. Em segurança da informação, anonimato é o que permite ocultar qualquer informação que possa identificar um usuários antes de compartilhar informações.

Embora o uso de técnicas de anonimato seja muito abrangente, em MCS a intenção é remover qualquer informação que possa identificar os usuários/participantes ou outras entidade durante a distribuição e realização de tarefas [Pournajaf et al. 2014]. Os métodos de anonimato mais usuais para proteção da privacidade em MCS são:

Pseudônimos

Pseudônimo é um mecanismo comum e até certo ponto simples para proteção do anonimato e da privacidade dos participantes [Christin et al. 2011], visto que ao invés de transmitir nomes em texto simples, toda a interação com o aplicativo é executado sob um pseudônimo. O uso de pseudônimos em tarefas de sensoriamento já é bem conhecido e é empregado nos trabalhos de Shilton et al. [Shilton et al. 2008], Shilton [Shilton 2009] e Deng e Cox [Deng and Cox 2009] e mais recentemente em Konidala et al. [Konidala et al. 2013] e Gisdakis et al. [Gisdakis et al. 2014].

Vale ressaltar que o uso de pseudônimos não garante necessariamente a privacidade, especialmente em aplicativos baseados em localização. Todos autores que empregam esse mecanismo são unanimes em afirmar que quando usado em conjunto com outros esquemas de autenticação, soluções à base de pseudônimos garantem anonimato e confidencialidade para os participantes.

K-Anonimato

Além dos esquemas de roteamento, mecanismos baseados no *k-anonimato* podem ser aplicados para proteger a privacidade localização dos participantes quando recebendo tarefas ou enviando os dados. A ideia chave por trás *k-anonimato* [Sweeney 2002] é construir grupos de k participantes que partilham um atributo comum (por exemplo, os k participantes localizados no mesmo distrito), tornando-os indistinguíveis uns dos outros.

Diferentes métodos vêm sendo utilizados para encontrar um atributo comum apropriado e assim construir grupos de k usuários. Estes métodos podem ser classificados em

duas categorias principais de generalização e perturbação [Huang et al. 2010b]. No primeiro caso, o valor original do atributo é generalizado por um valor com menor grau de detalhe. Por exemplo, as coordenadas exatas dos k participantes são substituídas pelo nome do distrito da localização atual.

Em Shin et al. [Shin et al. 2011], os autores usaram uma forma de generalização baseada na divisão de uma área geográfica em várias regiões, chamada de *Tessellation*, para mapear os pontos de acesso. O ponto de acesso no centro de cada região mantém um registo do número médio de dispositivos ligados, que é igual ao valor máximo K que pode ser conseguida dentro da região. Para garantir *k-anonimato* em toda a rede, regiões com $K < k$ são combinados em células com um valor igual à soma de cada região individual. Uma vez que as células tenham sido definidas, os participantes marcam seus dados com o limite geográfico da sua célula atual ao invés de fornecer suas coordenadas exatas.

Por outro lado, a perturbação é baseada na substituição dos dados dos sensores originais por um novo valor resultante de uma função aplicada às leituras dos sensores de k membros do grupo. Por exemplo, a localização de cada membro do grupo pode ser substituída pela localização média de todos os membros. Domingo-Ferrer e Mateo-Sanz [Domingo-Ferrer and Mateo-Sanz 2002] usam micro-agregação para substituir o local real pela localização mais próxima dos k participantes.

Um risco a soluções de *k-anonimato* é a possibilidade de ataques de homogeneidade, como apresentado por Machanavajjhala et al. [Machanavajjhala et al. 2007]. Esses ataques exploram a monotonia de determinados atributos para identificar indivíduos a partir do conjunto de k participantes.

Ocultação Seletiva

Locais sensíveis podem ser selecionados pelos participantes e protegidos usando a ocultação seletiva de localização [Mun et al. 2009]. A ideia é que quando um usuário se aproxima de um local que foi definido previamente como sensível, a aplicação gera registros fictícios de localização para evitar que o local seja selecionado. Obviamente, os registros gerados são e precisam ser realistas, ou seja, devem realmente representar estradas e ruas existentes. Em geral, um algoritmo de ocultação seleciona, primeiro, os lugares mais próximos e, em seguida, refina a seleção levando em conta o histórico de resultados dos participantes (por exemplo, suas capacidades físicas com base em suas experiências anteriores). Além disso, o algoritmo ainda altera as atividades e modifica sua duração para manter a consistência dos resultados da aplicação.

Quando comparado à *k-anonimato*, o esquema de ocultação seletiva melhora a privacidade de localização sem impactar os resultados da aplicação.

Agregação de dados

Em MCS, os dados precisam ser agregados e isso é normalmente feito pelo servidor de aplicação ou algum serviço próprio na nuvem. Contudo, a abordagem de agregação

de dados para preservação da privacidade proposta em Shi et al. [Shi et al. 2010] não depende de uma entidade central para proteger a privacidade de dados, mas sim da proteção mútua entre participantes. Antes de transmitir os dados para o servidor, os dispositivos móveis distribuem parcialmente seus dados entre os vizinhos. Em seguida, fazem o upload dos dados de sensoriamento vindo de seus vizinhos e os restantes de seus próprios dados. Esta distribuição diminui a probabilidade de atribuir, com sucesso, cada leitura do sensor para o dispositivo móvel que realmente a capturou. Por exemplo, se os dois dispositivos móveis (A e B) trocam metade de seus dados, a probabilidade de que os dados reportados por A ser realmente capturado por ele mesmo é só de 50% e o mesmo para B.

Dependendo da natureza das funções de agregação, dois regimes distintos podem ser aplicados. Para funções aditivas, cada dispositivo móvel particiona seus dados em $n+1$ fatias e envia uma fatia para cada um de n nós selecionados. Uma vez que cada nó distribuiu suas fatias para seus vizinhos, as fatias trocados e própria fatia do nó são combinadas e enviadas para o servidor de agregação que é então capaz de calcular o resultado agregado. Para as funções de agregação não-aditivas, tais como percentuais e histogramas, um método que combina corte, consulta de contagem e busca binária pode ser aplicado. No entanto, esta abordagem só assegura a proteção da privacidade de dados se os nós e o servidor não conspirarem para violar a privacidade de alvos potenciais.

2.3.1.4. Processamento de Dados

Em aplicações típicas de MCS e sensoriamento participativo, o processamento de dados é compartilhado entre os dispositivos móveis e servidores de aplicativos. No entanto, devido às limitações de recursos em plataformas móveis, a distribuição das tarefas de processamento entre ambas as partes é tipicamente feita em direção ao servidor. Embora o pré-processamento seja geralmente levado a cabo sobre os dispositivos para reduzir a quantidade de dados para transformar, a fim de economizar largura de banda e energia de transmissão, processamento de tarefas complexas pode exceder o poder computacional de dispositivos móveis, obrigando a execução no servidor.

O processamento de dados no dispositivo móvel consiste, principalmente, na extração de características, a partir dos dados brutos, a fim de remover informações sensíveis (por exemplo, vozes humanas gravadas ou pessoas fotografadas) que possam pôr em perigo a privacidade dos participantes e também para economia de recursos. Por exemplo, um classificador de áudio pode analisar as amostras de som para determinar se vozes humanas foram registradas [Miluzzo et al. 2008]. Além disso, o nível de intensidade das amostras de áudio pode ser determinado localmente através da execução de algoritmos de processamento de sinal para minimizar os dados a serem transferidos para o servidor [Maisonneuve et al. 2009]. Após processamento, os dados brutos podem ser excluídos do armazenamento local e os resumos processados são relatados ao servidor central.

No lado do servidor, os dados relatados podem então ser processado para eliminar informações sensíveis sobre a privacidade, tais como as características de identidade ou dados que ameaçam o anonimato/privacidade dos participantes. Por exemplo, os dados capturados podem ser agregados entre vários participantes para torná-los indistinguíveis ou publicados sob a forma de estatísticas [Ganti et al. 2011] e mapas [Dong et al. 2008].

Ao fazer isso, os dados confidenciais não estão diretamente relacionados aos usuários finais, o que evita a identificação direta dos participantes. No entanto, os participantes devem contar com o aplicativo para: anonimizar eficientemente os dados, proteger suficientemente sua privacidade, e não divulgar informações sensíveis à privacidade em seus dados comunicados a terceiros.

2.3.1.5. Controle de Acesso e Auditoria

Dependendo do cenário de aplicação, os resultados de um sensoriamento podem não ser só de interesse dos participantes, mas também de outras pessoas como, por exemplo, pesquisadores, pessoal médico, amigos, familiares, conselhos municipais ou até mesmo um público maior. No entanto, os participantes podem não estar dispostos a partilhar seus dados com todos os tipos de pessoas dentro do grupo de pessoas interessados com a mesma granularidade. Além de abordar questões de privacidade antes do processo de sensoriamento e da liberação de dados para a aplicação, os participantes podem definir o público-alvo que está autorizado a acessar seus dados a partir da interface de usuário de muitas aplicações.

Os participantes podem definir grupos [Grosky et al. 2007, Gaonkar et al. 2008], selecionar certas pessoas [Reddy et al. 2007, Gaonkar et al. 2008, Shilton 2009] ou autorizar todos [Gaonkar et al. 2008]. Podem também refinar sua seleção, especificando a natureza dos dados que compartilham, bem como definir subconjuntos específicos de dados acessíveis [Reddy et al. 2007, Miluzzo et al. 2008, Shilton 2009]. Além disso, eles podem definir condições precisas de liberação do acesso aos dados [Shilton et al. 2008].

Para destacar as implicações de privacidade de partilhar dados, ferramentas gráficas, incluindo mapas ou imagens, são usadas para visualizar os dados sendo liberados e aumentar a consciência dos participantes [Shilton et al. 2008]. Depois que os dados foram publicados, os participantes também podem monitorar o acesso aos dados, consultando arquivos de log da aplicação. Esses logs registram a natureza dos dados acessados, a frequência desses acessos, bem como a identidade das pessoas a acessá-los [Shilton et al. 2008, Shilton 2009, Mun et al. 2010]. Com base nos resultados dessas auditorias, os participantes controlam a distribuição de suas informações e podem julgar sua adequabilidade. Se necessário, eles podem atualizar suas políticas de controle de acesso para restringir ou ampliar as condições de autorização, a fim de corresponder às suas preferências de privacidade.

2.3.2. Confiabilidade dos Dados

Embora, num primeiro momento possa ser difícil estabelecer uma relação entre confiabilidade dos dados e segurança em MCS, é fácil afirmar que o simples fato de haver participação humana no processo já é o problema. Em outras palavras, os dados coletados e enviados pelos participantes nem sempre podem ser tratados ou considerados como confiáveis. Os motivos são:

1. **Contribuições erradas**: Infelizmente, aplicações MCS podem permitir que qualquer participante possa contribuir com dados, o que significa que as aplicações

podem receber dados errados e mal intencionados. Os participantes podem propositalmente registrar medições incorretas, colocando seus dispositivos em posições inadequadas. Por exemplo, gravações de áudio incorretas podem ser obtidas caso o dispositivo móvel do participante esteja indevidamente colocado em locais que atrapalham seu funcionamento, como no bolso durante uma tarefa de detecção de ruído.

2. **Conluio**: Nos atuais sistemas de reputação que usam MCS, cada participante pode publicar falsos relatórios para o servidor de aplicativo e fornecer informações arbitrárias sobre os valores durante o processo de votação de reputação. Além disso, atacantes podem conspirar para inutilizar aplicativos MCS.

3. **Contribuições mal-intencionadas**: Participantes mal-intencionados, para seus próprios benefícios, podem forjar informações para obter um maior ganho antes de finalizar tarefas que envolvem algum tipo de ganho monetário [Tuncay et al. 2012].

O problema da confiabilidade dos dados é geralmente discutido como diretamente relacionado a privacidade dos usuários [Pournajaf et al. 2014, Ganti et al. 2011] . Por exemplo, para alcançar a confiabilidade dos dados de localização dos participantes, algumas soluções baseiam-se em verificar a localização segura do dispositivos móveis em tempo real [Capkun et al. 2006] enquanto outras tentar estimar a confiabilidade dos dados coletados.

Contudo, pesquisas recentes envolvendo sistemas confiáveis (*trusted system*) aplicados a MCS vêm sendo realizadas. Em linhas gerais, estes sistemas se concentram em como avaliar a confiabilidade dos dados compartilhados e como manter a reputação de entidades da rede de processamento de dados. Huang et al. [Huang et al. 2010a] propuseram um sistema de reputação com base na função de Gompertz para calcular scores de pontuações de dispositivos para medir a confiabilidade dos dados coletados. No entanto, ele não leva em conta a preservação da privacidade.

Mais recentemente, vários esquemas de reputação que estão têm sido propostos [Dua et al. 2009, Christin et al. 2012, Li et al. 2014, Shin et al. 2011]. Algumas das abordagens [Dua et al. 2009, Christin et al. 2012] invocam a existência de uma terceira parte confiável, mas o estabelecimento e manutenção desta entidade em um ambiente distribuído não é trivial. No esquema de [Dua et al. 2009], vários pseudônimos são atribuídos para cada participante. Um servidor confiável é necessário para gerenciar o mapeamento entre a verdadeira identidade um participante e seus pseudônimos, e os valores de reputação entre os diferentes pseudônimos. Em comparação com outros métodos, este método não requer operações criptográficas caras e tem baixo custo de comunicação. Além disso, Dua et ai. [Christin et al. 2012] propôs e implementou um **Trusted Platform Module** (TPM), que é um microcontrolador embutido dentro de cada dispositivo móvel, para atestar a integridade de leituras dos sensores. No entanto, os chips TPM ainda não são amplamente adotado em dispositivos móveis.

Alguns métodos que não dependem da existência de uma terceira parte de confiança foram propostos. Em [Li et al. 2014], foi proposta uma solução de preservação da reputação de anonimato chamado IncogniSense. Ele gera pseudônimos periódicos usando

assinaturas cegas e disfarça valores exatos de reputação dinamicamente em grupos de reputação. Por exemplo, a solução depende de uma redundante número de participantes e incorre em despesas gerais de comunicação pesados. Shin et al. [Shin et al. 2011] propôs outra solução baseada em assinatura cega. Os autores consideraram o problema do ponto de vista do incentivo e tem como objetivo permitir aos participantes ganharem créditos através de contribuições. Assim, a necessidade de penalizar participantes maliciosos não precisa ser considerada.

Além disso, fazendo uso do fato de que múltiplas fontes de informação estão geralmente disponíveis em uma rede, Wang et al. [Wang et al. 2011b] propuseram avaliar a semelhança de múltiplas informações de diferentes fontes sobre o mesmo acontecimento e, em seguida, ajustar a pontuação de confiança de cada parte de informação. Com base na avaliação de confiança, as pontuações de confiança de nós também pode ser ajustada dinamicamente.

2.4. Implementações Seguras e de Segurança para MCS

Esta Seção apresenta algumas arquiteturas de aplicações MCS projetadas para garantir a segurança e privacidade dos usuários. Também apresenta ferramentas e bibliotecas voltados à segurança em sistemas MCS. Primeiramente serão discutidas as arquiteturas e posteriormente as bibliotecas. No fim, uma discussão sumariza os trabalhos apresentados.

2.4.1. Arquiteturas

Anonymous Authentication of Visitors for Mobile Crowd Sensing at Amusement Parks

Konidala et al. [Konidala et al. 2013] elaboraram um aplicativo que emprega pseudônimos certificados e um esquema de assinaturas parcialmente cegas para garantir a autenticidade e privacidade de usuários em parques de diversão. A primeira técnica utiliza criptografia para tornar anônima a presença do visitante no parque. Já a assinatura parcialmente cega (*Partially Blind Signature*) permite ao visitante ocultar a mensagem a ser assinada bem como explicitamente incorporar informações necessárias como, data da emissão, data da validade, a identidade do assinante.

Em linhas gerais, a solução funciona da seguinte forma: ao chegar no parque, o visitante solicita o aplicativo, instala e informa dados como idade, sexo, nacionalidade, altura (para recomendação de atrações), restrições alimentares, além de problemas de saúde e preferências de passeio. Para manter a privacidade do usuário, alguns dados são omitidos e não são solicitados, como nome, endereço, entre outros. Após essa coleta, o aplicativo envia as preferências e localização do usuário para o servidor de aplicativos, gerenciado pelo parque, o que permite ao servidor ser capaz de produzir, de forma dinâmica, um itinerário personalizado para cada visitante. Como incentivo para uso do aplicativo e eliminação dos tickets físicos, o visitante concorre a prêmios e participa de promoções.

A solução também permite ao parque gerir e implantar seus recursos de forma eficiente, melhorando o fluxo de acesso às atrações e analisando vários índices de desempenho (filas, idade, tempo de espera de cada atração, atrações mas requisitadas, entre

outras). Com a localização dos visitantes, o parque também pode inferir sobre o comportamento de grupos e visitantes individuais. Os autores acreditam que com a solução proposta pode alcançar um nível satisfatório de segurança, porém não asseguram que o visitante tenha sua privacidade totalmente preservada, tendo em vista que algumas informação são armazenadas e existem fatores alheios ao escopo do trabalho como, por exemplo, uso de WiFi dentro do parque.

No que diz respeito a segurança, os autores desenvolveram um protocolo para autenticação anônima dos visitantes (AAV). O protocolo utiliza pseudônimos para dissociar os dados dos visitantes de suas verdadeiras identidades e um procedimento de ofuscação, implementado através de um esquema de assinatura parcialmente cega (fase de emissão de pseudônimos certificados). Assim, nenhum pseudônimo de visitante é revelado para o servidor de aplicativo. Como o servidor de aplicativo não tem nenhum papel na geração dos pseudônimos, ele não pode vincular o pseudônimo de qualquer visitante com qualquer ticket ou cartão de crédito utilizado para a compra de bilhetes. Segundo os autores, o protocolo alcança uma autenticação anônima, através do qual o servidor de aplicativo aceita dados somente de pseudônimos que foram certificados.

Para lidar com ataques de falsa localização, o protocolo usa a abordagem de [He et al. 2011], onde o servidor formula e poem em prática certas heurísticas, tais como calcular o tempo decorrido entre o local anterior dos visitantes e a localização atual. Se esse tempo coincide com o tempo médio gasto por outros visitantes entre os mesmos dois locais, os dados são considerados legítimos. Sempre que o servidor de aplicativo identifica que os dados de um determinado pseudônimo não corresponde a estas heurísticas e os valores limite, o servidor pode revogar o pseudônimo certificado e negar todas as comunicações futuras. O protocolo também usa HTTPS, por padrão, no canal de comunicação entre o aplicativo e o servidor.

SPPEAR: Security Privacy-preserving Architecture for Participatory-sensing Applications

Com o foco direcionado em preservar a privacidade dos participantes e permitir a concessão de incentivos aos participantes, Gisdakis et al. [Gisdakis et al. 2014] propuseram um arquitetura para Sensoriamento Participativo que faz uso de pseudônimos. Denominada SPPEAR, a arquitetura é abrangente e segura, e capaz de: (i) ser escalável, confiável e aplicável a qualquer tipo de aplicação de sensoriamento participativo e MCS; (ii) garantir a não identificação dos usuários, oferecendo forte proteção a privacidade; (iii) limitar a participação de usuários legítimos de forma totalmente responsável; (iv) evitar de forma eficiente que as identidades dos usuários sejam reveladas; (v) ser resiliente as entidades participantes; e (vi) poder suportar vários mecanismos de incentivo de forma a preservar a privacidade.

A arquitetura SPPEAR é formada por:

- **Usuários**: Atuam tanto como produtores de informação (ou seja, enviando dados) quanto consumidores de informação (ou seja, solicitando informações do sistema). Os dispositivos de usuários com capacidades de sensoriamento participam das tare-

fas submetendo amostras autenticadas ou por meio de consulta aos dados coletados.

- **Serviço de Tarefas - TS**: Esta entidade inicia tarefas e campanhas de sensoriamento. Também, define e fornece as recompensas que os participantes receberão por suas contribuições.

- **Gerente de Grupo - GM**: É responsável por registrar os dispositivos do usuário e emitir as credenciais anônimas para eles. Além disso, o GM autoriza a participação de dispositivos em várias tarefas de forma indiferente, usando tokens de autorização.

- **Provedor de identidade - IdP**: Oferece serviços de gerenciamento e identificação de credenciais (por exemplo, autenticação de usuário e controle de acesso, entre outros) para o sistema.

- **Certificação pseudônimo Authority - PCA**: Fornece credenciais anônimos efêmeras, denominadas pseudônimos, para os dispositivos. Um pseudônimo é um certificado X.509, que se liga a uma identidade anônima com uma chave pública e que possui uma validade, medida em tempo. PCA gera a quantidade desejada de pares de chaves e gera o mesmo número de assinaturas de pedidos de certificado.

- **Serviços de Agregação de Amostras - SAS**: Os dispositivos de usuário mandam amostras para esta entidade que é responsável por armazenar e processar os dados coletados. Para cada amostra submetida autêntico, o SAS emite um recibo para o dispositivo, que depois o envia para reivindicar créditos para a tarefa de detecção. O SAS possui interfaces que permitem a qualquer usuário autenticado e autorizado consultar os resultados das tarefas de sensoriamento e campanhas.

- **Autoridade de Resolução - RA**: É a entidade responsável pela revogação do anonimato dos dispositivos (por exemplo, dispositivos que perturbam o sistema ou poluem o processo de recolha de dados).

A Figura 2.8 apresenta a arquitetura SPPEAR.

Uma vez que separa processos e funções através de entidades, de acordo com o princípio de separação de direitos: "a cada entidade é dado o mínimo de informações necessárias para executar a tarefa desejada", SPPEAR atinge os objetivos de segurança e confiabilidade para qualquer aplicação.

Participatory Privacy: Enabling Privacy in Participatory Sensing

Em [De Cristofaro and Soriente 2013], Cristofaro e Soriente propuseram uma arquitetura de reforço a privacidade para aplicações MCS. A arquitetura proposta, denominada PEPSI (Figura 2.9) faz uso de um provedor de serviços de aplicações que gerencia as campanhas MCS. Os participantes (chamados de nós móveis) enviam relatórios dos dados de sensoriamento para o provedor de serviços, que após processar os dados, encaminha os relatórios aos usuários finais. PEPSI envolve uma quarta entidade, designada

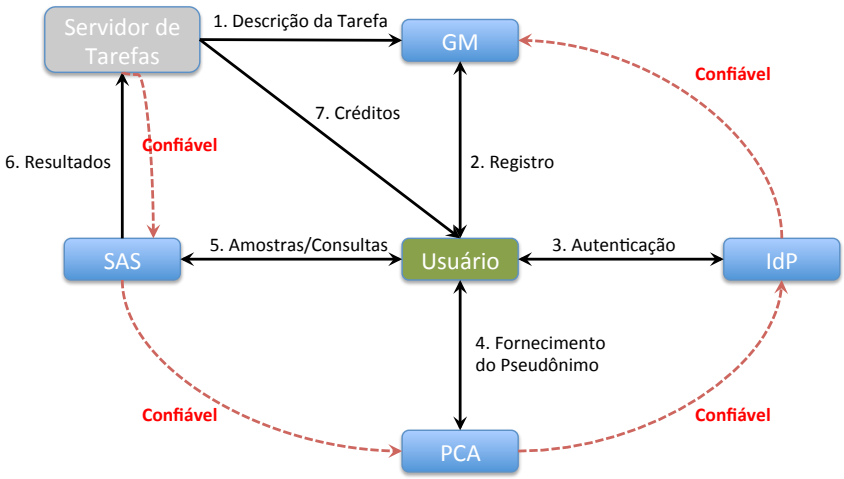


Figura 2.8. Visão geral da arquitetura SPPEAR

como autoridade de registro. Ela é responsável pela definição inicial dos parâmetros do sistema e pelo registro dos nós móveis e usuários finais.

Uma vez que um dos principais objetivos da PEPSI é ocultar relatórios de dados e consultas de pessoal não autorizado, todos os dados, relatórios e consultas são criptografadas. PEPSI usa Identidade baseada em Criptografia (*Identity-base Encryption* - IBE). Cada relatório é identificado por um conjunto de rótulos (etiquetas), que são usados ??como identidades. Assim, nós móveis podem derivar uma chave de criptografia pública única a partir desses rótulos e usá-la para criptografar os relatórios a serem transmitidos ao provedor de serviço. Durante o registro, nós móveis registram esses rótulos para a autoridade de registro, que, em seguida, atua como o gerador de chave privada do sistema IBE. Ele gera a chave de decifração privada correspondente a esses rótulos e passa a chave privada para os usuários finais interessante sobre seu registro.

Outra característica de PEPSI é que o provedor de serviços não apenas retransmiti todos os relatórios a todos os usuários finais. Isto geraria uma carga de processamento pesada para os usuários finais, porque eles precisariam tentar todas as suas chaves privadas para descriptografar um relatório. Ao invés disso, o provedor de serviços impõe um mecanismo de marcação através do qual ele pode combinar, de forma eficiente, os relatórios com as consultas. O mecanismo exige que os nós móveis etiquetem cada relatório com um token criptográfico que identifica o tipo de relatório apenas para usuários finais autorizados, sem vazamento de nenhuma informação do relatório. A marca é calculada a partir das mesmas etiquetas utilizadas para derivar a chave pública. Ao mesmo tempo, devido às propriedades de mapeamento bilinear inerente a IBE, usuários finais podem computar a mesma marca para o mesmo relatório usando suas chaves de criptografia privadas e fornecer a etiqueta para o provedor de serviços quando fizer uma subscrição de consulta. Em seguida, o provedor de serviços apenas encaminha um relatório marcado aos usuários finais que forneceram a mesma marca.

Towards a Practical Deployment of Privacy-preserving Crowd-sensing Tasks

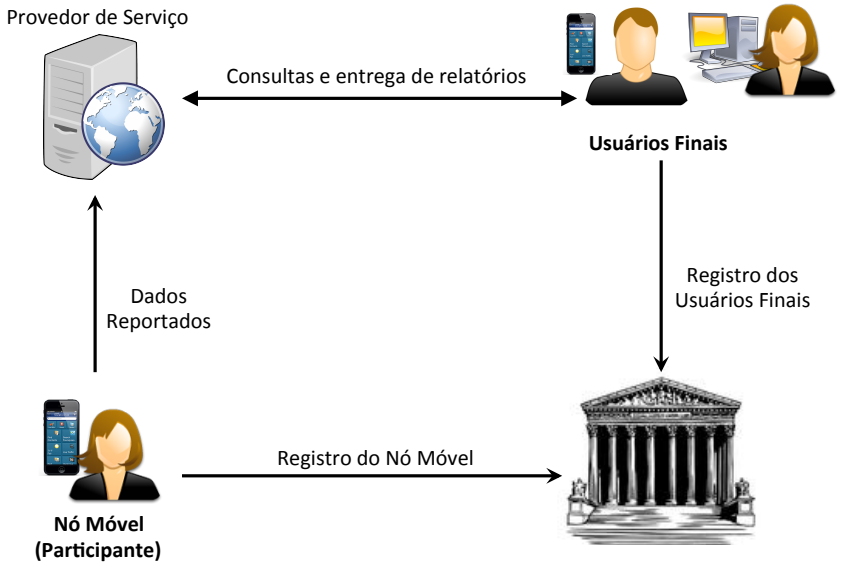


Figura 2.9. Arquitetura PEPSI. Fonte: [De Cristofaro and Soriente 2013]

Haderer et al. [Haderer et al. 2014] apresentaram uma nova plataforma de *Crowdsourcing* capaz de preservar a privacidade através da união de dois middlewares conhecidos: APISENSE e PRIVAPI.

APISENSE é uma plataforma de *Crowdsourcing* móvel que facilita a implantação de experiências MCS por cuidar dos desafios críticos neste domínio [Haderer et al. 2013]. Ela fornece uma plataforma de *Software-as-a-Service*, onde experimentos são descritos como scripts que serão enviados para os dispositivos móveis, a fim de recolher dados. A arquitetura distribuída do APISENSE é formada pelo: (i) serviço Hive, responsável por gerenciar a comunidade de usuários móveis e publicar as tarefas de sensoriamento; (ii) Honeypot, utilizado para descrever as tarefas como scripts (com base em uma extensão de JavaScript) e enviá-los para (iii) os dispositivos móveis, que recebem e executam as tarefas. A plataforma APISENSE apoia a implementação de diferentes estratégias de incentivo, incluindo *feedback* do usuário, *ranking* do usuário e recompensas. A seleção de estratégias de incentivo depende da natureza das experiências *Crowdsourcing*. No que diz respeito a privacidade, uma camada no dispositivo móvel implementa vários algoritmos para filtrar e "perturbar" informações sensíveis (por exemplo, catálogo de endereços, localização) dependendo das preferências do usuário. O usuário mantém o controle do seu dispositivo móvel para seleccionar os sensores, bem como quando e onde estes sensores pode ser utilizados pela plataforma.

PRIVAPI é um middleware de preservação da privacidade que pode ser facilmente integrado no topo da APISENSE. Seu objetivo é pré-processar os dados recolhidos de mobilidade antes de ser liberado. Graças ao seu conhecimento sobre o conjunto de dados inteiro, pode-se usar uma estratégia de anonimização ideal nos dados de mobilidade enquanto ainda oferece um nível satisfatório de utilidade.

A Figura 2.10 apresenta a arquitetura proposta.

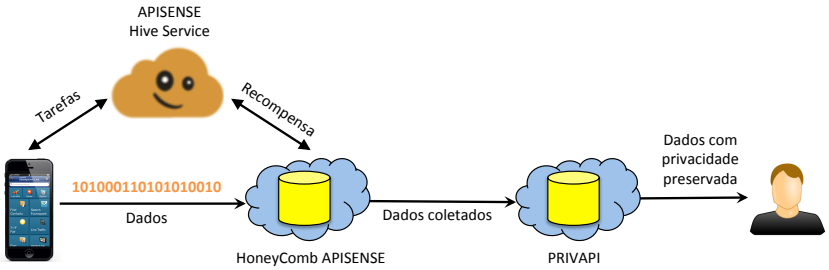


Figura 2.10. Visão geral da arquitetura proposta

2.4.2. Bibliotecas

Caché: Caching Location-Enhanced Content to Improve User Privacy

Amini et al. [Amini et al. 2011] propuseram a criação de uma biblioteca capaz que atender a uma quantidade significativa de aplicações que realizam a busca de dados de localização dos usuários. Denominada Caché, o objetivo da biblioteca, além de dar suporte o outros aplicativos, é oferecer aos usuários um nível de privacidade aceitável, tendo em vista a crescente preocupação com a exposição de informações sensíveis. Arquitetada para trabalhar como um repositório de localização, aplicada a privacidade, Caché atua realizando uma pré-busca das informações solicitadas antes que o usuário necessite. Como a busca é feita localmente, o usuário evita que sua localização seja exposta no momento que realmente esteja utilizado-as. Assim, ao invés de compartilhar a localização atual em cada pedido de informação, o usuário só precisa compartilhar o período de tempo.

A biblioteca funciona da seguinte maneira (Figura 2.11). Em primeiro lugar, existe a necessidade do aplicativo se adequar ao Caché. Por exemplo, o desenvolvedor do aplicativo deve fornecer algumas informações relevantes como a forma de fazer o download do conteúdo, URL de acesso e intervalo de atualização do conteúdo. Depois, o usuário instala o aplicativo habilitando o Caché e seleciona as regiões de seu interesse. Em seguida, o Caché realiza o download e atualiza o conteúdo com base na taxa de atualização que o desenvolvedor especificou. Como o Caché trabalha com bloco de informações relativamente grandes, é esperado o momento mais oportuno para baixar o conteúdo e atualiza-lo (por exemplo, quando o dispositivo móvel estiver conectado a uma conexão WiFi). A última etapa acontece quando a aplicação exige um conteúdo, que é recuperado do Caché ao invés de fazer uma consulta externa. Desta forma, o conteúdo a ser utilizado offline (sem necessidade de conexão de dados). Segundo o autor, sua arquitetura é semelhante a um proxy Internet (não transparente).

Embora não tenha sido criado especificamente para MCS, Caché, através do uso da pré-busca, permite o uso de conteúdo com capacidade de localização, garantindo a privacidade do usuário, sem contar com os benefícios relacionados ao download de dados. Caché também incentiva os desenvolvedores de aplicativos a elaborarem ideias para melhorar a privacidade do usuário como, por exemplo, trabalhar com esquemas melhores de amostragem para que os usuários não necessitem acessar conteúdo online e sem a necessidade de de infraestrutura de terceiros para garantir a privacidade dos usuários.

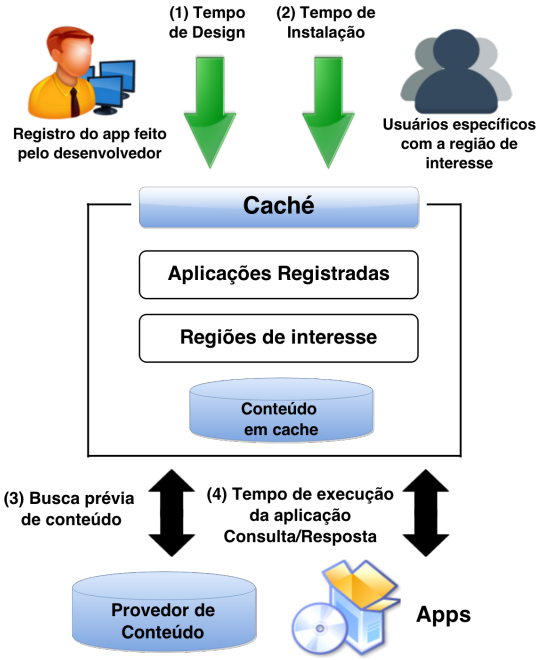


Figura 2.11. Funcionamento do Caché. Uma vez que o desenvolvedor tenha registrado a aplicação (1) e o usuário ter especificado as regiões para quais conteúdos devem ser armazenados em cache (2), o Caché faz o pedido e armazena o conteúdo (3) para uso futuro pela aplicação (4). Fonte: [Amini et al. 2011].

2.5. Pesquisas em Aberto

Neste Capítulo já foram discutidas as ameaças à segurança em MCS, bem como foram apresentados algumas soluções, ferramentas e bibliotecas seguras ou que fornecem segurança para MCS. O leitor deve ter percebido que as soluções são normalmente sob medida (para atender uma determinada situação) e que práticas para lidar com a privacidade são escassas. Assim, ainda existe uma ampla gama de desafios de pesquisa sem solução. Esta Seção destaca algumas pontos de pesquisa ainda em aberto.

2.5.1. Como inserir os participantes na questão da privacidade

Um dos principais desafios para novas gerações de sistemas e aplicações MCS é a inclusão dos participantes na questão da privacidade. Para tanto, os seguintes aspectos precisam ser estudados:

- **Privacidade sob medida**: Nas atuais soluções, a noção de privacidade é altamente individual e depende do ponto de vista e da opinião do usuário. Assim, é fundamental para criar consciência para as ameaças à privacidade ao usuário, e ajudá-lo, tornando a configuração complexa de aplicações de sensoriamento participativos de fácil compreensão. No entanto, a maioria das contramedidas de preservação da privacidade discutidos não apresentam uma interface de usuário que pode sensibilizar e facilitar a compreensão dos mecanismos complexos em uso. Interfaces de usuário personalizadas, que consideram as características originais de dispositivos móveis, são, portanto, muito procurados. Além disso, a existência de tais interfaces podem incentivar a aceitação e, mais tarde, a adoção dos mecanismos de preserva-

ção da privacidade por parte dos participantes, como eles serão capazes de entender melhor (através das interfaces) os meandros dos mecanismos.

- **Facilidade de uso**: A usabilidade das aplicações e suas configurações de privacidade precisam ser levadas em consideração. Toda vez que um participante é obrigado a realizar uma extensa e manual configuração de suas preferências, frequentemente sua paciência termina e ela acaba ou por deixar as configurações no modo padrão ou por marca quaisquer opções sem entender as implicações de suas escolhas. O estudo de Gross e Acquisti [Gross and Acquisti 2005] demonstra essa situação no caso das configurações de privacidade em redes sociais online.

- **A transparência dos níveis de protecção da privacidade**: Para avaliar se a protecção da privacidade oferecida é adequada, os usuários precisam ser capazes de comparar o nível oferecido de proteção contra seus requisitos de proteção individual. Embora a maioria das soluções pesquisadas baseiam suas avaliações em métricas matemáticos e verificáveis, a percepção do usuário e seu nível de satisfação com as soluções existentes não tem sido explicitamente considerado.

- **Incorporação de *feedback* do usuário**: Além de fornecer interfaces de usuário para configurar os níveis de privacidade, insights sobre como a proteção é percebida e para quais extensões os usuários estão envolvidos na configuração de suas configurações de privacidade são obrigados a apresentar a usabilidade.

Em linhas gerais, os usuários ainda são considerados no processo de avaliação da usabilidade e utilidade das soluções de privacidade. Os estudos existentes na área apenas correlacionam preocupações com a privacidade com as modalidades de sensoriamento usadas (Klasnja et al., 2009), ou analisam a forma como os participantes entendem, selecionam e se comportam com os métodos, por exemplo, de ocultação da localização (Brush et al. , 2010).

2.5.2. Adaptabilidade das Soluções de Privacidade

Toda vez que uma nova modalidades de sensoriamento é incorporada as plataformas de dispositivos móveis, surgem novas famílias de aplicações e surgem também novos desafios a privacidade. A capacidade de lidar com essa ampla gama de cenários é extremamente necessária. Assim, as futuras soluções de segurança e privacidade em MCS precisam ser combináveis e adaptáveis.

Entre as questões estão:

- **Aplicações independentes X soluções sob medida**: Algumas das soluções apresentadas neste Capítulo são ou foram adaptadas para cenários de específicos. Por exemplo, a ocultação de locais sensíveis, através da criação de registros falsos de localização para evitar correlações entre usuários e locais só foi avaliada no cenário aplicação PEIR [Mun et al. 2009]. Outros cenários precisam ser investigados para determinar os limites potenciais e os inconvenientes das soluções propostas em função das especificidades de aplicação. Esta investigação irá destacar as mudanças necessárias para proceder a partir de soluções de privacidade adaptados aos conceitos de privacidade de aplicação agnóstica.

- **Abordagem sistêmica**: Não existe, ou pelo menos os autores deste Capítulo não encontraram, uma arquitetura de privacidade flexível que aborde o problema do ponto de vista do sistema. Existem várias contramedidas onde os aspectos de privacidade são abordados.

- **Cenários evolutivos de sensoriamento**: Em cenários nos quais as características dos dados dos sensores são conhecidas antecipadamente, soluções de privacidade podem ser adaptadas em conformidade, por exemplo, pela adição de ruído com propriedades correspondentes. No entanto, em caso de sensoriamento de cenários dinâmicos e/ou imprevisíveis, em que as características dos dados do sensor não podem ser determinadas com antecedência, novos conceitos de privacidade precisam ser inventados.

2.5.3. *Trade-offs* entre Privacidade, Desempenho e Fidelidade dos Dados

Mecanismos robustos para proteção da privacidade (remoção ou ofuscação de leituras do sensor, por exemplo) podem influenciar a fidelidade de dados, o atraso no sensoriamento ou a integridade dos dados. No entanto, proteger a integridade dos dados do sensor neutraliza mecanismos de preservação da privacidade. Consequentemente, existe um *trade-off* entre as garantias de privacidade e fidelidade.

Já está claro que a participação do usuário precisa ser incentivada através da garantia da sua privacidade. Por outro lado, sistemas vulneráveis ou defeituosos podem contribuir para que dados corrompidos ou errados sejam utilizados pelas aplicações. Percebe-se então que existe um ***trade-off* entre anonimato e qualidade/integridade dos dados**. Para evitar que dados degradem a precisão dos resultados das aplicações, os dispositivos ou os dados em questão precisam ser identificados e eliminados a partir do conjunto de dispositivos encarregados do sensoriamento. Assim, a investigação sobre sistemas de reputação que servem tanto para o anonimato quanto para as exigências e especificidades dos cenários de sensoriamento é necessária.

Outro ponto que precisa de análise é a **proteção da privacidade de outras pessoas**. O trabalho de Tang et ai. (2010) demonstra que os participantes valorizam a privacidade da localização de seus amigos, mas a maioria dos mecanismos de preservação da privacidade atuais se concentram na proteção apenas próprios participantes (DietSense [] prova que os rostos de pessoas não envolvidas podem aparecer nas imagens da aplicação). O fato que é que os sistemas atuais, a função de proteger a privacidade dos outros é do usuário participante. Soluções automatizadas para minimizar os dados capturados de forma que ele não viole a privacidade dos outros é de grande interesse.

Por fim, embora a proteção de dados sensíveis seja altamente valorizado, em certas situações, como em cenários de emergência, podem ser necessários meios para **substituir ou sobrescrever as configurações de privacidade** especificadas pelos participantes. Esta questão pode ser comparada a encontrada no cenário de saúde, onde os médicos podem ser capazes de substituir o controle de acesso de sensores corporais para obter acesso a dados críticos de saúde.

2.5.4. Como medir privacidade?

Diferentes métodos, critérios ou métricas estão sendo usados para avaliar o desempenho das soluções propostas em termos de protecção da privacidade. Embora possa ser difícil ou mesmo impossível chegar a métricas universais para quantificar a privacidade, a necessidade de definir métricas generalizadas é amplamente reconhecida. Capturando o nível de proteção de privacidade, independentemente do cenário de aplicação particular, pode ser visto como uma meta de pesquisa de longo prazo, mas a definição dessas métricas é obrigatória para alcançar uma base comum para a comparação de mecanismos. Mas como obtê-las?

As métricas de privacidade empregadas atualmente precisam ser pesquisadas para determinar quais parâmetros de entrada (por exemplo, a quantidade de participantes na mesma região) são considerados necessários para calcular o grau de privacidade e qual é a natureza dos parâmetros de saída (por exemplo, a distância euclidiana entre os dados reais e os ocultados/perturbados), considerando os cenários de aplicação. Adicionalmente, as métricas de privacidade de outros domínios de aplicação devem ser analisadas em relação à sua aplicabilidade em MCS.

Além disso, analisando certas soluções, percebe-se a existência de uma entidade central para proteger a privacidade e anonimato do participante. No entanto, tais soluções não demonstram garantias ou provas de que o grau prometido de privacidade é respeitado, uma vez que detalhes de implementação dificilmente estão disponíveis e até mesmo a abordagem empregada para proteger a privacidade é normalmente desconhecida. A investigação sobre viabilidade dos mecanismos de privacidade ainda permanece como um campo de pesquisa em aberto.

2.5.5. Normas para Investigar Privacidade

Devido sua natureza sensível, conjuntos de dados públicos do mundo real para aplicações MCS são escassos. Por isso, a investigação da privacidade ocorre geralmente com conjuntos de dados privados ou sintéticos. Como resultado, a base de dados não é bem aceita para a avaliação de novos mecanismos, o que dificulta sua aferição em relação à outros.

Para superar essa limitação, a comunidade de pesquisa deve fornecer para conjuntos de dados abertos que podem servir como uma base para avaliações de desempenho e segurança. Isso inclui conjuntos de dados do mundo real, bem como conjuntos de dados sintéticos representativos para várias modalidades de sensoriamentos diferentes.

Além disso, como as implementações de mecanismos de privacidade estão quase sempre indisponíveis para o público em geral, torna-se difícil ou mesmo impossível referência-las contra mecanismos propostos. Tornar a descrição técnica e detalhada da implementação ou sua própria execução disponível para a comunidade de pesquisa permite validar os resultados e as soluções individuais de referência.

2.6. Considerações Finais

Este Capítulo apresentou o paradigma de *Mobile Crowd Sensing* (MCS) e suas aplicações para o dia a dia. A Seção 2.2 tratou da explicar MCS. Primeiro, a evolução das ideais sobre sobre sensoriamento foi contextualizada. Em seguida, definições sobre MCS foram

feitas e os componentes e o ciclo de vida explicados. Depois, as características únicas de MCS foram enumeradas. Por fim, algumas aplicações MCS foram apresentadas.

A Seção 2.3 tratou os aspectos de segurança em MCS. Os dois principais problemas de segurança (privacidade do usuário e confiabilidade dos dados) foram bem discutidos. Os problemas e também as técnicas para resolvê-los foram apresentadas. Perto do fim, as soluções existentes (Seção 2.4) baseadas em segurança para MCS foram discutidas. A Seção 2.5 apontou várias questões em aberto

2.6.1. Observações Finais

A longo prazo, MCS é capaz de estimular a pesquisa em uma série de domínios.

Em primeiro lugar, o mundo hoje consiste de espaços físicos e virtuais, onde qualquer objeto sensoriado está entre esses espaços. Assim, é importante para explorar abordagens para agregação e fusão dos dados complementares para o melhor entendimento.

Em segundo lugar, ainda é preciso estudar a fusão da inteligência humana e da máquina em todo o ciclo de vida de MCS, desde o sensoriamento, transmissão de dados e processamento de dados.

Em terceiro lugar, alguns dos fatores éticos, como a inventividade e privacidade do usuário, devem ser os blocos de construção fundamentais de arquiteturas MCS.

Finalmente, o sucesso de MCS baseia-se na utilização de conhecimentos multidisciplinares, incluindo ciências sociais, ciência cognitiva, economia, ciência de computação, e assim por diante. Todas essas áreas devem ser considerada no desenho de técnicas e sistemas MCS.

Referências

[Acampora et al. 2013] Acampora, G., Cook, D., Rashidi, P., and Vasilakos, A. (2013). A survey on ambient intelligence in healthcare. *Proceedings of the IEEE*, 101(12):2470–2494.

[Amini et al. 2011] Amini, S., Lindqvist, J., Hong, J., Lin, J., Toch, E., and Sadeh, N. (2011). Caché: caching location-enhanced content to improve user privacy. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 197–210. ACM.

[Burke et al. 2006] Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., and Srivastava, M. B. (2006). Participatory sensing. In *In: Workshop on World-Sensor-Web (WSW'06): Mobile Device Centric Sensor Networks and Applications*, pages 117–134.

[Capkun et al. 2006] Capkun, S., Cagalj, M., and Srivastava, M. (2006). Secure localization with hidden and mobile base stations. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–10.

[Christin et al. 2011] Christin, D., Reinhardt, A., Kanhere, S. S., and Hollick, M. (2011). A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, 84(11):1928 – 1946. Mobile Applications: Status and Trends.

[Christin et al. 2012] Christin, D., Rosskopf, C., Hollick, M., Martucci, L., and Kanhere, S. (2012). Incognisense: An anonymity-preserving reputation framework for participatory sensing applications. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 135–143.

[Clementi et al. 2013] Clementi, A., Pasquale, F., and Silvestri, R. (2013). Opportunistic manets: Mobility can make up for low transmission power. *IEEE/ACM Trans. Netw.*, 21(2):610–620.

[Conti and Kumar 2010] Conti, M. and Kumar, M. (2010). Opportunities in opportunistic computing. *Computer*, 43(1):42–50.

[Das et al. 2010] Das, T., Mohan, P., Padmanabhan, V. N., Ramjee, R., and Sharma, A. (2010). Prism: Platform for remote sensing using smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 63–76, New York, NY, USA. ACM.

[De Cristofaro and Soriente 2013] De Cristofaro, E. and Soriente, C. (2013). Participatory privacy: Enabling privacy in participatory sensing. *Network, IEEE*, 27(1):32–36.

[Deng and Cox 2009] Deng, L. and Cox, L. P. (2009). Livecompare: Grocery bargain hunting through participatory sensing. In *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*, HotMobile '09, pages 4:1–4:6, New York, NY, USA. ACM.

[Derawi et al. 2010] Derawi, M. O., Nickel, C., Bours, P., and Busch, C. (2010). Unobtrusive user-authentication on mobile phones using biometric gait recognition. In *Proceedings of the 2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, IIH-MSP '10, pages 306–311, Washington, DC, USA. IEEE Computer Society.

[Dimov 2014] Dimov, D. (2014). Crowdsensing: State of the art and privacy aspects. `http://resources.infosecinstitute.com/crowdsensing-state-art-privacy-aspects/`.

[Domingo-Ferrer and Mateo-Sanz 2002] Domingo-Ferrer, J. and Mateo-Sanz, J. M. (2002). Practical data-oriented microaggregation for statistical disclosure control. *IEEE Trans. on Knowl. and Data Eng.*, 14(1):189–201.

[Dong et al. 2008] Dong, Y. F., Kanhere, S., Chou, C. T., and Bulusu, N. (2008). Automatic collection of fuel prices from a network of mobile cameras. In *Proceedings of the 4th IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS '08, pages 140–156, Berlin, Heidelberg. Springer-Verlag.

[Dua et al. 2009] Dua, A., Bulusu, N., Feng, W.-C., and Hu, W. (2009). Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX Conference on Hot Topics in Security*, HotSec'09, pages 8–8, Berkeley, CA, USA. USENIX Association.

[Dutta et al. 2009] Dutta, P., Aoki, P. M., Kumar, N., Mainwaring, A., Myers, C., Willett, W., and Woodruff, A. (2009). Common sense: Participatory urban sensing using a network of handheld air quality monitors. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 349–350, New York, NY, USA. ACM.

[Eisenman et al. 2007] Eisenman, S. B., Miluzzo, E., Lane, N. D., Peterson, R. A., Ahn, G.-S., and Campbell, A. T. (2007). The bikenet mobile sensing system for cyclist experience mapping. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 87–101, New York, NY, USA. ACM.

[Ganti et al. 2011] Ganti, R., Ye, F., and Lei, H. (2011). Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39.

[Gao and Cao 2011] Gao, W. and Cao, G. (2011). User-centric data dissemination in disruption tolerant networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 3119–3127.

[Gaonkar et al. 2008] Gaonkar, S., Li, J., Choudhury, R. R., Cox, L., and Schmidt, A. (2008). Micro-blog: Sharing and querying content through mobile phones and social participation. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pages 174–186, New York, NY, USA. ACM.

[Gisdakis et al. 2014] Gisdakis, S., Giannetsos, T., and Papadimitratos, P. (2014). Sppear: Security & privacy-preserving architecture for participatory-sensing applications. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, pages 39–50, New York, NY, USA. ACM.

[Google 2015] Google (2015). Waze. `http://waze.com`.

[Grosky et al. 2007] Grosky, W., Kansal, A., Nath, S., Liu, J., and Zhao, F. (2007). Senseweb: An infrastructure for shared sensing. *MultiMedia, IEEE*, 14(4):8–13.

[Gross and Acquisti 2005] Gross, R. and Acquisti, A. (2005). Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 71–80, New York, NY, USA. ACM.

[Guo et al. 2015] Guo, B., Wang, Z., Yu, Z., Wang, Y., Yen, N. Y., Huang, R., and Zhou, X. (2015). Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Comput. Surv.*, 48(1):7:1–7:31.

[Guo et al. 2013] Guo, B., Zhang, D., Wang, Z., Yu, Z., and Zhou, X. (2013). Opportunistic iot: Exploring the harmonious interaction between human and the internet of things. *J. Netw. Comput. Appl.*, 36(6):1531–1539.

[Haderer et al. 2014] Haderer, N., Primault, V., Raveneau, P., Ribeiro, C., Rouvoy, R., and Ben Mokhtar, S. (2014). Towards a Practical Deployment of Privacy-preserving Crowd-sensing Tasks. In *Middleware Posters and Demos '14*, Bordeaux, France.

[Haderer et al. 2013] Haderer, N., Rouvoy, R., and Seinturier, L. (2013). Dynamic deployment of sensing experiments in the wild using smartphones. In Dowling, J. and TaÃ¯ani, F., editors, *Distributed Applications and Interoperable Systems*, volume 7891 of *Lecture Notes in Computer Science*, pages 43–56. Springer Berlin Heidelberg.

[He et al. 2011] He, W., Liu, X., and Ren, M. (2011). Location cheating: A security challenge to location-based social network services. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 740–749.

[Howe 2006] Howe, J. (2006). Crowdsourcing: A definition. *Crowdsourcing: Tracking the rise of the amateur*.

[Hu et al. 2013] Hu, X., Liu, Q., Zhu, C., Leung, V. C. M., Chu, T. H. S., and Chan, H. C. B. (2013). A mobile crowdsensing system enhanced by cloud-based social networking services. In *Proceedings of the First International Workshop on Middleware for Cloud-enabled Sensing*, MCS '13, pages 3:1–3:6, New York, NY, USA. ACM.

[Huang et al. 2010a] Huang, K. L., Kanhere, S. S., and Hu, W. (2010a). Are you contributing trustworthy data?: The case for a reputation system in participatory sensing. In *Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, MSWIM '10, pages 14–22, New York, NY, USA. ACM.

[Huang et al. 2010b] Huang, K. L., Kanhere, S. S., and Hu, W. (2010b). Preserving privacy in participatory sensing systems. *Computer Communications*, 33(11):1266 – 1280.

[IBM 2010] IBM (2010). Creekwatch: Explore your watershed. `http://creekwatch.researchlabs.ibm.com`.

[Kapadia et al. 2009] Kapadia, A., Kotz, D., and Triandopoulos, N. (2009). Opportunistic sensing: Security challenges for the new paradigm. In *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, pages 1–10.

[Karamshuk et al. 2011] Karamshuk, D., Boldrini, C., Conti, M., and Passarella, A. (2011). Human mobility models for opportunistic networks. *Communications Magazine, IEEE*, 49(12):157–165.

[Kong et al. 2015] Kong, L., He, L., Liu, X.-Y., Gu, Y., Wu, M.-Y., and Liu, X. (2015). Privacy-preserving compressive sensing for crowdsensing based trajectory recovery. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 31–40.

[Konidala et al. 2013] Konidala, D., Deng, R., Li, Y., Lau, H., and Fienberg, S. (2013). Anonymous authentication of visitors for mobile crowd sensing at amusement parks. In Deng, R. and Feng, T., editors, *Information Security Practice and Experience*, volume 7863 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg.

[Krumm 2007] Krumm, J. (2007). Inference attacks on location tracks. In *Proceedings of the 5th International Conference on Pervasive Computing*, PERVASIVE'07, pages 127–143, Berlin, Heidelberg. Springer-Verlag.

[Lane et al. 2010] Lane, N. D., Miluzzo, E., Lu, H., Peebles, D., Choudhury, T., and Campbell, A. T. (2010). A survey of mobile phone sensing. *Comm. Mag.*, 48(9):140–150.

[Leonardi et al. 2014] Leonardi, C., Cappellotto, A., Caraviello, M., Lepri, B., and Antonelli, F. (2014). Secondnose: An air quality mobile crowdsensing system. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, NordiCHI '14, pages 1051–1054, New York, NY, USA. ACM.

[Levy and da Costa 1993] Levy, P. and da Costa, C. I. (1993). *tecnologias da inteligência, As*. Editora 34.

[Li et al. 2014] Li, Q., Cao, G., and La Porta, T. (2014). Efficient and privacy-aware data aggregation in mobile sensing. *Dependable and Secure Computing, IEEE Transactions on*, 11(2):115–129.

[Liu 2007] Liu, L. (2007). From data privacy to location privacy: Models and algorithms. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1429–1430. VLDB Endowment.

[Lu et al. 2010] Lu, H., Lane, N. D., Eisenman, S. B., and Campbell, A. T. (2010). Fast track article: Bubble-sensing: Binding sensing tasks to the physical world. *Pervasive Mob. Comput.*, 6(1):58–71.

[Ludwig et al. 2015] Ludwig, T., Reuter, C., Siebigteroth, T., and Pipek, V. (2015). Crowdmonitor: Mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4083–4092, New York, NY, USA. ACM.

[Ma et al. 2014] Ma, H., Zhao, D., and Yuan, P. (2014). Opportunities in mobile crowd sensing. *Communications Magazine, IEEE*, 52(8):29–35.

[Machanavajjhala et al. 2007] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1).

[Maisonneuve et al. 2009] Maisonneuve, N., Stevens, M., Niessen, M., and Steels, L. (2009). Noisetube: Measuring and mapping noise pollution with mobile phones. In Athanasiadis, I. N., Rizzoli, A. E., Mitkas, P. A., and Gomez, J. M., editors, *Information Technologies in Environmental Engineering*, Environmental Science and Engineering, pages 215–228. Springer Berlin Heidelberg.

[Maisonneuve et al. 2010] Maisonneuve, N., Stevens, M., and Ochab, B. (2010). Participatory noise pollution monitoring using mobile phones. *Info. Pol.*, 15(1,2):51–71.

[Mathur et al. 2010] Mathur, S., Jin, T., Kasturirangan, N., Chandrasekaran, J., Xue, W., Gruteser, M., and Trappe, W. (2010). Parknet: Drive-by sensing of road-side parking statistics. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 123–136, New York, NY, USA. ACM.

[Mediated Spaces, Inc 2015] Mediated Spaces, Inc (2015). The wildlab: Use mobile technology to explore, discovery, and share the natural world. `http://thewildlab.org`.

[Miluzzo et al. 2008] Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S. B., Zheng, X., and Campbell, A. T. (2008). Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 337–350, New York, NY, USA. ACM.

[Minkman et al. 2015] Minkman, E., van Overloop, P., and van der Sanden, M. (2015). Citizen science in water quality monitoring: Mobile crowd sensing for water management in the netherlands. In *World Environmental and Water Resources Congress 2015*, pages 1399–1408.

[Minson et al. 2015] Minson, S. E., Brooks, B. A., Glennie, C. L., Murray, J. R., Langbein, J. O., Owen, S. E., Heaton, T. H., Iannucci, R. A., and Hauser, D. L. (2015). Crowdsourced earthquake early warning. *Science Advances*, 1(3):e1500036+.

[Mohan et al. 2008] Mohan, P., Padmanabhan, V., and Ramjee, R. (2008). Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. In *ACM Sensys*. Association for Computing Machinery, Inc. Raleigh, NC, USA.

[Mun et al. 2010] Mun, M., Hao, S., Mishra, N., Shilton, K., Burke, J., Estrin, D., Hansen, M., and Govindan, R. (2010). Personal data vaults: A locus of control for personal data streams. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 17:1–17:12, New York, NY, USA. ACM.

[Mun et al. 2009] Mun, M., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R., and Boda, P. (2009). Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, MobiSys '09, pages 55–68, New York, NY, USA. ACM.

[Pan et al. 2013] Pan, B., Zheng, Y., Wilkie, D., and Shahabi, C. (2013). Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, pages 344–353, New York, NY, USA. ACM.

[Pournajaf et al. 2014] Pournajaf, L., Xiong, L., Garcia-Ulloa, D. A., and Sunderam, V. (2014). A survey on privacy in mobile crowd sensing task management. Technical report, Technical Report TR-2014-002, Department of Mathematics and Computer Science, Emory University.

[Rachuri et al. 2011] Rachuri, K. K., Mascolo, C., Musolesi, M., and Rentfrow, P. J. (2011). Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, MobiCom '11, pages 73–84, New York, NY, USA. ACM.

[Rana et al. 2010] Rana, R. K., Chou, C. T., Kanhere, S. S., Bulusu, N., and Hu, W. (2010). Ear-phone: An end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 105–116, New York, NY, USA. ACM.

[Reddy et al. 2007] Reddy, S., Parker, A., Hyman, J., Burke, J., Estrin, D., and Hansen, M. (2007). Image browsing, processing, and clustering for participatory sensing: Lessons from a dietsense prototype. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 13–17, New York, NY, USA. ACM.

[Reddy et al. 2009] Reddy, S., Samanta, V., Burke, J., Estrin, D., Hansen, M., and Srivastava, M. (2009). Mobisense: mobile network services for coordinated participatory sensing. In *Autonomous Decentralized Systems, 2009. ISADS '09. International Symposium on*, pages 1–6.

[Sherchan et al. 2012] Sherchan, W., Jayaraman, P., Krishnaswamy, S., Zaslavsky, A., Loke, S., and Sinha, A. (2012). Using on-the-move mining for mobile crowdsensing. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*, pages 115–124.

[Shi et al. 2010] Shi, J., Zhang, R., Liu, Y., and Zhang, Y. (2010). Prisense: Privacy-preserving data aggregation in people-centric urban sensing systems. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 758–766, Piscataway, NJ, USA. IEEE Press.

[Shilton 2009] Shilton, K. (2009). Four billion little brothers?: Privacy, mobile phones, and ubiquitous data collection. *Commun. ACM*, 52(11):48–53.

[Shilton et al. 2008] Shilton, K., Burke, J., Estrin, D., Hansen, M., and Srivastava, M. (2008). Participatory privacy in urban sensing. In *Proceedings of the International Workshop on Mobile Devices and Urban Sensing*, MODUS, pages 1–7.

[Shin et al. 2011] Shin, M., Cornelius, C., Peebles, D., Kapadia, A., Kotz, D., and Triandopoulos, N. (2011). Anonysense: A system for anonymous opportunistic sensing. *Pervasive and Mobile Computing*, 7(1):16 – 30.

[Surowiecki 2005] Surowiecki, J. (2005). *The Wisdom of Crowds*. Anchor.

[Sweeney 2002] Sweeney, L. (2002). K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570.

[Thepvilojanapong et al. 2010] Thepvilojanapong, N., Ono, T., and Tobe, Y. (2010). A deployment of fine-grained sensor network and empirical analysis of urban temperature. *Sensors*, 10(3):2217.

[Tuncay et al. 2012] Tuncay, G. S., Benincasa, G., and Helmy, A. (2012). Autonomous and distributed recruitment and data collection framework for opportunistic sensing. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 407–410, New York, NY, USA. ACM.

[Vieira and Alves 2014] Vieira, A. P. and Alves, J. C. R. (2014). Direito à privacidade na sociedade da informação. *Revista Jus Navigandi*, (3979).

[Wang et al. 2011a] Wang, H., Uddin, M., Qi, G.-J., Huang, T., Abdelzaher, T., and Cao, G. (2011a). Photonet: A similarity-aware image delivery service for situation awareness. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pages 135–136.

[Wang et al. 2013] Wang, L., Zhang, D., and Xiong, H. (2013). effsense: Energy-efficient and cost-effective data uploading in mobile crowdsensing. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 1075–1086, New York, NY, USA. ACM.

[Wang et al. 2011b] Wang, X., Govindan, K., and Mohapatra, P. (2011b). Collusion-resilient quality of information evaluation based on information provenance. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*, pages 395–403.

[Weppner and Lukowicz 2013] Weppner, J. and Lukowicz, P. (2013). Bluetooth based collaborative crowd density estimation with mobile phones. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 193–200.

[Yan et al. 2009] Yan, T., Marzilli, M., Holmes, R., Ganesan, D., and Corner, M. (2009). mcrowd: A platform for mobile crowdsourcing. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 347–348, New York, NY, USA. ACM.

[Zhang et al. 2014a] Zhang, D., Wang, L., Xiong, H., and Guo, B. (2014a). 4w1h in mobile crowd sensing. *Communications Magazine, IEEE*, 52(8):42–48.

[Zhang et al. 2014b] Zhang, X., Yang, Z., Wu, C., Sun, W., Liu, Y., and Liu, K. (2014b). Robust trajectory estimation for crowdsourcing-based mobile applications. *Parallel and Distributed Systems, IEEE Transactions on*, 25(7):1876–1885.

Capítulo 3

Implementação Eficiente e Segura de Algoritmos Criptográficos

Armando Faz Hernández, Roberto Cabral, Diego F. Aranha, Julio López

Abstract

Software implementation of a cryptographic algorithm is not an easy task even for advanced programmers, because it requires a careful knowledge not only about algorithms but also of the target architecture. In this tutorial, we will describe some techniques to produce an efficient and secure software implementation. For the sake of efficiency, we will detail how advanced vector instruction sets accelerate the execution of the following cryptographic algorithms: the AES encryption algorithm, the SHA-3 cryptographic hash function and the key agreement protocol based on the elliptic curve Curve25519. Focusing on the secure software development, we will illustrate some implementations that are vulnerable against side-channel attacks; also we will present some countermeasures that mitigate such attacks thereby preventing leakage of secret information.

Resumo

A implementação segura de um algoritmo criptográfico não é uma tarefa trivial nem mesmo para os programadores mais experientes, pois requer um conhecimento cuidadoso não apenas do próprio algoritmo, mas também da arquitetura alvo. Neste minicurso, vamos nos concentrar em descrever os aspectos que ajudam a tornar uma implementação de criptografia em software eficiente e segura. Do lado da eficiência, detalharemos como os conjuntos avançados de instruções aceleram a execução dos algoritmos criptográficos a seguir: o algoritmo de encriptação AES, a função de resumo SHA-3 e o protocolo de acordo de chaves baseado na curva elíptica Curve25519. Pensando no desenvolvimento seguro, mostraremos como algumas implementações são vulneráveis contra ataques de canais laterais, adicionalmente apresentaremos técnicas que mitigam ditos ataques evitando assim o vazamento de informação secreta.

3.1. Introdução à criptografia

Nesta seção, serão apresentados um conjunto de conceitos básicos sobre criptografia que serão usados ao longo deste capítulo. Espera-se que leitores familiarizados com o tema possam relembrar os conceitos básicos e propriedades dos algoritmos criptográficos, e leitores que estão tendo contato com o tema pela primeira consigam uma base teórica mínima necessária para entender o capítulo.

3.1.1. Conceitos básicos

O modelo de comunicação mais simples consiste em duas entidades, A e B, querendo se comunicar. Neste modelo a entidade A (emissor) quer enviar uma mensagem para a entidade B (receptor) através de um meio de comunicação; esse meio pode ser acessado por uma terceira entidade E (também chamada de intruso ou adversário). Visando facilitar a descrição de protocolos de comunicação, denotamos as entidades A, B e E por *Alice*, *Bob* e *Eva*, respectivamente.

Um dos problemas presente neste modelo de comunicação é assegurar que uma mensagem enviada por Alice seja lida unicamente por Bob, que a mensagem recebida por Bob tenha sido gerada por Alice e que a mensagem não tenha sofrido qualquer alteração no caminho.

A criptografia moderna pode ser vista como um conjunto de técnicas matemáticas que permitem estabelecer uma comunicação segura na presença de adversários maliciosos. Ela engloba mecanismos para assegurar a integridade de dados, técnicas para troca de chave secreta, protocolos para autenticar usuários, votação eletrônica, moeda eletrônica, entre outros.

Uma comunicação segura deve possuir os seguintes serviços segurança:

- **Sigilo.** A informação é mantida em segredo de todos, exceto das partes autorizadas.
- **Autenticação.** Deve ser possível garantir a autenticidade do remetente de uma mensagem; um atacante não deve ser capaz de se passar por outra pessoa.
- **Integridade.** O receptor deve conseguir verificar se a mensagem recebida não foi modificada durante a transmissão; um atacante não deve ser capaz de substituir uma mensagem falsa pela legítima.
- **Irretratabilidade.** Garante que uma entidade não pode negar ter participado de uma comunicação.

De acordo com o tipo de chave usado, os métodos criptográficos podem ser subdivididos em duas grandes categorias: criptografia simétrica e criptografia assimétrica:

- **Criptografia Simétrica**. Consiste nas funções de encriptação que usam a mesma chave para encriptar e decriptar. Esses algoritmos requerem que o

emissor e o receptor estabeleçam uma chave para realizar a comunicação encriptada. Esta parte da criptografia é também conhecida como criptografia de chave secreta no sentido que a chave só pode ser conhecida pelos envolvidos na comunicação, pois toda a segurança do esquema baseia-se na chave criptográfica, uma vez que qualquer pessoa de posse da chave consegue encriptar ou decriptar mensagens.

- **Criptografia Assimétrica**. Também conhecida como criptografia de chave pública, considera o uso de duas chaves distintas, uma pública e uma privada, para cada participante da comunicação; sendo que a chave pública pode ser divulgada e a chave privada deve ser mantida em segredo. Assim, uma mensagem que é encriptada com a chave pública, será decriptada somente com o uso da correspondente chave privada. A geração do par de chaves deve garantir que a chave privada não possa ser obtida a partir da chave pública. Na criptografia assimétrica as operações de encriptação podem ser realizadas por qualquer entidade, contudo as operações de decriptação são restritas à entidade que possui a chave privada. Dentre os usos da criptografia assimétrica encontram-se a encriptação de dados, a geração de assinaturas digitais, o estabelecimento seguro de chaves, entre outros.

Os protocolos criptográficos podem ser construídos a partir de primitivas criptográficas. Um protocolo criptográfico é composto por uma série de computações e comunicações entre duas ou mais entidades com o fim de realizar uma tarefa especifica. Por exemplo: transmitir dados de forma segura, verificar a integridade de uma mensagem ou estabelecer chaves privadas mediante um canal inseguro.

3.1.2. Encriptação de dados

A encriptação de dados tem como objetivo prover sigilos das mensagens quando se usa um meio de comunicação inseguro. Portanto, o processo de *encriptação* (as vezes também denominado como cifração) consiste em converter um texto claro em um texto encriptado usando uma chave secreta. O processo inverso, conhecido por *decriptação*, transforma um texto encriptado em um texto claro usando a mesma chave usada na encriptação.

Um dos princípios fundamentas da criptografa foi postulado em 1883 por Kerckhoff e diz que a segurança do processo de encriptação não deve estar baseada apenas em manter em segredo o funcionamento interno do algoritmo. Na criptografia moderna, os algoritmos devem ser conhecidos por toda a comunidade e a segurança dos mesmos deve estar atrelada a uma *chave criptográfica*.

Em termos matemáticos o texto claro é representado por M e o texto encriptado por C. Portanto, a função de encriptação E usa uma chave criptográfica k e opera sobre M para produzir $C = E_k(M)$. No processo reverso, tem-se a função D que usa a mesma chave k que foi usada para encriptar e opera sobre C para produzir $M = D_k(C)$. A encriptação e decriptação são operações inversas, por conta disso, a identidade $M = D_k(E_k(M))$ deve ser verdadeira para quaisquer valores de M e k.

Se uma função de encriptação possui o mesmo tamanho de entrada e saída,

esta função também é conhecida como *cifrador de bloco*; alguns exemplos de cifradores de bloco são: o *Digital Encryption Standard* (DES) e o *Advanced Encryption Standard* (AES) com blocos de tamanho 64 e 128 bits, respectivamente. Para encriptar mensagens de tamanho arbitrário é possível usar *modos de operação*, os quais usam os cifradores de bloco como primitiva básica. Esses modos serão examinados com mais detalhe na Seção 3.4.1.2.

Um dos ataques que pode ser aplicado à grande maioria dos cifradores é o ataque de força bruta, que consiste em achar a chave usada na encriptação, dado um texto claro e um texto encriptado, testando cada uma das chaves possíveis. A complexidade desse ataque é 2^n, onde n é o tamanho da chave. Um cifrador possui um nível de segurança de n bits se o melhor algoritmo capaz de encontrar a chave criptográfica possui uma complexidade de $O(2^n)$.

3.1.3. Funções de resumo criptográfico

As funções de resumo criptográfico agem como uma função de compressão que mapeia uma cadeia de bits de tamanho arbitrário em uma cadeia de bits de tamanho fixo. Elas são muito usadas na criptografia moderna e seu uso é essencial em várias aplicações de segurança, tais como: esquemas de assinatura digital, verificação da integridade dos dados, geração de números pseudo aleatórios, geração de chaves, dentre outras.

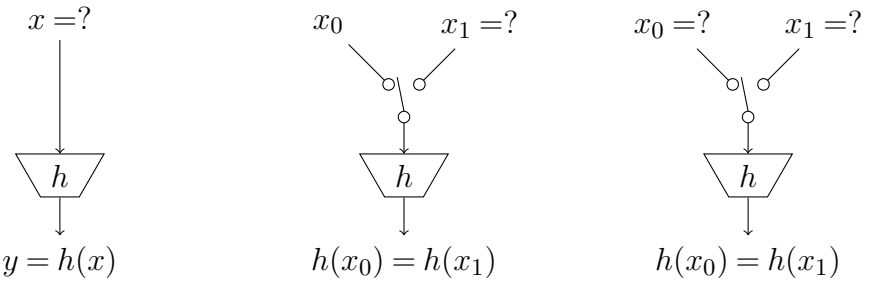
A cadeia de bits gerada por uma função de resumo é chamada de valor de resumo ou *digest*; e teoricamente, deve identificar à mensagem de forma única. Uma função de resumo $h : M \rightarrow R$ mapeia uma cadeia de bits $x \in M$ de tamanho finito e arbitrário em uma cadeia de bits $y \in R$ de tamanho fixo n. Tendo, $y = h(x)$.

Pode-se notar que mais de uma mensagem poderá ser mapeada em um mesmo valor de resumo, visto que o domínio M de h é maior que sua imagem R. Porém, algumas aplicações, como as assinaturas digitais, por exemplo, requerem que seja computacionalmente inviável encontrar duas mensagens diferentes que gerem o mesmo valor de resumo; outras aplicações apenas necessitam que seja inviável encontrar uma mensagem dado o valor de resumo.

As primeiras definições, analises e construções de funções de resumo criptográficos podem ser encontradas nos trabalhos de Rabin [Rab78], Yuval [Yuv78] e Merkle [Mer79]. Rabin propôs um modelo baseado no algoritmo de encriptação DES; Yuval usou o paradoxo de aniversário para mostrar como encontrar colisões para uma função de resumo de n bits com $2^{n/2}$ operações; e Merkle introduziu que esse tipo de função deveria ser resistente à colisões, primeira pré-imagem e segunda pré-imagem.

As funções de resumo criptográfico, diferentemente dos algoritmos de encriptação, não possuem chaves secretas. Assim, para termos uma função de resumo segura é necessário que satisfaça as seguintes propriedades de segurança, ilustradas na Figura 3.1:

- **Resistência à pré-imagem**: Dada uma função $h : M \rightarrow R$ e um valor de resumo $y \in R$, é computacionalmente inviável encontrar $x \in M$ tal que $h(x) = y$.



(a) Pré-imagem. (b) Segunda pré-imagem. (c) Colisões.

Figura 3.1: As três propriedades de segurança de uma função de resumo.

- **Resistência à segunda pré-imagem**: Dada uma função $h : M \to R$ e uma mensagem $x_0 \in M$, é computacionalmente inviável encontrar uma mensagem $x_1 \in M$ tal que $x_0 \neq x_1$ e $h(x_0) = h(x_1)$.
- **Resistência à colisão**: Dada uma função $h : M \to R$, é computacionalmente inviável encontrar duas mensagens $x_0, x_1 \in M$ tal que $x_0 \neq x_1$ e $h(x_0) = h(x_1)$.

Em 1993, a função de resumo chamada *Secure Hash Algorithm* (SHA) foi desenvolvida pelo *National Institute of Standards and Technology* (NIST) e publicada como padrão nacional americano para o processamento seguro de informações [Nat93]. Posteriormente esta versão foi revisada e ficou conhecida por SHA-1 [Nat95]. Já em 2002, o NIST definiu três novas funções de resumo, conhecidas por SHA-2, com tamanhos de valor de resumo de 256, 384 e 512 bits [Nat02], e em 2008 mais uma função de resumo com tamanho de saída de 224 bits [Nat08a]. Em 2015, uma terceira versão do padrão, chamada SHA-3, foi publicada [Nat08b, Nat15].

3.1.4. Protocolo de acordo de chaves

Suponha que Alice e Bob querem compartilhar uma chave secreta para ser usada em uma função de encriptação simétrica, sendo que o único meio de comunicação é inseguro, ou seja qualquer informação transmitida pode ser observada por Eva. Esse problema é comumente conhecido como acordo de chaves e foi engenhosamente resolvido por Diffie e Hellman em 1976 [DH76].

Eles propuseram uma solução para esse problema observando que a computação de algumas funções possuem uma certa assimetria; isto é, dado uma entrada x é fácil calcular $y = f(x)$, entretanto dado y é computacionalmente inviável obter x tal que $x = f^{-1}(y)$. Por exemplo, obter o produto de dois números primos grandes é simples, mas é difícil recuperar os números primos que compõem o produto. Diffie e Hellman observaram que este fenômeno poderia ser usado para estabelecer um protocolo de troca de chaves seguro. O *protocolo Diffie-Hellman* está descrito a seguir:

1. Alice e Bob negociam os seguintes parâmetros de domínio: um número primo p grande e um gerador g do grupo cíclico $\mathbb{Z}_p^*$.

2. Alice escolhe aleatoriamente um valor $x \in \mathbb{Z}_p$, computa $k_A = g^x$ e envia o resultado para Bob.

3. Bob atua de forma similar, escolhendo aleatoriamente um valor $y \in \mathbb{Z}_p$, computando $k_B = g^y$ e enviando o resultado para a Alice.

4. Bob recebe o valor k_A da Alice e computa k_A^y. De forma análoga, Alice computa k_B^x com o valor recebido pelo Bob.

5. Neste ponto, o seguinte valor $k_B^x = k_A^y = g^{xy}$ é o segredo compartilhado que pode ser usado para gerar uma chave privada.

A intuição atrás deste protocolo consiste na existência de uma função eficiente para calcular a exponenciação de elementos em $\mathbb{Z}_p^*$; no entanto, a computação de sua função inversa, conhecida como o logaritmo discreto, é computacionalmente inviável. Assim, qualquer atacante que interceptar as mensagens transmitidas deve calcular g^{xy} a partir de g, g^x e g^y.

Originalmente, Diffie e Hellman propuseram o uso do grupo multiplicativo dos inteiros modulo p para a implementação do protocolo. Embora, existam outros grupos onde a operação de exponenciação é mais eficiente; por exemplo, o grupo formado pelo conjunto de pontos de uma curva elíptica. Se a curva elíptica possui certas propriedades que tornam o logaritmo discreto inviável, então esse grupo é uma opção viável para a implementação do protocolo de acordo de chaves, o qual é comumente chamado de protocolo *Diffie-Hellman baseado em curvas elípticas* (ECDH).

3.1.5. Curvas elípticas

Uma curva elíptica E está definida sobre um corpo finito primo $\mathbb{F}_p$ pela seguinte equação:

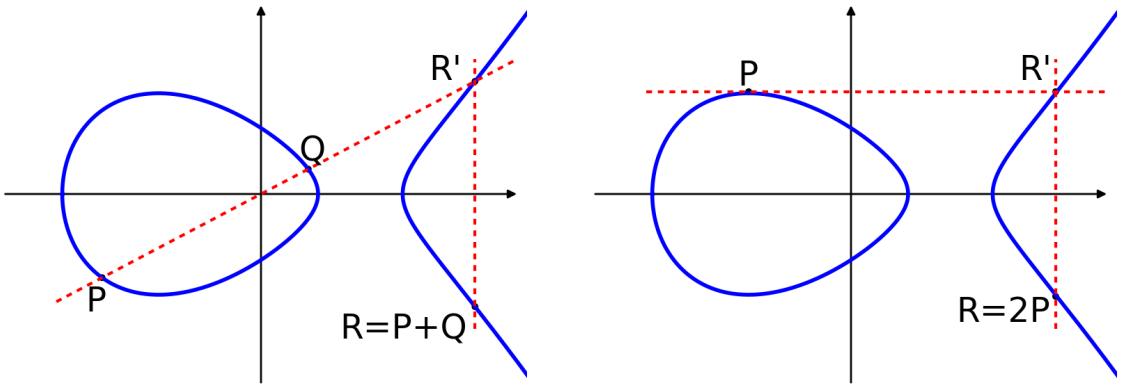
$$E\colon y^2 = x^3 + Ax + B \tag{1}$$

tal que $A, B \in \mathbb{F}_p$ e $4A^3 + 27B^2 \neq 0$. Os pontos pertencentes à curva elíptica E, junto como o ponto no infinito, formam um grupo cíclico:

$$E(\mathbb{F}_p) = \{(x,y)\colon x,y \in \mathbb{F}_p \text{ tal que } y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}, \tag{2}$$

onde $\mathcal{O}$ é o elemento identidade do grupo. O número de elementos deste conjunto está delimitado pelo intervalo de Hasse: $\#E(\mathbb{F}_p) = p + 1 \pm 2\sqrt{p}$. Usualmente a operação de grupo é expresso de forma aditiva; assim a lei do grupo é a soma de dois pontos $P, Q \in E(\mathbb{F}_p)$ denotado como $P + Q$.

Define-se geometricamente a soma de dois pontos da seguinte forma: trace uma linha reta que passe por P e Q, esta linha irá intersectar E em um ponto R'; agora trace uma linha vertical sobre o ponto R'; a linha intersectará à curva E no ponto R. Então, o ponto R representará a soma $P + Q$. O caso especial da soma de um ponto consigo mesmo é conhecido como duplicação de pontos. Geometricamente o cálculo de $2P = P + P$ segue o mesmo conceito descrito; embora ao invés de traçar uma linha que passe por dois pontos, deve-se traçar a linha tangente à curva E no ponto P. Essa linha irá intersectar à curva em um ponto R'; do qual se trace uma



(a) Soma de dois pontos diferentes P e Q.

(b) Duplicação de um ponto P.

Figura 3.2: Representação geométrica da lei do grupo em uma curva elíptica definida sobre $\mathbb{R}$.

linha vertical que cruza à curva num ponto R; este último representará a computação de $2P$; na Figura 3.2 é mostrada de forma gráfica a lei do grupo numa curva elíptica definida sobre $\mathbb{R}$.

A soma repetida de um ponto P, denominada por multiplicação de um ponto P por um inteiro k, está definida como:

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ vezes}}.$$

Um dos métodos mais eficientes para computar a multiplicação de pontos é o algoritmo *duplicação-e-soma*, chamado assim porque utiliza as funções de duplicação e soma, antes mencionadas. O algoritmo inicializa um ponto $Q = \mathcal{O}$ que serve como acumulador. Uma vez feito isso, o processamento lê os bits de k iterativamente, do mais ao menos significativo, de modo que a cada iteração i o ponto Q é duplicado e se o i-ésimo bit de k é igual a 1, o ponto P é somado ao acumulador. Uma vez que todos os bits foram lidos, o ponto Q conterá o ponto kP.

A obtenção de múltiplos de um ponto é uma operação que pode ser computada eficientemente, no entanto acredita-se que a operação inversa para certas curvas elípticas é computacionalmente inviável. Dado um ponto G que seja gerador do grupo elíptico $E(\mathbb{F}_p)$ e um ponto qualquer da curva $Q \in E(\mathbb{F}_p)$ determinar um $k \in \mathbb{Z}$ tal que $Q = kG$ é conhecido como o *Problema do Logaritmo Discreto em Curvas Elípticas* (ECDLP). Os algoritmos mais eficientes que resolvem ECDLP possuem uma complexidade de $O(\sqrt{\#E(\mathbb{F}_p)})$. Por exemplo, se $p \approx 2^{256}$ então a complexidade de resolvê-lo requer aproximadamente 2^{128} operações, tornando o calculo de k inviável.

3.2. Instruções avançadas dos processadores

Nesta seção serão descritos alguns conjuntos avançados de instruções que apareceram nos processadores atuais; por exemplo, os conjuntos de instruções AES-NI,

SSE, AVX e o mais recente, AVX2. O leitor será familiarizado a terminologia básica das instruções avançadas dos processadores e será apresentada a evolução das instruções vetoriais nas micro-arquiteturas; em particular a ênfase será dada à micro-arquitetura Haswell. Finalmente, será discutido um exemplo de como detectar a presença dos conjuntos avançados de instruções nos processadores.

3.2.1. Instruções vetoriais

No fim da década de 1990, os fabricantes de processadores focaram seus esforços em explorar o paralelismo de dados ao invés do paralelismo de instruções como era feito nas arquiteturas RISC. Para isso, eles incorporaram unidades funcionais que são capazes de executar uma única instrução sobre um conjunto de dados; esse processamento encaixa-se no paradigma conhecido como *Single Instruction Multiple Data* (SIMD), introduzido em [Fly72].

Um dos primeiros conjuntos de instruções a implementar o paradigma SIMD foi lançado em 1997, conhecido como *Multimedia eXtensions* (MMX) [Cor]. O MMX adicionou registradores de 64 bits e instruções vetoriais que habilitavam o processamento de duas operações de 32 bits, nessa época as arquiteturas possuíam registradores nativos de 32 bits. Essas instruções definem a semântica do conteúdo do registrador, isto é, um registrador de 64 bits poderia ser operado como um vetor de oito palavras de 8 bits, um vetor de quatro palavras de 16 bits, um vetor de duas palavras de 32 bits ou como um vetor de 64 bits. No começo as instruções MMX foram voltadas para auxiliar o processamento de aplicações gráficas e multimídia.

Nos últimos 18 anos, tanto a Intel como a AMD lançaram novos conjuntos de instruções vetoriais para explorar ainda mais o paralelismo a nível de dados. A seguir serão apresentados alguns dos conjuntos mais relevantes lançados por essas duas companhias:

- Em 1999, a Intel lançou *Streaming SIMD Extensions* (SSE) que adicionou oito registradores de 128 bits (denotados `XMM0-XMM7`) e incluiu instruções para dar suporte à computação de aritmética de ponto flutuante.
- No ano 2000, a AMD desenvolveu a arquitetura de 64 bits, com isso o tamanho dos registradores nativos aumentou de 32 bits para 64 bits e o número de registradores vetoriais foi duplicado (denotados `XMM0-XMM15`).
- O conjunto de instruções SSE foi evoluindo com o lançamento dos novos conjuntos SSE2, SSE3, SSSE3 e SSE4. O SSE2 foi lançado para dar suporte às operações de aritmética inteira. Os outros conjuntos, entretanto, foram incorporando outros tipos de instruções vetoriais; desse modo, começou a surgir instruções para manipulação de cadeias de caracteres, permutação de palavras dentro dos registradores e códigos de correção de erros.
- A diversidade das instruções aumentou ainda mais em 2010, quando foi incluído o *Advanced Encryption Standard New Instructions* (AES-NI) que acelera a computação do algoritmo de encriptação AES. Este conjunto será amplamente explicado na Seção 3.4.1.

- Já em 2011, o conjunto *Advanced Vector Extensions* (AVX) fez contribuições relevantes na arquitetura. Pois incluiu registradores de 256 bits, chamados `YMM`, que encontram-se sobrepostos sobre os registradores `XMM`. Além disso, AVX introduziu um novo formato de codificação que permite utilizar código de montagem de três operandos, o que torna a atribuição de registradores mais flexível.

- O mais recente conjunto de instruções é a segunda versão do AVX, chamada AVX2. Esse conjunto será detalhado na seção seguinte.

Como foi apresentado, as primeiras instruções vetoriais foram direcionadas ao processamento gráfico, esse cenário vem mudando nos últimos anos e agora está disponível uma vasta diversidade de instruções. Neste documento, serão apresentadas técnicas para se tirar proveito destes conjuntos na implementação eficiente de algoritmos criptográficos.

3.2.2. O conjunto de instruções AVX2

A microarquitetura Haswell foi lançada no início de 2013 e trouxe consigo uma série de inovações para acelerar a execução dos programas. Dentre as quais destacam-se: a inclusão de mais duas portas de execução, um novo multiplicador inteiro, as instruções de manipulação de bits, entre outras. No entanto, a característica mais relevante do Haswell é o suporte ao conjunto de instruções vetoriais AVX2.

O AVX2 contém novas instruções que expandem a computação de aritmética inteira nos registradores de 256 bits; pois o conjunto AVX continha instruções apenas para a aritmética de ponto flutuante. Além disso, AVX2 conta com instruções de permutação e combinação, que permitem movimentar as palavras contidas nos registradores vetoriais, entre outras características. A seguir, são destacadas algumas instruções que fazem parte do conjunto AVX2 e que são relevantes para o contexto de implementação eficiente de algoritmo criptográficos:

1. Acesso a memória.

   (a) `LOAD/STORE`. Essas funções carregam/armazenam um conjunto de dados de/para um endereço de memória de/para um registrador vetorial. Vale a pena mencionar que o endereço de memória deve estar alinhado para 32 bits, ou seja, o valor de endereço deve ser um múltiplo de 32, caso contrário o desempenho da aplicação é afetado.

   (b) `BRCAST`. Essa instrução replica um valor de 64 bits de um registrador nativo (ou endereço de memória) em um registrador vetorial.

2. Aritmética inteira.

   (a) `ADD/SUB`. Essas funções calculam a adição e subtração de números inteiros armazenados nas palavras de um registrador vetorial. Por exemplo, a instrução pode computar oito operações de 32 bits ou quatro operações de 64 bits.

(b) `MUL`. Usando essa instrução é possível calcular quatro multiplicações de palavras de 32 bits; os quatro produtos de 64 bits gerados são armazenados em um registrador de 256 bits.

3. Funções lógicas.

   (a) `XOR/AND/OR/ANDNOT`. Essas funções calculam operações lógicas de 256 bits.

   (b) `SHL/SHR`. Para cada palavra armazenada em um registrador vetorial, essas instruções deslocam para esquerda ou direta, respectivamente, uma quantidade fixa de bits.

   (c) `SHLV/SHRV`. Essas instruções potencializam o processamento paralelo das instruções de deslocamento. Porque ao invés de fazer um deslocamento fixo, a instrução recebe, além do registrador alvo, um segundo registrador contendo a quantidade de bits a serem deslocados; desta forma, cada palavra do vetor alvo será deslocada de acordo com às quantidades especificadas no segundo registrador.

4. Permutações internas no registrador.

   (a) `PERM`. Instruções para embaralhar a posição das palavras dentro do registrador são chamadas de permutações. No entanto, nas primeiras versões do SSE e AVX, foram também conhecidos como *shuffle* ou *shuffling*. Durante o texto o termo permutação será usado para se referir a esse tipo de instruções.

5. Combinação de registradores.

   (a) `UPCK`. Essas instruções preenchem um registrador intercalando as palavras da parte alta/baixa de outros dois registradores de origem.

   (b) `BLEND`. Esse tipo de instrução preenche um registrador a partir de palavras contidas em dois registradores diferentes; a escolha está baseada no valor de uma máscara binária.

   (c) `PRBLEND`. Essa instrução combina partes de 128 bits de dois registradores de 256 bits em um novo registrador de 256 bits; a escolha está baseada no valor de uma máscara binária.

   (d) `ALIGNR`. Essa instrução concatena dois registradores de 128 bits e desloca n bytes, os 128 bits menos significativos do valor intermediário gerado são armazenado no registrador destino.

6. Conversão de elementos.

   (a) `CAST`. São pseudo-instruções que mudam a semântica do conteúdo dos registradores. A maioria dessas instruções são tratadas diretamente pelos compiladores e não geram nenhuma instrução em código de montagem.

Instrução vetorial	Latência	Vazão	Porta de execução							
			0	1	2	3	4	5	6	7
LOAD	3	2			×	×				
STORE	3	1			×	×	×			×
BRCAST	5	2			×	×				
ADD/SUB	1	2		×				×		
MUL	5	1	×							
XOR	1	3	×	×				×		
SHL/SHR	1	1						×		
SHLV SHRV	2	0.5	×					×		
PERM	3	1						×		
BLEND	1	3	×	×				×		
PRBLEND	3	1						×		
ALIGNR	1	1						×		
UPCK	1	1						×		
CAST	0	-								
EXTRACT	3	1						×		
INSERT	3	1						×		

Tabela 3.1: Latência, vazão (instruções executadas por ciclo quando não há dependências) e portas de execução de algumas instruções AVX2.

(b) `EXTRACT`. Essa instrução retorna um registrador de 128 bits composto pela parte alta (ou baixa) de um registrador de 256 bits. Diferente das instruções de `CAST`, esta é uma instrução explicita que é codificada em linguagem de montagem.

(c) `INSERT`. Essa instrução tem a funcionalidade inversa à instrução `EXTRACT`, pois ela permite adicionar o conteúdo de um registrador de 128 bits na parte alta (ou baixa) de um registrador de 256 bits.

Essa lista não inclui todas as instruções pertencentes ao conjunto de instruções AVX2; é recomendado consultar [Cor11] para informação adicional sobre o conjunto AVX2. Na Tabela 3.1 são mostradas as latências, a vazão e as portas de execução das instruções AVX2 acima citadas; essas informações foram extraídas do relatório técnico produzido por Agner Fog [Fog14].

3.2.3. Detecção dos conjuntos de instruções

Antes de usar um conjunto avançado de instruções é preciso verificar se tal conjunto é suportado pelo processador alvo. Existem várias formas de fazer essa verificação, caso o sistema operacional usado seja o Linux a maneira mais fácil é rodar o seguinte comando no terminal:

```
1 $ cat /proc/cpuinfo | grep flags
```

```
flags:  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
        pge mca cmov pat pse36 clflush dts acpi mmx fxsr
        sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp
        pni pclmulqdq dtes64 monitor ds_cpl vmx smx est
        tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2
        x2apic movbe popcnt tsc_deadline_timer aes xsave
        fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
        invpcid rtm avx
```

Esse comando mostra as características do processador. Dentre as saídas temos o campo *flags*, onde é apresentado os conjuntos de instruções suportados. O processador onde esse comando foi executado é apresentado no Apêndice A e, como se pode perceber, o mesmo suporta todos os conjuntos de instruções aqui apresentados.

Outra forma de verificar se os conjuntos de instruções são suportados, independente de sistema operacional, é usando a instrução `CPUID` para verificar se o bit que identifica a presença de um conjunto de instruções está habilitado ou não. No seguinte trecho de código pode-se ver como usar `CPUID` para identificar se o conjunto de instrução AES-NI é suportado. Adicionalmente, recomenda-se a execução do programa *FeatureDetector*, que pode ser encontrado em [Yee15], para verificar se os outros conjuntos de instruções são suportados.

```
#include <stdio.h>
#define cpuid( func , ax , bx , cx , dx ) \
  __asm__ __volatile__ (" cpuid " : \
    "=a" ( ax ) , "=b" ( bx ) , \
    "=c" ( cx ) , "=d" ( dx ) : \
    "a" ( func ));
int Check_CPU_support_AES() {
  unsigned int a ,b ,c , d ;
  cpuid (1 ,a ,b ,c , d );
  return ( c & 0x2000000 );
}
void main (){
  printf("O processador suporta AES-NI: %s\n",
  Check_CPU_support_AES() ? "Sim" : "Nao" );
}
```

3.3. Técnicas para implementação segura

Utilizar criptografia de maneira segura na prática é não-trivial. Algoritmos criptográficos fornecem suas propriedades de segurança apenas em condições muito precisas, quanto todas as premissas são devidamente satisfeitas. Alguns erros comuns observados em sistemas em produção quando da utilização de algoritmos criptográficos são a geração, distribuição e armazenamento inseguros de chaves criptográficas, a escolha inadequada de primitivas, a utilização de algoritmos obsoletos, o desenvolvimento de algoritmos e implementações próprias sem revisão externa, a validação descuidada da titularidade de chaves públicas e certificados digitais, e o vazamento

de informação sensível por canais laterais [Sta10] ou recuperação de dados latentes da memória principal [HSH+09]. Como este capítulo se restringe à implementação segura e eficiente dos algoritmos criptográficos propriamente ditos, sem tratar em detalhes de sua utilização correta, a discussão será limitada ao escopo de implementação e mitigação de ataques de canal lateral.

3.3.1. Ataques de canal lateral

Eficiência não é o único requisito para aplicações práticas de criptografia, especialmente em sistemas embarcados que expõem superfícies de ataque consideráveis e alto potencial para manipulação direta por agentes maliciosos. Intervenções desse tipo podem incluir a medição em tempo real de características como tempo de execução, consumo de energia, emanações acústicas e eletro-magnéticas; e injeção de falhas para interromper a operação normal de uma função criptográfica, sempre na esperança de extrair informação secreta e material de chave diretamente do estado interno do algoritmo [Sta10]. É possível mitigar essas ameaças com o projeto cuidadoso das implementações de algoritmos criptográficos, aplicando técnicas para execução regular e invariante no tempo, de forma a desassociar padrões de computação com informação sensível.

Algumas vezes os recursos necessários para se implementar um algoritmo criptográfico de forma segura já estão disponíveis na plataforma, na forma de instruções dedicadas ou organização da arquitetura. Na maioria dos casos, o implementador precisa tomar cuidado adicional para verificar se suas contramedidas são corretamente traduzidas em linguagem de máquina pelo compilador ou utilizar linguagem de montagem diretamente. Em software, o escopo do que pode ser feito é bastante limitado, pois é difícil ter controle de granularidade fina sobre todos os aspectos da execução. Por essa razão, a ênfase será dada a contramedidas simples que protegem implementações contra ataques de canal lateral baseados no tempo.

3.3.2. Contramedidas algorítmicas

Implementações seguras de algoritmos criptográficos devem ser realizadas em tempo invariante com a informação sensível processada (dados a serem cifrados ou chave criptográfica), caso contrário um adversário capaz de monitorar o tempo de execução do algoritmo pode correlacionar suas observações com informação secreta. A monitoração não precisa ser necessariamente local e pode acontecer via iteração remota, pois já foi demonstrado que a latência de rede não insere ruído o suficiente para esconder pequenas variações do tempo de execução, mesmo que na granularidade de ciclos [BT11, BB05]. As variações ocorrem especialmente em trechos críticos que manipulam a chave privada, validam o formato de preenchimento (*padding*) ou conferem um valor de resumo ou autenticador.

Pequenas variações de tempo de execução podem se apresentar durante a comparação de cadeias de caracteres, desvios condicionais dependentes de bits da chave e limitantes do número de iterações de laços. No primeiro caso, se a interrupção do laço acontecer exatamente na primeira posição diferente entre as cadeias sendo comparadas, o adversário é capaz de realizar um ataque de busca exaustiva

em cada *byte* individualmente, baseado no tempo de resposta da comparação. Desta forma, é possível determinar o autenticador de uma mensagem, por exemplo, sem conhecimento da chave. O trecho de código abaixo ilustra uma forma segura de comparação:

```
int cmp_const(const void * a, const void * b,
              const size_t size) {
    const unsigned char *_a = a, *_b = b;
    unsigned char result = 0;
    size_t i;
    for (i = 0; i < size; i++) {
        result |= _a[i] ^ _b[i];
    }
    return result;
}
```

Desvios condicionais podem ser outra fonte de problemas [Koc96], especialmente quando se considera execução especulativa em processadores modernos ou o efeito da predição de desvios [AcKKS07]. A remoção de desvios condicionais envolve a aplicação de técnicas de programação sem desvios condicionais. A aplicação correta dessas técnicas é altamente dependente do algoritmo sendo estudado, mas uma generalização útil é calcular os dois ramos do desvio condicional simultaneamente e utilizar operações mascaradas para selecionar o valor correto apenas ao final da execução. A ideia é ilustrada pelo trecho de código abaixo para seleção entre dois valores em tempo constante, dependendo de um bit:

```
unsigned select(unsigned a, unsigned b,
                unsigned bit) {
  /* -0 = 0, -1 = 0xFF....FF */
  unsigned mask = - bit;
  unsigned ret = mask & (a^b);
  ret = ret ^ a;
  return ret;
}
```

Outra possibilidade é utilizar uma variante da função de seleção para alterar em tempo constante os valores de entrada do trecho de código protegido ser executado. Por fim, cuidado adicional deve ser tomado também para não permitir que o número de iterações de laços sejam variáveis com informação sensível, por exemplo o comprimento de uma chave criptográfica em um sistema criptográfico assimétrico.

3.3.3. Contramedidas de acesso à memória

Acessos à memória devem também ser cuidadosamente protegidos. No contexto de cifras de bloco, a preocupação se concentra na representação de caixas de substituição como vetores ou introdução de qualquer tabela pré-calculada para acelerar operações sobre bits realizadas pela cifra. O algoritmo AES ilustra muito bem o problema, pois seu desempenho depende enormemente da eficiência das caixas de substituição, motivando o implementador a utilizar tabelas simples. Entretanto, um

adversário capaz de monitorar o comportamento da memória *cache* pode determinar que porções da caixa de substituição são usadas na etapa de cifração e recuperar informação sensível [Ber04, Per05, BM06, TOS10].

Existem contramedidas de diversos tipos para mitigar o problema. Uma opção simples é adotar arquiteturas com latência de acesso uniforme à memória, como alguns microcontroladores simples. Outra possibilidade é utilizar uma implementação em hardware do algoritmo, quando disponível, que deve oferecer uma superfície de ataque menor. Alternativas mais sofisticadas para implementação em software são *bitslicing*, onde as operações sobre bits são realizadas explicitamente, sem ajuda de tabelas pré-calculadas, e o impacto em desempenho é reduzido pela aplicação do mesmo circuito lógico a bits de diferentes variáveis simultaneamente [Bih97, KS09]. Para tabelas pequenas, também é possível utilizar uma instrução para acesso a tabela armazenada em um registrador [Ham09] ou percorrer a tabela inteira a cada leitura, utilizando a função de seleção apresentada anteriormente para realizar uma cópia condicional entre o valor lido e um valor atual da variável de interesse.

A implementação segura de criptografia assimétrica, é comum a necessidade de manipular vetores em memória em tempo constante. Por exemplo, algoritmos de exponenciação costumam calcular pequenos múltiplos da base para acelerar o cálculo de uma potência da chave privada. Se o adversário puder recuperar as posições de memória sendo acessadas ao longo das iterações do algoritmo, o expoente privado termina relevado. A mitigação mais comum para esse problema é percorrer toda a tabela utilizando uma variação da função de seleção para vetores, na forma de uma cópia condicional sem desvios, que terminará visitando toda a tabela em ordem determinística mas copiando apenas as posições desejadas:

```
1 void copy_cond(int *c, const int *a,
2                int  d, dig_t b) {
3   dig_t mask, t;
4   mask = -b;
5   for (int i = 0; i < d; i++) {
6     t = (a[i] ^ c[i]) & mask;
7     c[i] ^= t;
8   }
9 }
```

Como será visto adiante, outros algoritmos de exponenciação possuem padrão regular de execução em cada uma de suas iterações, mas as variáveis de entrada e saída podem ser diferentes dependendo dos bits da chave [Mon87]. Nesse caso, monitorar a latência de acesso à memória pode revelar ao adversário informação sobre a chave a partir dos acertos ou erros na memória *cache*. Uma solução para o problema, é realizar trocas condicionais entre as variáveis a cada iteração a partir de uma variante da cópia condicional encontrada abaixo. Por mais que as versões das funções sejam apresentadas para vetores de palavras do processador, é possível aprimorar seu desempenho pela implementação de versões de maior granularidade utilizando instruções vetoriais.

```
void swap_cond(int *c, int *a, int d, dig_t b) {
  dig_t mask, t;

  mask = -b;
  for (int i = 0; i < d; i++) {
    t = (a[i] ^ c[i]) & mask;
    a[i] ^= t;
    c[i] ^= t;
  }
}
```

Outras técnicas para implementação segura de algoritmos criptográficos podem ser encontradas em `https://cryptocoding.net/`.

3.4. Técnicas para acelerar operações criptográficas

O objetivo desta seção é apresentar exemplos concretos sobre a implementação de algoritmos criptográficos, e mostrar conjuntos avançados de instruções que podem ser usados para acelerar a implementação de: o algoritmo de encriptação AES, a função de resumo SHA-3 e o protocolo de acordo de chaves baseado na curva elíptica *Curve25519*.

Para cada algoritmo, serão descritas as operações necessárias para seu funcionamento; os detalhes de como implementá-lo eficientemente em uma arquitetura de 64 bits, que preferencialmente suporte os conjuntos avançados de instruções; e finalmente os resultados de desempenho serão reportados.

3.4.1. Aceleração do algoritmo de encriptação AES

Esta seção descreve como maximizar o uso do suporte nativo em *hardware* para acelerar a computação do algoritmo de encriptação AES. Inicialmente, as operações realizadas para encriptar um bloco de dados serão descritas e posteriormente como acelerar a execução destas operações através de instruções especiais presentes nos processadores atuais.

3.4.1.1. Descrição do algoritmo AES

O algoritmo *Advanced Encryption Standard* (AES) foi publicado pelo NIST em 2001 [Nat01a]. AES é um cifrador de bloco que encripta uma mensagem M de 128 bits usando uma chave k e produz um texto cifrado C de 128 bits. O tamanho da chave pode ser de 128, 192 ou 256 bits; o cifrador AES é denotado por AES-128, AES-192 ou AES-256 dependendo do tamanho da chave usada.

O Algoritmo 1 descreve a sequência de operações do AES, cuja entrada é uma chave k de 128, 192 ou 256 bits e uma mensagem de 128 bits, que pode ser visto como uma matriz S de 4×4 bytes, denotada por *estado*; o AES modifica o estado iterativamente usando um conjunto de operações, onde o número de iterações N depende do tamanho da chave. O estado é modificado a cada rodada pelas seguintes transformações:

- `SubBytes`. O estado é atualizado pela aplicação da caixa de substituição da seguinte maneira: seja s algum byte do estado a ser substituído; primeiramente se calcula t' que é o inverso multiplicativo de s no corpo $\mathbb{F}_{2^8}$, quando $s = 0$ então $t' = 0$. Depois, seja $(t'_7, \ldots, t'_0)_2$ a representação na base 2 de t'. Então é possível substituir s pelo byte $t = At' + b$, onde A e b estão definidas no padrão [Nat01a]. As operações de multiplicação e soma devem ser feitas no corpo binário $\mathbb{F}_2$, isto é, com a aritmética modulo 2. A caixa de substituição foi projetada para que possua uma baixa correlação entre os bits de entrada e os bits de saída [DR01].

- `ShiftRows`. Realiza uma rotação nas linhas da matriz, isso assegura que os bytes de uma coluna sejam dispersos nas quatro colunas.

$$\begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_5 & t_6 & t_7 & t_4 \\ t_{10} & t_{11} & t_8 & t_9 \\ t_{15} & t_{12} & t_{13} & t_{14} \end{bmatrix} = \texttt{ShiftRows}\left(\begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_4 & t_5 & t_6 & t_7 \\ t_8 & t_9 & t_{10} & t_{11} \\ t_{12} & t_{13} & t_{14} & t_{15} \end{bmatrix}\right) \tag{3}$$

- `MixColumns`. Calcula a multiplicação do estado, visto como matriz, com uma matriz predefinida. Diferente da operação `SubBytes`, aqui as operações aritméticas são realizadas no corpo binário $\mathbb{F}_{2^8}$.

$$\begin{bmatrix} u_0 & u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 & u_7 \\ u_8 & u_9 & u_{10} & u_{11} \\ u_{12} & u_{13} & u_{14} & u_{15} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} t_0 & t_1 & t_2 & t_3 \\ t_5 & t_6 & t_7 & t_4 \\ t_{10} & t_{11} & t_8 & t_9 \\ t_{15} & t_{12} & t_{13} & t_{14} \end{bmatrix} \tag{4}$$

- `AddRoundKey`. Aplica uma operação XOR entre o estado e a chave de rodada correspondente, como pode ser visto nas linhas 3, 8 e 12 do Algoritmo 1.

A chave é usada para gerar um conjunto de *chaves de rodada* as quais são computadas mediante uma função de expansão de chaves, descrita no Algoritmo 2. Neste algoritmo são introduzidas duas operações que operam sobre palavras de 32 bits:

- `RotWord`. Seja $(t_3, t_2, t_1, t_0)_8$ a representação na base 8 da palavra T, então a função de rotação está definida como segue:

$$(t_2, t_1, t_0, t_3)_8 \leftarrow \texttt{RotWord}(T). \tag{5}$$

- `SubWord`. Seja $(t_3, t_2, t_1, t_0)_8$ a representação na base 8 da palavra T, a operação consiste em aplicar a caixa de substituição descrita em `SubBytes` para cada byte:

$$(t'_3, t'_2, t'_1, t'_0)_8 \leftarrow \texttt{SubWord}(T),\ t'_i \leftarrow \texttt{SubBytes}(t_i)\ \text{ para } i \in \{0, 1, 2, 3\}. \tag{6}$$

O conjunto de chaves de rodada pode ser gerado antes da execução do algoritmo de encriptação. Essas chaves podem ser reutilizadas para gerar as chaves de rodada do processo de decriptação.

Algoritmo 1 Encriptação de um bloco de 128 bits usando AES.

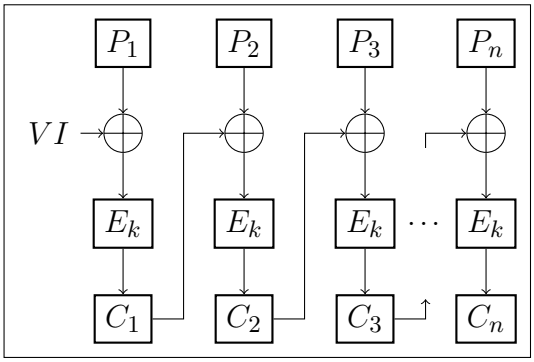
Entrada: M, uma cadeia binária de 128 bits.
k, uma chave de tamanho $l \in \{128, 192, 256\}$ bits.
Saída: C, uma cadeia binária de 128 bits que representa o texto encriptado de M.

$\{R_0, \ldots, R_N\} \leftarrow \texttt{KeyExpansion}(k)$
$N \leftarrow \begin{cases} 10 & \textbf{if } l = 128 \\ 12 & \textbf{if } l = 192 \\ 14 & \textbf{if } l = 256 \end{cases}$
$S \leftarrow M \oplus R_0$
for $i \leftarrow 1$ **to** $N - 1$ **do**
  $T \leftarrow \texttt{SubBytes}(S)$
  $T' \leftarrow \texttt{ShiftRows}(T)$
  $U \leftarrow \texttt{MixColumns}(T')$
  $S \leftarrow U \oplus R_i$
end for
$S \leftarrow \texttt{SubBytes}(S)$
$S \leftarrow \texttt{ShiftRows}(S)$
$C \leftarrow S \oplus R_N$
return C

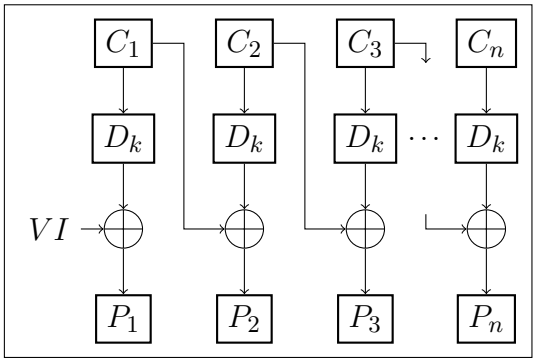
Algoritmo 2 Expansão de chaves do algoritmo AES (`KeyExpansion`).

Entrada: k, uma chave de tamanho $l \in \{128, 192, 256\}$ bits.
Saída: $\{R_0, \ldots, R_N\}$, o conjunto de chaves de rodada.

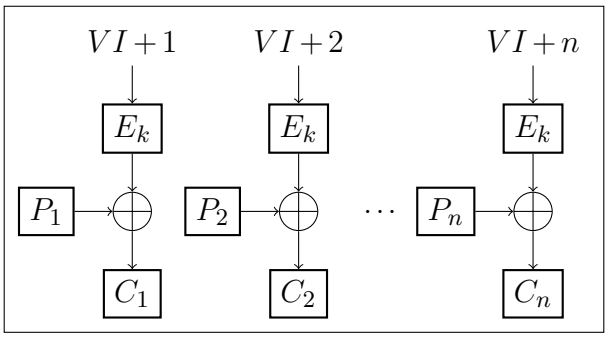
$N_k \leftarrow \frac{l}{32}$
$N \leftarrow \begin{cases} 10 & \textbf{if } l = 128 \\ 12 & \textbf{if } l = 192 \\ 14 & \textbf{if } l = 256 \end{cases}$
$\texttt{RCon}_i \leftarrow 2^{i-1} \in \mathbb{F}_{2^8}$ para $i \in \{1, \ldots, 10\}$.
Seja $(w_{N_k-1}, \ldots, w_0)_{32}$ a representação na base 32 da chave k.
for $i \leftarrow N_k$ **to** $4N + 3$ **do**
  $T \leftarrow w_{i-1}$
  if $i \equiv 0 \bmod N_k$ **then**
    $T \leftarrow \texttt{SubWord}(\texttt{RotWord}(T)) \oplus \texttt{RCon}_{i/N_k}$
  else if $N_K > 6$ **and** $i \equiv 4 \bmod N_k$ **then**
    $T \leftarrow \texttt{SubWord}(T)$
  end if
  $w_i \leftarrow w_{i-N_k} \oplus T$
end for
for $i \leftarrow 0$ **to** N **do**
  $R_i \leftarrow (w_{4i} \parallel w_{4i+1} \parallel w_{4i+2} \parallel w_{4i+3})$
end for
return $\{R_0, \ldots, R_N\}$



(a) Encriptação no modo CBC.



(b) Decriptação no modo CBC.



(c) Modo CTR.

Figura 3.3: Modos de operação definidos pelo NIST no padrão SP 800-38A [Nat01b].

3.4.1.2. Modos de operação

O algoritmo AES recebe como entrada um bloco de 128 bits e uma chave para produzir um texto encriptado do mesmo tamanho. Se o texto claro for maior que 128 bits ainda é possível usar o AES, dividindo-o em blocos de 128 bits. No entanto, quando múltiplos blocos são encriptados usando a mesma chave alguns problemas de segurança podem surgir.

Para resolver esse problema, existem algoritmos chamados *modos de operação* que permitem que os cifradores de bloco sejam usados para mensagens de qualquer tamanho. Esses modos dividem a mensagem em n blocos, $P = \{P_1, \ldots, P_n\}$, e utilizam o cifrador de bloco como primitiva básica para obter o texto cifrado, $C = \{C_1, \ldots, C_n\}$. O NIST definiu no padrão SP 800-38A cinco modos de operação: ECB, CBC, CFB, OFB e CTR [Nat01b]. Aqui serão detalhados dois dos mais utilizados.

No modo de operação *Cipher Block Chaining* (CBC) a computação do texto cifrado é feita como segue: o primeiro bloco, C_1, é computado como pela encriptação de $P_1 \oplus VI$, onde VI é um vetor de inicialização; os blocos subsequentes são computados por $C_i = E_k(P_i \oplus C_{i-1})$, para $2 \leq i \leq n$. Como pode ser visto na Figura 3.3a, a geração do texto encriptado é inerentemente sequêncial. Entretanto, a decriptação pode ser computada mais rapidamente, pois depende apenas dos blocos de texto

encriptado, veja a Figura 3.3b.

O modo de operação *Counter* (CTR) é um modo mais flexível e eficiente do que o CBC. O CTR calcula o texto cifrado da seguinte maneira: cada bloco do texto cifrado C_i é calculado pela operação XOR entre P_i e o resultado da encriptação de $VI+i$, para $1 \leq i \leq n$. A encriptação de um texto claro pode ser feita em paralelo, sendo mais eficiente do que o modo CBC. Além disso, o processo de decriptação é análogo ao de encriptação e pode ser computado por $P_i = C_i \oplus E_k(VI+i)$, para $1 \leq i \leq n$. Na Figura 3.3c está ilustrado o modo de operação CTR.

O vetor de inicialização (VI) tem o efeito de aleatorizar o processo de encriptação, portanto deve ser escolhido aleatoriamente sempre que uma encriptação é realizada. Por conta disso, é recomendado que o mesmo vetor de inicialização não seja utilizado mais de uma vez, caso contrário o modo de operação se torna inseguro.

3.4.1.3. Implementações do algoritmo AES

A maioria das operações usadas pelo AES podem ser calculadas através de operações binárias que estão presentes na maioria dos processadores atuais. Entretanto, uma implementação eficiente e resistente a ataques de canal lateral do AES não é uma tarefa trivial. Um conjunto de técnicas que podem ser usadas para conseguir uma implementação com tais características é apresentado a seguir:

- **Tabelas de consulta.** O AES pode ser implementado eficientemente usando tabelas de consulta; nessa técnica uma parte do processamento do algoritmo pode ser substituída diretamente por um acesso a uma tabela previamente computada. Essa técnica, entretanto, requer que os acessos à tabela sejam feitos de forma segura, evitando assim os ataques de canal lateral.

- **Bitslice**. A técnica de implementação *bitslicing* foi proposta por Biham para o algoritmo DES [Bih97]. Posteriormente essa técnica começou a ser usada também no AES [RSD06, MN07, KS09], visando aumentar o desempenho, pois com o *bitslice* as transformações do AES podem ser descritas na forma de circuitos lógicos que operam bit a bit.

- **AES-NI**. Em 2010 a Intel agregou a seus processadores um novo conjunto de instruções, *Intel Advanced Encryption Standard New Instructions* (AES-NI), que implementa todas as etapas do algoritmo AES em *hardware*, resultando assim em uma implementação do AES mais eficiente que as implementações de *software* conhecidas até então. Além de permitir a implementação do AES com alto desempenho as novas instruções também fornecem benefícios de segurança, pois elas não utilizam tabelas de consulta e seu tempo de execução deve ser igual, independente dos dados de entrada, eliminando assim os principais ataques de tempo conhecidos [Gue10].

Na seção seguinte serão apresentados mais detalhes de como as novas instruções são aplicadas na implementação segura e eficiente do algoritmo AES.

3.4.1.4. Aceleração usando instruções AES-NI

Em 2010, a Intel lançou uma arquitetura chamada *Westmere*, que adicionou seis novas instruções para acelerar o processo de encriptação usando o algoritmo AES. Esse novo conjunto de instruções é conhecido como AES-NI e tais instruções podem ser usadas diretamente em linguagem de montagem ou também como funções em linguagem C (comumente conhecidas como *intrinsics* [Cor14]).

As instruções AES-NI usam registradores de 128 bits para armazenar a informação da chave e o estado. A seguir são apresentadas as instruções presentes no conjunto AES-NI:

- `AESENC`. Realiza uma rodada da função de encriptação do AES. Aplica as operações de `ShiftRows`, `SubBytes` e `MixColumns` sobre um registrador `XMM1`, que contém o estado; posteriormente é aplicado uma operação XOR entre o resultado e a chave de rodada, guardada no registrador `XMM2`.

- `AESENCLAST`. Computa a última rodada de função de encriptação do AES. Aplica as operações de `ShiftRows` e `SubBytes` sobre um registrador `XMM1`, que contém o estado; posteriormente é aplicado uma operação XOR entre o resultado e a chave de rodada, guardada no registrador `XMM2`.

- `AESDEC/AESDECLAST`. Essas instruções são análogas às instruções previamente explicadas, com a diferença que estas são usadas para computar a decriptação.

- `AESKEYGENASSIST`. Essa instrução é usada para gerar as chaves de rodada do AES; na Figura 3.4d pode ser visto o código que é usado para gerar as chaves de rodada do algoritmo AES-128.

- `AESIMC`. Essa função gera as chaves de rodada da decriptação a partir das chaves de rodada da encriptação, usando o trecho de código na Figura 3.4c.

Os trechos de código abaixo mostram como implementar em linguagem de montagem o algoritmo AES-128 para encriptar um bloco de 128 bits. Inicialmente, as chaves de rodada previamente computadas se encontram armazenadas nos registradores `XMM0` a `XMM10`, e o estado S é armazenado no registrador `XMM15`. A primeira instrução faz um XOR com a primeira chave de rodada e depois utiliza nove vezes a instrução `AESENC` para computar uma rodada. Finalmente, a computação da última rodada é executada pela instrução `AESENCLAST`. Nas Figuras 3.4a e 3.4b são mostradas as sequencias de instruções para realizar a encriptação e a decriptação de blocos de 128 bits, respectivamente.

3.4.1.5. Testes de desempenho

Alguns experimentos foram projetados para nos auxiliar a observar os ganhos obtidos com a utilização do conjunto AES-NI.

```
pxor    xmm15, xmm0       ; AddRoundKey
aesenc  xmm15, xmm1       ; Round 1
aesenc  xmm15, xmm2       ; Round 2
aesenc  xmm15, xmm3       ; Round 3
aesenc  xmm15, xmm4       ; Round 4
aesenc  xmm15, xmm5       ; Round 5
aesenc  xmm15, xmm6       ; Round 6
aesenc  xmm15, xmm7       ; Round 7
aesenc  xmm15, xmm8       ; Round 8
aesenc  xmm15, xmm9       ; Round 9
aesenclast xmm15, xmm10 ; Round 10
```

(a) Encriptação.

```
pxor    xmm15, xmm10     ; AddRoundKey
aesdec  xmm15, xmm9      ; Round 1
aesdec  xmm15, xmm8      ; Round 2
aesdec  xmm15, xmm7      ; Round 3
aesdec  xmm15, xmm6      ; Round 4
aesdec  xmm15, xmm5      ; Round 5
aesdec  xmm15, xmm4      ; Round 6
aesdec  xmm15, xmm3      ; Round 7
aesdec  xmm15, xmm2      ; Round 8
aesdec  xmm15, xmm1      ; Round 9
aesdeclast xmm15, xmm0 ; Round 10
```

(b) Decriptação.

```
mov rdx, OFFSET Key_Schedule
mov rax, OFFSET Key_Schedule_Decrypt
movdqu xmm1, XMMWORD PTR [rdx]
movdqu XMMWORD PTR [rax], xmm1
add rdx, 0x10
add rax, 0x10
mov ecx, 9   ;{9,11,13}

repeat:
  movdqu xmm1, XMMWORD PTR [rdx]
  aesimc xmm1, xmm1
  movdqu XMMWORD PTR [rax], xmm1
  add rdx, 0x10
  add rax, 0x10
  sub rcx, 0x01
  jnz repeat

movdqu xmm1, XMMWORD PTR [rdx]
movdqu XMMWORD PTR [rax], xmm1
```

(c) Geração das chaves de rodada para decriptação.

```
keygen_128:
  movdqu xmm1, XMMWORD PTR Keyindent
  movdqu XMMWORD PTR KeySch, xmm1
  mov rcx, OFFSET KeySch+16
  aeskeygenassist xmm2, xmm1, 0x01
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x02
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x04
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x08
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x10
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x20
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x40
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x80
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x1b
  call key_expansion_128
  aeskeygenassist xmm2, xmm1, 0x36
  call key_expansion_128
  jmp END

key_expansion_128:
  pshufd xmm2, xmm2, 0xFF
  pslldq xmm3, xmm1, 0x04
  pxor   xmm1, xmm3
  pslldq xmm3, xmm1, 0x04
  pxor   xmm1, xmm3
  pslldq xmm3, xmm1, 0x04
  pxor   xmm1, xmm3
  pxor   xmm1, xmm2
  movdqu XMMWORD PTR [rcx], xmm1
  add rcx, 0x10
  ret
END:
```

(d) Geração das chaves de rodada para encriptação.

Figura 3.4: Lista de códigos de montagem usados na implementação das funções de encriptação, decriptação e geração das chaves de rodada do algoritmo AES-128 usando as instruções AES-NI.

Teste 1. *Calcular os ciclos por byte e a vazão para encriptar mensagens de 50 MB usando o algoritmo AES com tamanho de chave de 128 bits e com os modos de operação CBC e CTR.*

O desempenho de dois códigos que implementam o algoritmo AES-128 serão mostrados, usando os modos de operação CBC e CTR. A primeira implementação

Implementação	Modo CBC		Modo CTR	
	Ciclos/byte	MB/s	Ciclos/byte	MB/s
Nativa de x64	11,08	306,08	11,67	290,51
AES-NI	4,50	752,36	0,83	4061,86

Tabela 3.2: Comparação do desempenho do AES-128-CBC e AES-128-CTR. Os resultados deste teste foram obtidos no computador HW-ULTRA especificado no Apêndice A.

não aproveita o suporte de encriptação por *hardware* e simplesmente utiliza instruções nativas de 64 bits. A segunda implementação, entretanto, utiliza as instruções AES-NI presentes nos processadores da Intel. Esses códigos fazem parte da biblioteca de exemplos da Intel (versão 1.2) que está disponível em [Rot11].

Os resultados do Teste 1 são apresentados na Tabela 3.2; pode-se perceber que no modo de operação CBC, a implementação que usa AES-NI é aproximadamente 2,5 vezes mais rápida que a implementação nativa de 64 bits. O impacto no desempenho causado pelas instruções AES-NI é ainda maior no CTR, que chega a ser 14 vezes mais rápido do que uma implementação sequencial nativa de 64 bits.

Teste 2. *Utilizar o modulo de medição de desempenho da biblioteca OpenSSL para determinar o tempo de execução do algoritmo AES-128-CBC. O teste será feito, com e sem suporte das instruções AES-NI.*

Neste experimento foi utilizado o OpenSSL, versão 1.0.2d [The03]. O desempenho do algoritmo AES pode ser medido pela seguinte linha de comando:

```
$ openssl speed -elapsed -evp aes-128-cbc | tail
```

Na hora de indicar o parâmetro `-evp` o OpenSSL escolherá a implementação adequada para a arquitetura alvo; caso seu computador não conte com o conjunto de instruções AES-NI, então será executado o teste com uma implementação suportada. Parte da saída deste comando, executada no processador HW-ULTRA, é apresentada a seguir:

```
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes
aes-128-cbc     285865.74k   306641.68k   311269.29k
type           1024 bytes   8192 bytes
aes-128-cbc     313865.59k   314833.08k
```

Agora, o mesmo experimento será executado indicando explicitamente, através de uma variável de ambiente, que execute uma implementação genérica, isto é, sem suporte ao AES-NI.

```
$ OPENSSL_ia32cap="~0x200000200000000" \
  openssl speed -elapsed -evp aes-128-cbc | tail
```

```
1 The 'numbers' are in 1000s of bytes per second processed.
2 type             16 bytes     64 bytes    256 bytes
3 aes-128-cbc    129931.97k   144134.36k   146498.66k
4 type           1024 bytes   8192 bytes
5 aes-128-cbc    148852.61k   149275.23k
```

Os resultados dos experimentos, mostram que a implementação usando AES-NI é acima de 2 vezes mais rápida do que a implementação genérica. Maiores detalhes sobre o desempenho do OpenSSL usando instruções AES-NI podem ser consultados em [GGF+15].

Observações: esses dois experimentos são uma prova de que é possível atingir um nível de desempenho alto para encriptar dados desde que a implementação aproveite de maneira adequada as instruções especializadas contidas nos processadores.

3.4.2. Implementação da função de resumo SHA-3

Esta seção detalha como usar o conjunto de instruções vetoriais para acelerar a computação do mais novo padrão de funções de resumo SHA-3. Inicialmente será apresentada uma breve introdução da família de funções de resumo SHA-3 e, posteriormente, como a implementação em software do SHA-3 pode ser acelerada por meio de instruções vetoriais.

A família de funções de resumo *Standard Hash Algorithm* (SHA) [Nat08a] foi padronizada pelo Instituto Nacional de Padrões e Tecnologias americano (NIST) e atualmente é usada em muitas aplicações e protocolos. Nos últimos anos, a família SHA sofreu alguns ataques; em 2005, [BCJ+05] e [RO05] mostraram ataques à colisões em uma versão reduzida do SHA-1; no mesmo ano, [WYY05] mostrou um ataque que quebra, teoricamente, a resistência à colisão do SHA-1. A segunda versão da família, SHA-2, possui uma estrutura muito similar ao SHA-1 e já possui alguns ataques à sua versão reduzida, como é mostrado em [IMPR09].

Adicionalmente, foram feitas críticas aos algoritmos SHA-1 e SHA-2 por seus detalhes de projeto serem fechados ao público. Por este motivo, após um período de consulta pública e dois *workshops*, o NIST decidiu publicar, em 2007, uma chamada aberta para a seleção de um novo algoritmo, o SHA-3. Ao contrário dos algoritmos anteriores, o concurso para a escolha do SHA-3 foi feito nos mesmos moldes do AES, algoritmo padrão de cifra de bloco, selecionado em 2001 após cinco anos de concurso [Nat01a]. Por esse motivo, todos os algoritmos que foram submetidos tiveram que ter suas patentes abertas e o código disponível, afim de serem analisados e testados durante o concurso. Em 2012, após três rodadas de competição, o algoritmo Keccak [BDPA11] foi anunciado como vencedor [jCPB+12], este algoritmo baseia-se na construção esponja definida sobre uma permutação Keccak-p.

Em Agosto de 2015 foi publicado o documento oficial FIPS 202 [Nat15], que descreve os detalhes da família SHA-3 e padroniza quatro funções de resumo, SHA3-224, SHA3-256, SHA3-384 e SHA3-512 e duas funções de resumo com saída variável, *Extendable Output Functions* (XOF), chamadas SHAKE128 e SHAKE256.

Funções	Tamanho de saída	Nível de segurança em bits		
		Colisão	Pré-imagem	2ª Pré-imagem
SHA-1	160	<80	160	$160-L(M)$
SHA-224	224	112	224	$\min(224, 256-L(M))$
SHA-512/224	224	112	224	224
SHA-256	256	128	256	$256-L(M)$
SHA-512/256	256	128	256	256
SHA-384	384	192	384	384
SHA-512	512	256	512	$512-L(M)$
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE128	d	$\min(d/2, 128)$	$\geq \min(d, 128)$	$\min(d, 128)$
SHAKE256	d	$\min(d/2, 256)$	$\geq \min(d, 256)$	$\min(d, 256)$

Tabela 3.3: Nível de segurança das funções SHA-1, SHA-2 e SHA-3 [Nat15].

Os parâmetros de todas as funções da família SHA podem ser encontrados na Tabela 3.3. Nas seções seguintes serão apresentadas a família SHA-3 baseada no algoritmo Keccak e como implementar esse algoritmo eficientemente em arquiteturas de 64 bits.

3.4.2.1. Permutações Keccak-p

Uma permutação Keccak-p que possui n_r rodadas e trabalha sobre um estado S de b bits é denotada por Keccak-$p[b, n_r]$; a permutação é definida para quaisquer $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ e qualquer inteiro positivo n_r. Uma rodada da permutação Keccak-p consiste de uma sequência de cinco transformações, chamadas de *etapas de mapeamento*.

O estado é organizado em uma matriz $5 \times 5 \times w$, onde $w = b/25$ pode ser visto como o tamanho das palavras do estado em bits. As 25 palavras de w bits de um estado S são denotadas mediante s_i para $0 \leq i < 25$ como pode ser visto a seguir:

$$S = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 & s_4 \\ s_5 & s_6 & s_7 & s_8 & s_9 \\ s_{10} & s_{11} & s_{12} & s_{13} & s_{14} \\ s_{15} & s_{16} & s_{17} & s_{18} & s_{19} \\ s_{20} & s_{21} & s_{22} & s_{23} & s_{24} \end{bmatrix}; \; S[x, y] = s_{5x+y} \text{ para } 0 \leq x, y < 5. \tag{7}$$

Etapas de mapeamento. As cinco etapas de mapeamento usadas durante uma rodada da função Keccak-$p[b, n_r]$ são denotadas por θ, ρ, π, χ e ι. O algoritmo para cada uma dessas etapas tem como entrada um estado S e como retorno esse estado

atualizado. A etapa de mapeamento ι é a única que possui uma entrada adicional, um inteiro i_r chamado índice da rodada.

O tamanho do estado é um parâmetro que está omitido da notação, pois b sempre é especificado quando as etapas de mapeamento são chamadas; visando simplificar a notação, todas as operações lógicas e de rotação usadas nas etapas de mapeamento são aplicadas sobre palavras de tamanho w.

As especificações das etapas de mapeamento são apresentadas a seguir:

- **Etapa de mapeamento** θ. O efeito da etapa θ é aplicar uma operação XOR entre cada palavra do estado com a paridade de duas colunas do estado.
- **Etapa de mapeamento** ρ. Nesta etapa cada palavra do estado é rotada uma quantidade fixa de r_i bits. A seguir pode-se ver a matriz R, onde cada elemento r_i representa a quantidade de bits que a palavra s_i será rotada.

$$R = \begin{bmatrix} 0 & 1 & 62 & 28 & 27 \\ 36 & 44 & 6 & 55 & 20 \\ 3 & 10 & 43 & 25 & 39 \\ 41 & 45 & 15 & 21 & 8 \\ 18 & 2 & 61 & 56 & 14 \end{bmatrix}; \; R[x,y] = r_{5x+y} \text{ para } 0 \leq x, y < 5. \tag{8}$$

- **Etapa de mapeamento** π. O efeito da etapa de mapeamento π é embaralhar as palavras do estado. Ela promove uma difusão a longo prazo dentro das rodadas, a fim de evitar que padrões sejam explorados em determinados ataques.
- **Etapa de mapeamento** χ. O efeito da etapa de mapeamento χ é aplicar uma operação XOR em cada palavra do estado s_i com a saída de uma função não linear aplicada a duas palavras da mesma linha de s_i.
- **Etapa de mapeamento** ι. A etapa de mapeamento ι consiste na aplicação de uma operação XOR entre o elemento s_0 com um valor contante $rc(i_r)$, onde i_r é o índice da rodada; os valores de rc são definidos em [Nat15] e são gerados a partir de um *Linear Feedback Shift Register* (LFSR).

Dado um estado S e o índice de uma rodada i_r, uma função Rnd é a transformação que resulta na aplicação das etapas de mapeamento θ, ρ, π, χ e ι, como pode ser visto a seguir:

$$\text{Rnd}(S, i_r) = \iota(\chi(\pi(\rho(\theta(S)))), i_r). \tag{9}$$

A permutação Keccak-$p[b, n_r]$ consiste de n_r iterações de Rnd.

3.4.2.2. Construção esponja

A construção esponja [BDPA07] define uma função `ESPONJA`[f, `pad`, r] com domínio $\{0,1\}^*$ e codominio $\{0,1\}^\infty$ usando uma permutação de tamanho fixo f, uma regra

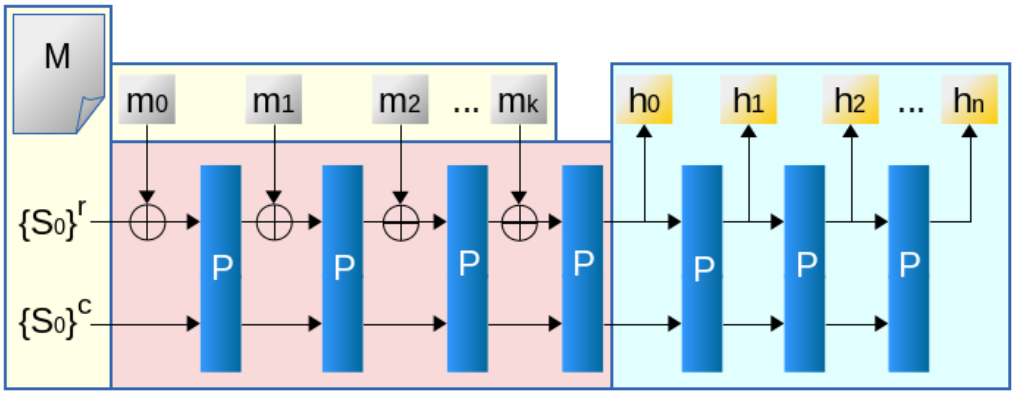


Figura 3.5: Construção esponja: $Z = \texttt{ESPONJA}[f, \texttt{pad}, r](M, d)$.

de preenchimento `pad` e uma *taxa de bits* r. A partir desta função, uma saída de tamanho finito pode ser obtida pelo truncamento dos primeiros l bits. Uma instância da construção esponja é chamada de *função esponja*.

A função esponja opera sobre um estado de $b = r + c$ bits; onde r é a taxa de bits e c é a *capacidade*. Primeiramente, todos os bits do estado são inicializados com zeros. A regra de preenchimento pad é aplicada sobre a mensagem de entrada e posteriormente a mensagem é dividida em k blocos de r bits; uma vez feito isso o processamento é composto por duas fases: a fase de absorção e a fase de extração.

Na *fase de absorção* é aplicado uma operação XOR entre um bloco de r bits da mensagem de entrada com os primeiros r bits do estado atual. Então, o estado é atualizado por meio de uma permutação f; esse processo é aplicado para cada um dos k blocos da mensagem.

Na *fase de extração* os primeiros r bits do estado são usados como saída; se o tamanho de l for maior que o tamanho de r, o estado é processado novamente pela permutação f e os r bits retornados são concatenados com os r bits retornados previamente; esse processo é repetido até que os l bits requeridos sejam extraídos.

Os últimos c bits do estado nunca são afetados diretamente pelos blocos de entrada na fase de absorção e nem extraídos na fase de extração; o parâmetro c determina a segurança atingível pela construção [BDPA08].

A construção esponja é ilustrada na Figura 3.5 e seu pseudo-código é apresentado no Algoritmo 3.

3.4.2.3. O algoritmo Keccak

Keccak é uma família de funções de resumo, originalmente definida em [BDPA11]. A seguir, a regra de preenchimento usada pela família Keccak será descrita com os parâmetros e a permutação fundamental usada pelo Keccak:

- **Regra de preenchimento pad10*1**. Recebe como entrada o tamanho da mensagem m e a taxa de bits r, retornando uma cadeia de dados $Z = 1||0^j||1$,

Algoritmo 3 $\texttt{ESPONJA}[f, \texttt{pad}, r](M, d)$

Entrada: Cadeia de dados M e um inteiro não negativo d.
Saída: Cadeia de dados Z, tal que $|Z| = d$.

$P = M \parallel \texttt{pad}(r, |M|)$
$n = |P|/r$
$c = b - r$
Seja $P = P_0 \parallel \ldots \parallel P_{n-1}$ a divisão de P em blocos de tamanho r.
$S = 0^b$
for all i tal que $0 \leq i < n$ **do**
    $S = f(S \oplus (P_i \parallel 0^c))$
end for
Seja Z uma cadeia de dados vazia.
while $d > |Z|$ **do**
    $Z = Z \parallel \texttt{Trunc}_r(S)$
    Se $d > |Z|$ então $S = f(S)$
end while
return $\texttt{Trunc}_d(Z)$

onde $j = -(m+2) \bmod r$.

- **Especificação de Keccak**$[c]$. A família de funções esponjas Keccak com a função de permutação Keccak-$p[b, n_r]$ e a regra de preenchimento pad10*1 está definida para quais quer par de taxa de bits r e capacidade c tal que $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$.

Quando o tamanho do estado é restrito para $b = 1600$, a família Keccak é denotada por Keccak[c]; nesse caso, r é determinado pela escolha do parâmetro c. Em particular,

$$\text{Keccak}[c] = \texttt{ESPONJA}[\text{Keccak-}p[1600, 24], \text{pad10}^*1, 1600 - c]. \tag{10}$$

Assim, dado uma mensagem M e um tamanho de saída d,

$$\text{Keccak}[c](M, d) = \texttt{ESPONJA}[\text{Keccak-}p[1600, 24], \text{pad10}^*1, 1600 - c](M, d). \tag{11}$$

3.4.2.4. Especificações da função SHA-3

Nesta seção, serão descritas as quatro funções de resumo SHA-3 e as duas funções com saída variável SHAKE128 e SHAKE256.

As quatro funções de resumo SHA-3 são definidas a partir da aplicação da função Keccak[c] sobre uma mensagem M concatenada com dois bits e a especificação do tamanho de saída, como pode ser visto a seguir:

$$\begin{aligned}
&\text{SHA3-224(M)} = \text{Keccak[448](M||01,224)} \\
&\text{SHA3-256(M)} = \text{Keccak[512](M||01,256)} \\
&\text{SHA3-384(M)} = \text{Keccak[768](M||01,384)} \\
&\text{SHA3-512(M)} = \text{Keccak[1024](M||01,512)}
\end{aligned}$$

Em cada caso, a capacidade sempre é o dobro do tamanho da saída; os dois bits adicionados à mensagem (01) servem para dar suporte à separação de domínio, por exemplo, distinguir as mensagens processadas pela função de resumo SHA-3 e pelas funções com saída variável.

As duas funções com saída variável, SHAKE128 e SHAKE256 são definidas a partir da função Keccak[c], uma mensagem M concatenada com quatro bits e pela especificação do tamanho de saída d, como pode ser visto a seguir:

$$\text{SHAKE128(M)} = \text{Keccak[256](M||1111,d)}$$
$$\text{SHAKE256(M)} = \text{Keccak[512](M||1111,d)}$$

Os primeiros dois bits servem para dar suporte a separação de domínio e os dois bits adicionais (11) são usados para prover compatibilidade ao esquema Sakura [BDPV14]. Esse esquema facilitará o desenvolvimento de uma extensão dessas funções, chamada resumos em árvore [Mer88], onde a mensagem pode ser processada em paralelo, conseguindo assim computar mensagens longas de forma mais eficiente.

3.4.2.5. Implementação nativa de 64 bits do SHA-3

Dado que a permutação Keccak-p[1600,24] trabalha com palavras de 64 bits, a implementação para uma arquitetura de 64 bits pode ser feita de forma direta. O primeiro passo para a implementação é definir como o estado do algoritmo será representado.

Organização do estado. As palavras do estado são mapeadas diretamente à variáveis de 64 bits; desse modo, para comportar as 25 palavras do estado $S = \{s_0, \ldots, s_{24}\}$, é preciso usar 25 variáveis de 64 bits, denominadas $M = \{M_0, \ldots, M_{24}\}$, onde $\{M_0 = s_0, M_1 = s_1, \ldots, M_{24} = s_{24}\}$.

Uma vez mapeado o estado a variáveis de 64 bits, pode-se começar a computação da permutação Keccak-p[1600,24]. Uma implementação direta de uma rodada, denotada $\text{Rnd}(M, i_r)$, pode ser vista no Algoritmo 4.

A maioria das arquiteturas de 64 bits atuais possuem menos que 25 registradores de propósito geral e sabe-se que operações em registradores são substancialmente mais rápidas que operações que buscam dados da memória. Por conta disso, uma implementação eficiente tem que aproveitar os dados dentro dos registradores na maior parte do tempo.

Visando um melhor aproveitamento dos registradores, pode-se implementar as etapas de mapeamento ρ, π, χ e ι de forma modular, processando as etapas ρ e π para um bloco de cinco palavras que serão usadas conjuntamente na etapa χ. O trecho de código a seguir é necessário para computar as novas palavras s_5, s_6, s_7, s_8 e s_9 após a aplicação das etapas de mapeamento ρ, π e χ:

Algoritmo 4 Rnd(M, i_r)

Entrada: Estado M e i_r o índice da rodada.
Saída: Estado M atualizado.

Etapa de mapeamento θ

1: $C_y = M_{0+y} \oplus M_{5+y} \oplus M_{10+y} \oplus M_{15+y} \oplus M_{20+y}$ para $0 \leq y < 5$
2: $D_x = C_{(x-1) \bmod 5} \oplus (C_{(x+1) \bmod 5} \lll 1)$ para $0 \leq x < 5$
3: $M_{5x+y} = M_{5x+y} \oplus D_x$ para $0 \leq x, y < 5$

Etapas de mapeamento ρ e π

4: $B_{(16x+10y) \bmod 25} = (M_{5x+y} \lll r_{5x+y})$ para $0 \leq x, y < 5$

Etapa de mapeamento χ

5: **for all** (x, y) tal que $0 \leq x, y < 5$ **do**
6: $\quad T = (B_{(5x+y+2) \bmod 5}) \wedge (\neg B_{(5x+y+1) \bmod 5})$
7: $\quad M_{5x+y} = B_{5x+y} \oplus T$
8: **end for**

Etapa de mapeamento ι

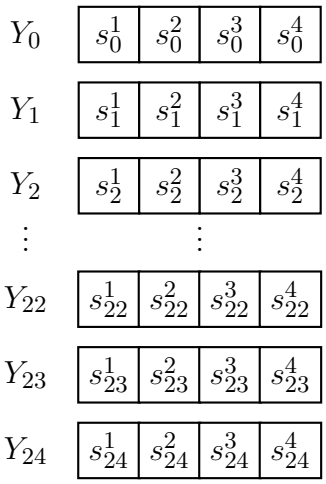
9: $M_0 = M_0 \oplus rc(i_r)$
10: **return** M

$$
\begin{aligned}
T_0 &= M_3 \lll 28 \\
T_1 &= M_9 \lll 20 \\
T_2 &= M_{10} \lll 3 \\
T_3 &= M_{16} \lll 45 \\
T_4 &= M_{22} \lll 61 \\
M_{16} &= T_0 \oplus (T_2 \oplus (\neg T_1)) \qquad & M_{16} &= \chi(\pi(\rho(s_5))) \\
M_3 &= T_1 \oplus (T_3 \oplus (\neg T_2)) & M_3 &= \chi(\pi(\rho(s_6))) \\
M_{10} &= T_2 \oplus (T_4 \oplus (\neg T_3)) & M_{10} &= \chi(\pi(\rho(s_7))) \\
M_{22} &= T_3 \oplus (T_0 \oplus (\neg T_4)) & M_{22} &= \chi(\pi(\rho(s_8))) \\
M_9 &= T_4 \oplus (T_1 \oplus (\neg T_0)) & M_9 &= \chi(\pi(\rho(s_9)))
\end{aligned}
$$

Após a execução das etapas ρ, π e χ as palavras computadas não ficaram armazenadas nos registradores M_5, M_6, M_7 e M_8, isso acontece porque as palavras armazenadas nesses registradores ainda não foram processadas pelas etapas ρ, π e χ. A computação da etapa ι depende apenas do registrador M_0, desse modo pode-se computá-la a qualquer momento após calcular o novo valor de X_0 após a execução da etapa χ.

Pode-se perceber que, após a execução dessas etapas na iteração i, o estado tem uma organização diferente da inicial. Por conta disso, a implementação das etapas de mapeamento para a iteração seguinte $(i+1)$ precisa levar em conta a nova configuração do estado. O estado volta à sua configuração original após aplicação das etapas de mapeamento ρ, π e χ na iteração $i+1$.

Com essas otimizações, pode-se dispensar o estado temporário usado no Algoritmo 4 e também diminuir o acesso a memória, visto que os dados podem ser encontrados em registradores na maior parte do tempo.



(a) Organização do estado para a implementação vetorial usando registradores de 256 bits.

$T_0 = \texttt{LOAD}(S_1)$	$[s_0^1, s_1^1, s_2^1, s_3^1]$
$T_1 = \texttt{LOAD}(S_2)$	$[s_0^2, s_1^2, s_2^2, s_3^2]$
$T_2 = \texttt{LOAD}(S_3)$	$[s_0^3, s_1^3, s_2^3, s_3^3]$
$T_3 = \texttt{LOAD}(S_4)$	$[s_0^4, s_1^4, s_2^4, s_3^4]$
$T_4 = \texttt{UNPACK\_LO}(T_0, T_1)$	$[s_0^1, s_0^2, s_2^1, s_2^2]$
$T_0 = \texttt{UNPACK\_HI}(T_0, T_1)$	$[s_1^1, s_1^2, s_3^1, s_3^2]$
$T_1 = \texttt{UNPACK\_LO}(T_0, T_1)$	$[s_0^3, s_0^4, s_2^3, s_2^4]$
$T_2 = \texttt{UNPACK\_HI}(T_0, T_1)$	$[s_1^3, s_1^4, s_3^3, s_3^4]$
$Y_0 = \texttt{PRBLEND}(T_4, T_1)$	$[s_0^1, s_0^2, s_0^3, s_0^4]$
$Y_1 = \texttt{PRBLEND}(T_0, T_2)$	$[s_1^1, s_1^2, s_1^3, s_1^4]$
$Y_2 = \texttt{PRBLEND}(T_4, T_1)$	$[s_2^1, s_2^2, s_2^3, s_2^4]$
$Y_3 = \texttt{PRBLEND}(T_0, T_2)$	$[s_3^1, s_3^2, s_3^3, s_3^4]$

(b) Sequencia de instruções para ordenar as palavras dos registradores Y_0, Y_1, Y_2 e Y_3.

Figura 3.6: Organização do estado e sequencia de instruções para organizar as primeiras quatro palavras do estado.

3.4.2.6. Implementação vetorial usando registradores de 256 bits

Uma forma direta de usar as instruções vetoriais para implementar a função de permutação Keccak-p[1600,24] é processar mais de um estado por vez. Tirando proveito dos registradores de 256 bits pode-se processar quatro estados concorrentemente. A seguir são apresentados os detalhes internos dessa implementação; primeiramente, como criar um novo estado, com registradores de 256 bits, que contenha os quatro estados a serem processados e posteriormente como aplicar as etapas de mapeamento sobre esse novo estado.

Organização do estado. As instruções de 256 bits processam quatro palavras de 64 bits usando apenas uma instrução. O primeiro passo para a implementação foi mapear os quatro estados a serem processados em registradores de 256 bits; para comportar as 100 palavras dos estados $S^1 = \{s_0^1, \ldots, s_{24}^2\}$, $S^2 = \{s_0^2, \ldots, s_{24}^2\}$, $S^3 = \{s_0^3, \ldots, s_{24}^3\}$ e $S^4 = \{s_0^4, \ldots, s_{24}^4\}$ foi preciso usar 25 registradores de 256 bits, denominados $Y = \{Y_0, \ldots, Y_{24}\}$. O mapeamento de cada uma das palavras do estado em seus respectivos registradores pode ser visto na Figura 3.6a.

O mapeamento dos estados S^1, S^2, S^3 e S^4 em Y pode ser feito eficientemente com o uso das instruções `LOAD`, `UNPACK_LO`, `UNPACK_HI` e `PERM`; na Figura 3.6b é apresentado o trecho de código que mapeia as primeiras quatro palavras dos estados nos registradores Y_0, Y_1, Y_2 e Y_3; o processamento para as outras palavras é análogo ao apresentado.

Uma vez mapeado os estados a registradores de 256 bits, pode-se começar a computação da função de permutação Keccak-p[1600,24]. A implementação dessa versão que usa registradores de 256 bits para processar quatro estados independentes concorrentemente é muito similar ao Algoritmo 4; a diferença é que aqui todas as

operações são sobre registradores de 256 bits enquanto no Algoritmo 4 as operações são sobre palavras de 64 bits.

Também pode-se aplicar diretamente todas as otimizações usadas na implementação de 64 bits. Esta implementação usa praticamente a mesma quantidade de instruções que a implementação 64 bits.

3.4.2.7. Testes de desempenho

Para analisar o desempenho das implementações do algoritmo SHA-3 aqui apresentadas projetamos alguns experimentos que nos auxiliarão observar os ganhos obtidos com a utilização de instruções vetoriais.

Teste 3. *Calcular os ciclos por byte para produzir o valor de resumo de uma mensagem de 50 MB usando a implementação nativa de 64 bits e para produzir os valores de resumo de quatro mensagens de 50 MB concorrentemente usando a implementação que aproveita as instruções vetoriais.*

A implementação nativa de 64 bits, foi desenvolvida por Ronny Van Keeris, e é a implementação mais rápida disponível no *ECRYPT Benchmarking of Cryptographic Systems* (eBACS)[BL15a]. Essa implementação foi otimizada para a arquitetura de 64 bits usando as técnicas apresentadas nesta seção. A implementação vetorizada é uma implementação que usa registradores de 256 bits para calcular quatro valores de resumo concorrentemente e faz parte das implementações vetoriais do SHA-3 que foram apresentadas em [CL14].

Na Figura 3.7a é apresentado os ciclos por bytes necessários para computar o valor de resumo de mensagens de 50 MB para cada uma das funções da família SHA-3. Pode-se perceber o impacto das instruções vetoriais na aceleração desse algoritmo, uma vez que calcular o valor de resumo de quatro mensagens de mesmo tamanho usando as instruções vetoriais é em torno de 2,5 vezes mais rápido do que fazer a mesma computação usando a implementação de 64 bits.

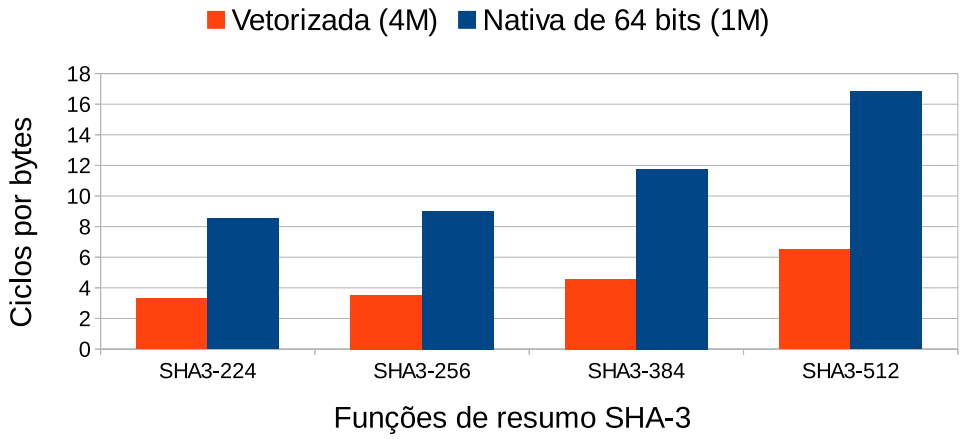
Teste 4. *Calcular os ciclos por byte para produzir o valor de resumo de mensagens com tamanho variando de 256 B até 512 MB.*

Na Figura 3.7b é apresentado os ciclos por byte necessários para calcular o valor de resumo de mensagens de tamanho 256 B a 512 MB usando a função SHA3-256.

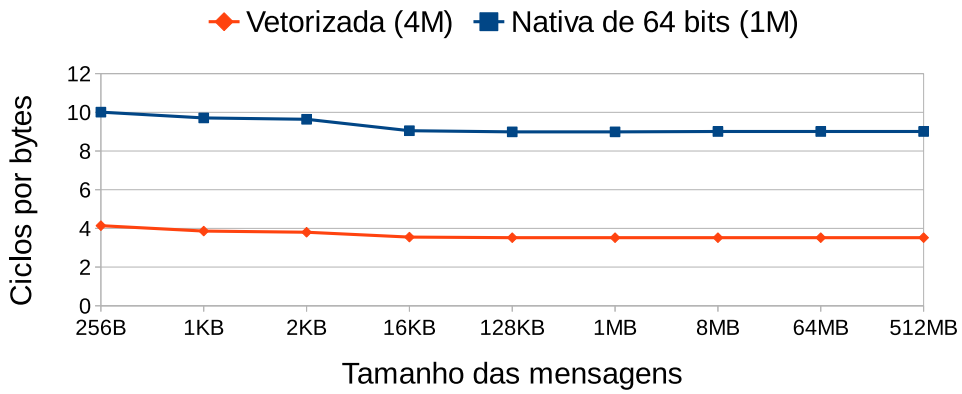
Observações: devido à importância do novo padrão de funções de resumo, é interessante dispor de técnicas de implementação que ajudem na aceleração da execução desses algoritmos, como as que foram apresentadas nesta seção.

3.4.3. Implementação do protocolo ECDH usando AVX2

A implementação do protocolo de acordo de chaves baseado em curvas elípticas envolve diversos aspectos a serem levados em conta: do ponto de vista de desempenho, como usar as instruções vetoriais na aceleração da execução do protocolo; do ponto



(a) Ciclos por byte para calcular o valor de resumo para as funções da família SHA-3.



(b) Ciclos por byte para calcular o valor de resumo de mensagens de tamanho 256 B a 512 MB.

Figura 3.7: Resultado dos testes de desempenho das implementações do algoritmo SHA-3.

de vista de segurança, quais contramedidas são usadas para evitar vazamento de informação sigilosa.

Atualmente o protocolo ECDH é implementado usando as diretrizes definidas pelo padrão SP 800-56A do NIST [BJS07] e utilizando as curvas elípticas recomendadas no padrão de assinaturas digitais definido em [KSD13]. Embora esse padrão ainda esteja em vigência; recentemente, surgiram inúmeras propostas de implementação do protocolo ECDH usando novos modelos de curvas elípticas. Essas curvas permitem acelerar a execução do protocolo sem afetar a segurança do esquema [BL15c].

Atualmente uma das propostas que tem ganhado muita atenção encontra-se disponível em um documento publicado pela comunidade *Internet Engineering Task Force* (IETF) [LHT15]; dita proposta consiste de duas curvas elípticas, conhecidas por *Curve25519* [Ber06] e *Goldilocks* [Ham15], que suportam os níveis de segurança

Algoritmo 5 Multiplicação de pontos mediante o algoritmo de Montgomery.

Entrada: k, um número inteiro de t bits.
$x(P)$, a coordenada x de um ponto $P \in E$.
Saída: $x(kP)$, a coordenada x do ponto kP.

1: Seja $(k_t = 0, k_{t-1}, \ldots, k_0)_2$ a representação binária de k.
2: $X_{Q_0-Q_1} \leftarrow x(P)$, $Z_{Q_0-Q_1} \leftarrow 1$
3: $X_{Q_0} \leftarrow x(P)$, $Z_{Q_0} \leftarrow 1$
4: $X_{Q_1} \leftarrow 1$, $Z_{Q_1} \leftarrow 0$
5: **for** $i \leftarrow t-1$ **to** 0 **do**
6: $\quad b \leftarrow k_i \oplus k_{i+1}$
7: $\quad X_{Q_1}, X_{Q_0} \leftarrow \texttt{CondSwap}(b, X_{Q_1}, X_{Q_0})$
8: $\quad Z_{Q_1}, Z_{Q_0} \leftarrow \texttt{CondSwap}(b, Z_{Q_1}, Z_{Q_0})$
9: $\quad X_{Q_1}, Z_{Q_1}, X_{Q_0}, Z_{Q_0} \leftarrow \texttt{Ladder}(X_{Q_0-Q_1}, Z_{Q_0-Q_1}, X_{Q_1}, Z_{Q_1}, X_{Q_0}, Z_{Q_0})$
10: **end for**
11: **return** $x(kP) \leftarrow X_{Q_0}(Z_{Q_0})^{-1}$

de 128 e 224 bits, respectivamente. Essas curvas permitem calcular a multiplicação de pontos com um menor número de operações aritméticas no corpo finito $\mathbb{F}_p$.

Nesta seção serão apresentadas a curva elíptica *Curve25519*, como implementar eficientemente a aritmética de pontos e a aritmética do corpo primo $\mathbb{F}_{2^{255}-19}$, onde as instruções vetoriais possuem um papel fundamental na aceleração do protocolo.

3.4.3.1. Descrição da curva elíptica Curve25519

A curva elíptica *Curve25519* denotada por E é definida pela seguinte equação:

$$E: \quad y^2 = x^3 + 486662x^2 + x \tag{12}$$

onde as coordenadas dos pontos pertencem ao corpo primo $\mathbb{F}_p$ sendo $p = 2^{255} - 19$.

Essa curva permite que a multiplicação de pontos seja feita usando apenas a coordenada x do ponto P; isto é, dado um inteiro k e a coordenada x do ponto P, existe um algoritmo que computa a coordenada x de ponto kP; esse algoritmo foi proposto por Montgomery [Mon87] e é apresentado no Algoritmo 5. O algoritmo utiliza as coordenadas projetivas, onde a coordenada x é representada por dois valores (X, Z) sendo $x = X/Z$. Para realizar o calculo de kP são usados dois pontos acumuladores Q_0 e Q_1. O primeiro contendo o valor de P e o segundo contendo $\mathcal{O}$. O conteúdo dos acumuladores será atualizado iterativamente em função do valor dos bits de k; assim sendo, se o i-ésimo bit de k for igual a 1, então o conteúdo dos acumuladores é trocado; feito isso, a função `Ladder` atualizará os acumuladores $Q_0 \leftarrow 2Q_0$ e $Q_1 \leftarrow Q_0 + Q_1$. Após percorrer todos os bits de k, o valor de kP se encontrará armazenado no acumulador $Q_0 = (X, Z)$, finalmente a coordenada x de kP é computada como $x = X/Z$.

A função `CondSwap` deve ser implementada em tempo constante, isto é, o

Algoritmo 6 Soma e duplicação de dois pontos na curva *Curve25519* (`Ladder`).

Entrada: $X_{P-Q}, Z_{P-Q}, X_P, Z_P, X_Q, Z_Q \in \mathbb{F}_p$, e seja $\hat{a}_2 = 486662$.
Saída: $X_{2P}, Z_{2P}, X_{P+Q}, Z_{P+Q} \in \mathbb{F}_p$.

1:	$A \leftarrow X_P + Z_P$	$C \leftarrow X_Q + Z_Q$	[soma]
2:	$B \leftarrow X_P - Z_P$	$D \leftarrow X_Q - Z_Q$	[subt]
3:	$DA \leftarrow A \times D$	$CB \leftarrow C \times B$	[mult]
4:	$t_1 \leftarrow DA + CB$	$t_0 \leftarrow DA - CB$	[soma/subt]
5:	$t_1 \leftarrow t_1^2$	$t_0 \leftarrow t_0^2$	[quad]
6:	$X_{P+Q} \leftarrow t_1 \times Z_{P-Q}$	$Z_{P+Q} \leftarrow t_0 \times X_{P-Q}$	[mult]
7:	$A' \leftarrow A^2$	$B' \leftarrow B^2$	[quad]
8:	$A'x \leftarrow \frac{1}{4}(\hat{a}_2 + 2) \cdot A'$	$B'y \leftarrow \frac{1}{4}(\hat{a}_2 - 2) \cdot B'$	[mult]
9:	$E \leftarrow A' - B'$	$F \leftarrow A'x - B'y$	[subt]
10:	$X_{2P} \leftarrow A' \times B'$	$Z_{2P} \leftarrow E \times F$	[mult]
11:	**return** $X_{2P}, Z_{2P}, X_{P+Q}, Z_{P+Q}$.		

mesmo processamento deve ser executado independente do valor do bit b, seguindo as recomendações apresentadas na Seção 3.3.

A função `Ladder` consiste na computação de $2Q_0$ e $Q_0 + Q_1$ sobre a curva *Curve25519*, onde as formulas da lei de grupo para esta curva estão listadas no Algoritmo 6, lembrando que as operações deste algoritmo são realizadas na aritmética do corpo primo $\mathbb{F}_{2^{255}-19}$. Esse algoritmo tem a propriedade de que algumas operações aritméticas podem ser computadas em paralelo; como podemos observar, cada linha processa o mesmo tipo de operação em ambas colunas. Esta característica nos permite um melhor aproveitamento das instruções vetoriais.

3.4.3.2. Aritmética do corpo primo

Um corpo primo $\mathbb{F}_p$ é uma estrutura algébrica que define duas operações binárias sobre um conjunto finito de números; o qual pode ser representado como o conjunto dos números inteiros no intervalo de 0 a $p-1$. A aritmética do corpo segue as mesma operações do conjunto dos inteiros, mas os resultados devem ser reduzidos modulo p para continuarem no intervalo determinado. O primo usado na curva *Curve25519* é $p = 2^{255} - 19$ e ele pertence ao conjunto dos primos *pseudo-Mersenne*, que são apropriados para fazer a redução modular de forma eficiente.

A aritmética do corpo primo opera sobre números de 256 bits. Usualmente, uma implementação que usa instruções nativas de 64 bits armazena um elemento do corpo em quatro palavras de 64 bits; algumas operações envolvem a possível propagação de resultados intermediários entre as palavras que compõem o elemento do corpo e esta propagação de dados introduz dependências na execução das instruções, implicando em uma implementação lenta.

Uma forma de minimizar a propagação de dados consiste em representar os elementos do corpo de modo que as operações possam ser computadas em paralelo, o que permitiria usar instruções vetoriais ao invés de instruções nativas de 64 bits.

Assim, para representar eficientemente um elemento $\alpha \in \mathbb{F}_{2^{255}-19}$ a tupla de coeficientes será utilizada; de agora em diante α será denotado por $\mathbf{A} = \{a_0, \ldots, a_9\}$, tal que a seguinte equação seja satisfeita:

$$\alpha = a_0 + a_1 2^{26} + a_2 2^{51} + a_3 2^{77} + a_4 2^{102} + a_5 2^{128} + a_6 2^{153} + a_7 2^{179} + a_8 2^{204} + a_9 2^{230}, \quad (13)$$

o número será representado por cinco palavras de 26 bits e cinco palavras de 25 bits.

A representação dos elementos também pode ser vista como um polinômio em x de grau 9 tal que $x^i = 2^{\lceil 25,5i \rceil}$. Por conta disso, as operações aritméticas seguem a mesma lógica das operações sobre polinômios como descrito a seguir:

- **Soma e subtração.** A soma e substração são realizadas coeficiente a coeficiente que podem ser calculadas usando instruções de 32 ou 64 bits, pois os bits restantes servirão para armazenar os bits de *carry*. Desta forma, é possível realizar uma sequencia de somas e substrações antes de precisar realizar a redução modular; isto reduz a propagação de dados e acrescenta o grau de paralelismo nas computações.

- **Multiplicação.** A multiplicação de elementos no corpo pode ser feita como a multiplicação de dois polinômios, produzindo um polinômio de grau 18. A fim de manter uma representação compacta, se aplica a redução modulo $2^{255} - 19$; ela consiste em reduzir os monômios de grau maior ou igual a 10 pela equivalência $x^{10} = 2^{255} \equiv 19$, portanto os monômios com fator x^{10} são multiplicados por 19 e adicionados com os monômios correspondentes. A distribuição dos produtos da multiplicação modular é mostrada na Figura 3.8, onde cada coluna lista os produtos a serem adicionados na potência correspondente. Nesta representação do corpo $\mathbb{F}_{2^{255}-19}$, quando i e j são ímpares existe um caso especial onde $(a_i x^i)(b_j x^j) = 2a_i b_j x^{i+j}$, por conta disso na Figura 3.8 alguns produtos aparecem multiplicados por 2.

- **Inverso multiplicativo.** O inverso multiplicativo é calculado usando a seguinte equivalência $a^{-1} \equiv a^{p-2} \equiv (\alpha_{250})^{2^5} a^{11}$. Parte da exponenciação pode ser eficientemente computada pelo algoritmo de Itoh-Tsujii [IT88] que calcula o termo $\alpha_x = a^{2^x - 1}$ mediante uma cadeia de adição. Assim, para obter α_{250} calcula-se $\alpha_5 \to \alpha_{10} \to \alpha_{20} \to \alpha_{40} \to \alpha_{50} \to \alpha_{100} \to \alpha_{200} \to \alpha_{250}$, onde a transição $\alpha_x \to \alpha_y$ encontra-se definida como $\alpha_y = (\alpha_x)^{2^{y-x}} \alpha_{y-x}$ para $x \leq y \in \mathbb{Z}^+$.

A representação apresentada permite que o calculo das operações aritméticas seja feito por uma série de operações independentes, tentando acrescentar o grau de paralelismo e agilizar a execução das operações aritméticas. Como foi visto, essas operações servem como blocos básicos para a implementação da aritmética de curvas elípticas.

3.4.3.3. Implementação vetorial da curva Curve25519

Nas seções anteriores foram mostrados algoritmos para a computação da multiplicação de pontos e uma representação dos elementos do corpo primo. Nos dois casos,

x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
a_9b_0	$38a_9b_9$	$19a_9b_8$	$38a_9b_7$	$19a_9b_6$	$38a_9b_5$	$19a_9b_4$	$38a_9b_3$	$19a_9b_2$	$38a_9b_1$
a_8b_1	a_8b_0	$19a_8b_9$	$19a_8b_8$	$19a_8b_7$	$19a_8b_6$	$19a_8b_5$	$19a_8b_4$	$19a_8b_3$	$19a_8b_2$
a_7b_2	$2a_7b_1$	a_7b_0	$38a_7b_9$	$19a_7b_8$	$38a_7b_7$	$19a_7b_6$	$38a_7b_5$	$19a_7b_4$	$38a_7b_3$
a_6b_3	a_6b_2	a_6b_1	a_6b_0	$19a_6b_9$	$19a_6b_8$	$19a_6b_7$	$19a_6b_6$	$19a_6b_5$	$19a_6b_4$
a_5b_4	$2a_5b_3$	a_5b_2	$2a_5b_1$	a_5b_0	$38a_5b_9$	$19a_5b_8$	$38a_5b_7$	$19a_5b_6$	$38a_5b_5$
a_4b_5	a_4b_4	a_4b_3	a_4b_2	a_4b_1	a_4b_0	$19a_4b_9$	$19a_4b_8$	$19a_4b_7$	$19a_4b_6$
a_3b_6	$2a_3b_5$	a_3b_4	$2a_3b_3$	a_3b_2	$2a_3b_1$	a_3b_0	$38a_3b_9$	$19a_3b_8$	$38a_3b_7$
a_2b_7	a_2b_6	a_2b_5	a_2b_4	a_2b_3	a_2b_2	a_2b_1	a_2b_0	$19a_2b_9$	$19a_2b_8$
a_1b_8	$2a_1b_7$	a_1b_6	$2a_1b_5$	a_1b_4	$2a_1b_3$	a_1b_2	$2a_1b_1$	a_1b_0	$38a_1b_9$
a_0b_9	a_0b_8	a_0b_7	a_0b_6	a_0b_5	a_0b_4	a_0b_3	a_0b_2	a_0b_1	a_0b_0
c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0

Figura 3.8: Forma compacta de arranjar os produtos intermediários na computação da multiplicação modular de dois elementos no corpo, $\mathbf{C} = \mathbf{A} \times \mathbf{B} \in \mathbb{F}_{2^{255}-19}$.

os algoritmos apresentavam um certo grau de paralelismo que pode ser explorado usando instruções vetoriais. A seguir será discutido como utilizar as instruções do conjunto AVX2 para implementar o protocolo.

Cada linha do Algoritmo 6 calcula duas operações aritméticas independentes, uma por coluna. Como foi discutido em [FL15], o raciocínio por trás deste processamento é reduzir o uso de instruções de alta latência, isto é, aquelas que movem dados entre a parte baixa e alta dos registradores de 256 bits. Partindo dessa observação, os primeiros 128 bits dos registradores `YMM` são usados para processar as operações da coluna esquerda e os bits restantes para processar as operações da coluna direita. Assim, os elementos do corpo são associados aos registradores vetoriais da seguinte maneira: denota-se por $\langle \mathbf{A}, \mathbf{B} \rangle$ as *tuplas entrelaçadas* de $\mathbf{A}$ e $\mathbf{B}$ (duas tuplas representando dois elementos do corpo), para representar cinco registradores `YMM` contendo:

$$\langle \mathbf{A}, \mathbf{B} \rangle_i = [a_{2i+1}, a_{2i}, b_{2i+1}, b_{2i}] \quad \text{para } 0 \leq i < 5. \tag{14}$$

assim, o registrador de 256 bits conterá dois coeficientes de duas tuplas armazenados em palavras de 64 bits. Contudo, a implementação das operações aritméticas ainda é beneficiada pelas instruções vetoriais, pois as operações podem usar vetores de 128 bits.

Uma das operações mais críticas em termos de desempenho é a multiplicação modular, pois requer calcular uma grande quantidade de multiplicações de coeficientes. O Algoritmo 7 apresenta a sequência de instruções para o processamento da multiplicação modular no corpo $\mathbb{F}_{2^{255}-19}$. Nas linhas 2-5 é inicializado um conjunto de registradores que contém alguns coeficientes de $\langle \mathbf{B}, \mathbf{D} \rangle$ multiplicados por 2; durante o ciclo principal, nas linhas 6-18, as variáveis Z_i acumularão o conteúdo dos produtos intermediários $a_i b_j$; e no último ciclo, nas linhas 19-22, a redução modular é calculada sobre os coeficientes de grau maior ou igual a 10.

A instrução `MUL` possui uma latência de 5 ciclos e ao fim de cada ciclo é possível começar a execução de uma nova multiplicação, fazendo com que a latência total de uma sequência de multiplicações independentes seja reduzida. Adicionalmente, enquanto uma multiplicação está sendo processada na porta 0 as outras portas de execução do processador ficam disponíveis para execução de outros tipos de instruções; por exemplo, pode-se fazer operações de acesso a memória concorrentemente às multiplicações.

A saída da função de multiplicação produz uma tupla entrelaçada onde cada palavra de 64 bits contém um coeficiente de aproximadamente 52 bits. Posteriormente, essa tupla pode ser usada para computar operações de soma ou substração. Entretanto, se duas multiplicações precisam ser computadas sucessivamente, o tamanho dos coeficientes devem ser reduzidos tal que sejam menor que 32 bits visto que a instrução `MUL` computa apenas produtos de palavras de 32 bits.

O processamento antes mencionado é denominado de *redução de coeficientes* e assegura que cada coeficiente seja de aproximadamente 26 bits. Neste processamento, cada coeficiente c_i é dividido em três partes (l_i, m_i e h_i), de modo que l_i contém os primeiros 26 bits, m_i os próximos 25 bits e h_i os bits remanescentes. Após a divisão, a redução dos coeficientes é calculada por:

$$\begin{aligned} c'_0 &= l_0 + 19(m_9 + h_8), \\ c'_1 &= l_1 + m_0 + 19h_9, \\ c'_i &= l_i + m_{i-1} + h_{i-2} \text{ para } 2 \leq i < 10. \end{aligned} \tag{15}$$

O Algoritmo 8 mostra o processamento realizado usando as instruções AVX2. No primeiro ciclo, nas linhas 1-6, é feita a divisão das palavras para calcular os vetores l, m e h. No segundo ciclo, nas linhas 7-9, o algoritmo arranja os coeficientes do vetor m, alinhando-os com l e h. Das linhas 10-15, são computadas as multiplicações por 19 dos termos m_9, h_8 e h_9; finalmente, o último ciclo calcula a soma dos vetores l, m e h. Cada coeficiente da tupla entrelaçada que é retornada tem um tamanho de aproximadamente 26 bits.

3.4.3.4. Testes de desempenho

O impacto causado pelas instruções AVX2 na computação do protocolo ECDH foi mostrada no trabalho de [FL15]. A Tabela 3.4 mostra uma comparação das implementações da curva elíptica *Curve25519*. Note que as implementações que usam instruções vetoriais obtiveram um melhor desempenho sobre as implementações com instruções nativas de 64 bits.

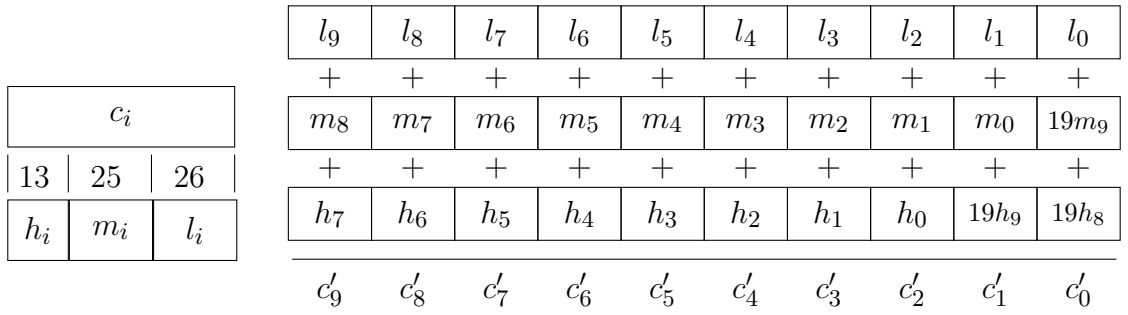
Para mostrar o desempenho obtido no protocolo ECDH usando a curva elíptica *Curve25519*, o seguinte teste será feito.

Teste 5. *Calcular a quantidade de operações de acordo de chaves calculadas por segundo para o conjunto de curvas suportado pela biblioteca OpenSSL e a curva elíptica Curve25519 usando AVX2.*

Algoritmo 7 Sequencia de instruções para calcular a multiplicação modular no corpo $\mathbb{F}_{2^{255}-19}$ usando as instruções AVX2.

Entrada: $\langle \mathbf{A}, \mathbf{C} \rangle$ e $\langle \mathbf{B}, \mathbf{D} \rangle$, duas tuplas entrelaçadas.
Saída: $\langle \mathbf{E}, \mathbf{F} \rangle$ uma tupla entrelaçada tal que $\mathbf{E} = \mathbf{A} \times \mathbf{B}$ e $\mathbf{F} = \mathbf{C} \times \mathbf{D}$.

1: $Z_i \leftarrow 0$ para $0 \leq i < 10$
2: **for** $i \leftarrow 0$ **to** 4 **do**
3: $\quad \langle \mathbf{B}', \mathbf{D}' \rangle_i \leftarrow \texttt{ALIGNR}(\langle \mathbf{B}, \mathbf{D} \rangle_{i+1 \bmod 5}, \langle \mathbf{B}, \mathbf{D} \rangle_i)$
4: $\quad \langle \mathbf{B}', \mathbf{D}' \rangle_i \leftarrow \texttt{SHLV}(\langle \mathbf{B}', \mathbf{D}' \rangle_i, [0, 1, 0, 1])$
5: **end for**
6: **for** $i \leftarrow 0$ **to** 4 **do**
7: $\quad U \leftarrow \texttt{SHUFFLE}(\langle \mathbf{A}, \mathbf{C} \rangle_i, 0)$
8: $\quad$ **for** $j \leftarrow 0$ **to** 4 **do**
9: $\quad\quad Z_{i+j} \leftarrow \texttt{ADD}(Z_{i+j}, \texttt{MUL}(U, \langle \mathbf{B}, \mathbf{D} \rangle_j))$
10: $\quad$ **end for**
11: $\quad V \leftarrow \texttt{SHUFFLE}(\langle \mathbf{A}, \mathbf{C} \rangle_i, 1)$
12: $\quad$ **for** $j \leftarrow 0$ **to** 3 **do**
13: $\quad\quad Z_{i+j+1} \leftarrow \texttt{ADD}(Z_{i+j+1}, \texttt{MUL}(V, \langle \mathbf{B}', \mathbf{D}' \rangle_j))$
14: $\quad$ **end for**
15: $\quad W \leftarrow \texttt{MUL}(V, \langle \mathbf{B}', \mathbf{D}' \rangle_4)$
16: $\quad Z_i \leftarrow \texttt{ADD}(Z_i, \texttt{BLEND}(W, [0, 0, 0, 0], 0101))$
17: $\quad Z_{i+5} \leftarrow \texttt{ADD}(Z_{i+5}, \texttt{BLEND}(W, [0, 0, 0, 0], 1010))$
18: **end for**
19: **for** $i \leftarrow 0$ **to** 4 **do**
20: $\quad 19Z_{i+5} \leftarrow \texttt{ADD}(\texttt{ADD}(\texttt{SHL}(Z_{i+5}, 4),\ \texttt{SHL}(Z_{i+5}, 1)), Z_{i+5})$
21: $\quad \langle \mathbf{E}, \mathbf{F} \rangle_i \leftarrow \texttt{ADD}(Z_i,\ 19Z_{i+5})$
22: **end for**
23: **return** $\langle \mathbf{E}, \mathbf{F} \rangle$



(a) Divisão do coeficiente c_i.

(b) Após da soma os coeficientes são de aproximadamente 26 bits.

Figura 3.9: Redução de coeficientes. Na esquerda, cada coeficiente c_i é divido em três partes chamadas l_i, m_i e h_i. Essas partes são adicionadas como é mostrado na figura da direita. Note que m_9, h_8 e h_9 são reduzidos modulo p.

Algoritmo 8 Sequencia de instruções para calcular a redução de coeficientes usando instruções AVX2.

Entrada: $\langle \mathbf{A}, \mathbf{B} \rangle$, uma tupla entrelaçada.
Saída: $\langle \mathbf{A}, \mathbf{B} \rangle$, tal que $|c_{2i}| \leq 27$ e $|c_{2i+1}| \leq 26$ para $0 \leq i < 5$ e $c \in \{a, b\}$.

for $i \leftarrow 0$ **to** 4 **do**
  $L_i \leftarrow \texttt{AND}(\langle \mathbf{A}, \mathbf{B} \rangle_i, [2^{25}-1, 2^{26}-1, 2^{25}-1, 2^{26}-1])$
  $M_i \leftarrow \texttt{SHRV}(\langle \mathbf{A}, \mathbf{B} \rangle_i, [25, 26, 25, 26])$
  $M_i \leftarrow \texttt{AND}(M_i, [2^{26}-1, 2^{25}-1, 2^{26}-1, 2^{25}-1])$
  $H_i \leftarrow \texttt{SHRV}(\langle \mathbf{A}, \mathbf{B} \rangle_i, [51, 51, 51, 51])$
end for
for $i \leftarrow 0$ **to** 4 **do**
  $M'_i \leftarrow \texttt{ALIGNR}(M_i, M_{i-1 \bmod 5})$
end for
$H_4 \leftarrow \texttt{SHRV}(\langle \mathbf{A}, \mathbf{B} \rangle_8, [51, 26, 51, 26])$
$U \leftarrow \texttt{ADD}(H_4, \texttt{SHR}(H_4, 64))$
$19U \leftarrow \texttt{ADD}(\texttt{ADD}(\texttt{SHR}(U, 4), \texttt{SHR}(U, 1)), U)$
$T \leftarrow \texttt{AND}(19U, [0, 2^{26}-1, 0, 2^{26}-1])$
$S \leftarrow \texttt{SHR}(19U, [0, 26, 0, 26])$
$H_4 \leftarrow \texttt{UPCK}(T, S)$
for $i \leftarrow 0$ **to** 4 **do**
  $\langle \mathbf{A}, \mathbf{B} \rangle_i \leftarrow \texttt{ADD}(\texttt{ADD}(L_i, M'_i), H_{i-1 \bmod 5})$
end for
return $\langle \mathbf{A}, \mathbf{B} \rangle$

Implementação	Processador	Instr. Vetoriais	Acordo de Chaves
NaCl [BLS13]	Core i7-4770	Não	261.000
amd64-51 [BL15b]	Core i7-4770	Não	163.200
amd64-51 [BL15b]	Xeon E3-1275 V3	Não	161.600
AVX [Cho15]	Core i5-3210M	Sim	159.100
AVX2 [FL15]	Core i7-4770 $^{(\alpha)}$	Sim	156.500

Tabela 3.4: Tempos de execução do protocolo de acordo de chaves usando a curva *Curve25519*; os tempos são reportados em ciclos.

Curva Elíptica	Operações/segundo
B-283	2.159
K-283	2.268
P-256	14.384
Curve25519	21.787

Tabela 3.5: Comparação do desempenho das curvas padrão do NIST contra a curva elíptica Curve25519.

Como pode ser visto na Tabela 3.5, as curvas binárias do NIST, B-283 e K-283, que estão implementadas na biblioteca do OpenSSL reportam uma taxa baixa de computações por segundo, atingindo na média dois mil operações por segundo. A implementação da curva P-256 é mais otimizada obtendo acima de quatorze mil operações por segundo; no entanto o desempenho oferecido pela curva *Curve25519* é 1.5 vezes mais rápido do que a curva NIST P-256.

Observações: A latência do protocolo ECDH pode ser reduzida com o uso de curvas específicas que reduzem o número de operações a serem calculadas; além disso, a aritmética do corpo primo foi adaptada para evitar propagação de dados, acrescentando o grau de paralelismo.

3.5. Comentários finais

Os algoritmos criptográficos garantem a segurança da informação através de meios de comunicação propensos a ataques. Algumas vezes, esses ataques são bem sucedidos devido à presença de implementações que não cumprem com os requisitos mínimos de segurança; por exemplo, a proteção contra ataques de canais laterais.

A implementação destes algoritmos precisa de um estudo meticuloso do fluxo da informação, dos acessos a memória, do tipo de instruções presentes na arquitetura e dos possíveis ataques. Portanto, a implementação de algoritmos criptográficos apresenta desafios em múltiplas dimensões.

A relevância que a segurança da informação tem ganhado nos últimos anos tem impactado no projeto dos processadores e arquiteturas atuais. Prova disso, é o suporte em *hardware* adicionado para o padrão de encriptação de dados AES, como foi visto nas sessões anteriores. Esse não é um caso isolado, pois a próxima arquitetura *Skylake* dará suporte à função de resumo SHA-2 [Cor13].

Este trabalho se aprofundou na eficiência das operações, visando aproveitar os recursos disponíveis nos processadores recentes. Atualmente existe uma tendência forte por explorar o paralelismo mediante o escalonamento de múltiplas instruções por ciclo, a inclusão de unidades de execução mais potentes, a divisão de tarefas, entre outras. Por conta disso é imprescindível encontrar técnicas que adaptem os algoritmos às arquiteturas modernas. Em particular, o conjunto de instruções vetoriais AVX2 se apresentou como uma alternativa bem sucedida para acelerar a execução de algoritmos criptográficos.

Consideramos que as técnicas aqui apresentadas podem ser dimensionadas para outros conjuntos de instruções vetoriais. Por exemplo, a arquitetura do co-processador Xeon Phi possui registradores de 512 bits, mais de 60 unidades de processamento e o novo conjunto de instruções vetoriais AVX-512 [Cor13]. Este tipo de arquitetura está voltado para o processamento de grandes quantidades de informação, o que a torna uma arquitetura ideal para a implementação de algoritmos criptográficos com alto grau de paralelismo.

Finalmente, é intenção dos autores incentivar ao leitor interessado nos tópicos aqui descritos a aprofundar mais seus conhecimentos na literatura recente. Caso mais informação seja necessária, sugerimos entrar em contato com a lista de autores do minicurso, pois alguns dos resultados aqui apresentados fazem parte de projetos de pesquisa em andamento.

A. Plataforma de teste de desempenho

Para os testes de desempenho realizados neste minicurso foram utilizados os computadores HW-DESKTOP e HW-ULTRA; ambos usam um processador *Haswell* da Intel. Eles são capazes de rodar instruções nativas de 64 bits, vetoriais de 128 e 256 bits e suportam os conjuntos de instruções AVX2 e AES-NI.

É importante mencionar que apesar de possuírem a mesma arquitetura, o computador HW-DESKTOP é um processador de propósito geral indicado para computadores de mesa, já o computador HW-ULTRA é um processador de baixo consumo energético, voltado para a computação móvel. Na Tabela 3.6 são mostradas as especificações técnicas de cada computador.

	HW-DESKTOP	HW-ULTRA
Processador	Core i7-4770	Core i5-4350U
Frequência	3.4 GHz	1.4 GHz
Memoria RAM	4 GB	4 GB
Sistema operacional	Fedora 18	Fedora 20

Tabela 3.6: Especificações técnicas dos computadores HW-DESKTOP e HW-ULTRA usados neste trabalho.

Referências

[AcKKS07] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting Secret Keys Via Branch Prediction. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2007.

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In *Advances in Cryptology–EUROCRYPT 2005*, pages 36–57. Springer, 2005.

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, May 2007.

[BDPA08] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *Advances in Cryptology – Eurocrypt 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. `http://sponge.noekeon.org/`.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. The keccak reference. *Submission to NIST (Round 3)*, 13, 2011.

[BDPV14] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sakura: A Flexible Coding for Tree Hashing. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, volume 8479 of *Lecture Notes in Computer Science*, pages 217–234. Springer International Publishing, 2014.

[Ber04] Daniel J. Bernstein. Cache-timing attacks on AES, 2004. URL: `http://cr.yp.to/papers.html#cachetiming`.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[Bih97] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption*, pages 260–272. Springer, 1997.

[BJS07] Elaine B. Barker, Don Johnson, and Miles E. Smid. SP 800-56A. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised). Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2007.

[BL15a] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems, September 2015.

[BL15b] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. Accessed on 20 March 2015, March 2015.

[BL15c] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to` accessed 20 March 2015, 2015.

[BLS13] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. NaCl: Networking and Cryptography library. `http://nacl.cr.yp.to/`, October 2013.

[BM06] Joseph Bonneau and Ilya Mironov. Cache-Collision Timing Attacks Against AES. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2006.

[BT11] Billy Bob Brumley and Nicola Tuveri. Remote Timing Attacks Are Still Practical. In Vijay Atluri and Claudia Díaz, editors, *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, volume 6879 of *Lecture Notes in Computer Science*, pages 355–371. Springer, 2011.

[Cho15] Tung Chou. Fastest Curve 25519 Implementation Ever. In National Institute of Standards and Technology, editors, *Workshop on Elliptic Curve Cryptography Standard*. National Institute of Standards and Technology, June 2015.

[CL14] Roberto Cabral and Julio López. Software implementation of SHA-3 family using AVX2. In *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, volume XIV, pages 330–333. Sociedade Brasileira de Computação, 2014.

[Cor] Intel Corporation. Hardware Design Site Archives. Intel® Pentium processor with MMX™ technology documentation. `http://www.intel.com/design/archives/Processors/mmx/`.

[Cor11] Intel Corporation. Intel® Advanced Vector Extensions Programming Reference, June 2011. Disponível em `https://software.intel.com/sites/default/files/m/f/7/c/36945`.

[Cor13] Intel Corporation. Intel Instruction Set Architecture Extensions. Available at `https://software.intel.com/en-us/intel-isa-extensions`, July 2013.

[Cor14] Intel Corporation. Intel® Intrinsics Guide. `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`, February 2014.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, Nov 1976.

[DR01] Joan Daemen and Vincent Rijmen. Algorithm Alley: Rijndael: The Advanced Encryption Standard. *Dr. Dobb's Journal of Software Tools*, 26(3):137–139, March 2001.

[FL15] Armando Faz-Hernández and Julio López. Fast Implementation of Curve25519 Using AVX2. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 329–345. Springer, 2015.

[Fly72] M. Flynn. Some Computer Organizations and Their Effectiveness. *Computers, IEEE Transactions on*, C-21(9):948–960, Sept 1972.

[Fog14] Agner Fog. *Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*, December 2014.

[GGF+15] Vinodh Gopal, Sean Gulley, Wajdi Feghali, Ilya Albrekht, and Dan Zimmerman. Improving OpenSSL Performance. Technical report, Intel Corporation, May 2015. Disponível em `https://software.intel.com/en-us/articles/improving-openssl-performance`.

[Gue10] Shay Gueron. Intel advanced encryption standard (AES) instructions set. *Intel White Paper, Rev*, 3, 2010.

[Ham09] Mike Hamburg. Accelerating AES with Vector Permute Instructions. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2009.

[Ham15] Mike Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/`.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.

[IMPR09] Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Christian Rechberger. Collisions and other non-random properties for step-reduced

SHA-256. In *Selected Areas in Cryptography*, pages 276–293. Springer, 2009.

[IT88] Toshiya Itoh and Shigeo Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. *Inf. Comput.*, 78(3):171–177, September 1988.

[jCPB+12] Shu jen Chang, Ray Perlner, William E Burr, Meltem Sönmez Turan, John M Kelsey, Souradyuti Paul, and Lawrence E Bassham. *Third-round report of the SHA-3 cryptographic hash algorithm competition.* US Department of Commerce, National Institute of Standards and Technology, 2012.

[Koc96] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *16th Annual International Cryptology Conference (CRYPTO 1996)*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[KS09] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant AES-GCM. *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 1–17, 2009.

[KSD13] Cameron F. Kerry, Acting Secretary, and Charles Romine Director. FIPS PUB 186-4 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS), 2013.

[LHT15] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security, September 2015. Disponível em `https://datatracker.ietf.org/doc/draft-irtf-cfrg-curves/`.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, 1979.

[Mer88] RalphC. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 1988.

[MN07] Mitsuru Matsui and Junko Nakajima. On the power of bitslice implementation on Intel Core2 processor. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 121–134, 2007.

[Mon87] Peter L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[Nat93] National Institute of Standards and Technology. *FIPS PUB 180: Secure Hash Standard.* National Institute for Standards and Technology, Gaithersburg, MD, USA, May 1993.

[Nat95] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard*. National Institute for Standards and Technology, Gaithersburg, MD, USA, April 1995. Supersedes FIPS PUB 180 1993 May 11.

[Nat01a] National Institute of Standards and Technology. *FIPS PUB 197, ADVANCED ENCRYPTION STANDARD (AES)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, November 2001.

[Nat01b] National Institute of Standards and Technology. *NIST Special Publication 800-38A. Recommendation for Block Cipher Modes of Operation*. National Institute for Standards and Technology, Gaithersburg, MD, USA, December 2001.

[Nat02] National Institute of Standards and Technology. *FIPS PUB 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2*. National Institute for Standards and Technology, Gaithersburg, MD, USA, August 2002. Supersedes FIPS PUB 180-1 1995 April.

[Nat08a] National Institute of Standards and Technology. *FIPS PUB 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 2008. Supersedes FIPS PUB 180-2 2002 August.

[Nat08b] National Institute of Standards and Technology. *FIPS PUB 180-4, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-4*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 2008. Supersedes FIPS PUB 180-3 October 2008.

[Nat15] National Institute of Standards and Technology. *FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. National Institute for Standards and Technology, Gaithersburg, MD, USA, August 2015.

[Per05] Colin Percival. Cache missing for fun and profit. In *Proceedings of BSDCan 2005*, 2005.

[Rab78] Michael O Rabin. Digitalized signatures. *Foundations of secure computation*, 78:155–166, 1978.

[RO05] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In *Topics in Cryptology–CT-RSA 2005*, pages 58–71. Springer, 2005.

[Rot11] Jeffrey Rott. Intel AESNI Sample Library. Technical report, Intel Corporation, May 2011. Disponível em `https://software.intel.com/en-us/articles/download-the-intel-aesni-sample-library`.

[RSD06] Chester Rebeiro, David Selvakumar, and A Devi. Bitslice implementation of AES. *Cryptology and Network Security*, pages 203–212, 2006.

[Sch95] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[Sta10] François-Xavier Standaert. Introduction to side-channel attacks. In *Secure Integrated Circuits and Systems*, pages 27–42. Springer, 2010.

[The03] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/-TLS. `www.openssl.org`, April 2003.

[TOS10] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptology*, 23(1):37–71, 2010.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology–CRYPTO 2005*, pages 17–36. Springer, 2005.

[Yee15] Alexander Yee. FeatureDetector, April 2015. Disponível em `https://github.com/Mysticial/FeatureDetector`.

[Yuv78] G. Yuval. *How to Swindle Rabin*. Rapport nr. IR. Vrije Universiteit, Wiskundig Seminarium, 1978.

Capítulo 4

Abordagens para Detecção de Spam de E-mail

Cleber K. Olivo, Altair O. Santin e Luiz Eduardo S. Oliveira

Abstract

The e-mail, one of the oldest and widely used services on the Internet, is the more used tool to send an indiscriminate number of unsolicited message, known as spam. Given the wide variety of techniques used for sending spam, this type of e-mail is a problem still far from being solved. This work aims to present works and techniques relating of spam detection from a new perspective. Instead of classifying the works by type of detection technique, as is usually made in the literature, each one will be organized using the technique applied in spam dissemination as entry key. Then, it will addressed the detection techniques for each case and it will make consideration about its efficiency.

Resumo

O e-mail, um dos serviços mais antigos e mais utilizados na Internet, é o meio mais utilizado para o envio indiscriminado de mensagens não solicitadas, conhecidas como spam. Devido à grande variedade de técnicas utilizadas para o envio de spam, esse tipo de e-mail é um problema que ainda está longe de ser solucionado. Este trabalho tem como objetivo apresentar as principais técnicas e trabalhos relacionados a detecção de spam sob uma nova perspectiva. Ao invés de classificar os trabalhos pelo tipo de técnica de detecção do spam, como normalmente é feito na literatura, as abordagens serão organizadas a partir da técnica utilizada na disseminação do spam. Então, serão abordadas as técnicas de detecção para cada caso e feitas considerações acerca de sua eficiência.

4.1. Introdução

4.1.1 Spam

O SMTP (*Simple Mail Transfer Protocol*) é o protocolo padrão utilizado para transferência de e-mails [1]. Nas últimas décadas, o e-mail tem sido um dos serviços mais utilizados na Internet, sendo o primeiro da lista quando o assunto é comunicação entre usuários na rede mundial. Por ser um dos serviços mais antigos e mais utilizados na Internet, o e-mail tornou-se o favorito para o envio de mensagens de marketing e, em alguns casos, até mesmo para o envio de mensagens fraudulentas ou contendo códigos maliciosos anexados. O envio indiscriminado de e-mails sem o consentimento de seus destinatários é conhecido como *spam.* O envio de *spam* pela Internet causa vários problemas, desde o aborrecimento dos usuários - por receberem mensagens indesejadas, até problemas de

sobrecarga dos servidores SMTP - devido ao grande volume de *spam* recebido. Em alguns casos mais graves, como o *phishing* (considerado uma subcategoria de *spam*), o dano causado pode ir muito além de um simples aborrecimento, levando o usuário a ter a segurança de seus computadores comprometida ou a ter prejuízos financeiros [2].

O *spam*, na sua forma virtual, é um problema cada vez mais presente na vida dos usuários e administradores de sistemas. O primeiro envio de *spam* por e-mail ocorreu em 1978, quando foi enviado para 393 usuários da ARPANET [3]. Desde então, as estatísticas sobre o envio de *spam* apresentam-se cada vez piores. Em 1997, a AOL estimou que de 5 a 30% de seus 10 milhões de e-mails recebidos por dia eram *spam* [4]. Em 2004, um estudo apresentou uma previsão de que esse percentual chegará em 95%, numa escala global, até 2015 [5]. Mais recentemente, estatísticas divulgadas pela Symantec revelaram que um percentual de 89,1% já foi atingido em 2010 [6], sendo o maior percentual até então [7]. Para se ter uma ideia, naquele ano a quantidade de e-mail enviado por dia foi de 62 bilhões. Nos últimos anos o volume que esses percentuais representam vem diminuindo para 60% no ano de 2013, para um volume global de spam estimado em 29 bilhões de mensagens por dia [8].

O *spamming* (envio de *spam*) vai muito além do aborrecimento do usuário. Enquanto o custo computacional para envio do *spam* é relativamente baixo, os provedores de serviços na Internet e seus usuários tem um custo alto, causado pelo desperdício de banda e pelos custos das tecnologias empregadas para a sua detecção [3].

Os *spammers* (termo utilizado para definir quem envia *spam),* a fim de dificultar a detecção de suas mensagens, fazem uso de subterfúgios técnicos que vão desde a sua forma de envio até informações inseridas propositalmente no corpo da mensagem, com o objetivo de confundir os mecanismos de detecção de spam (e.g. filtros *antispam*). O SPF (*Sender Policy Framework*), concebido para ser uma técnica *antispam*, embora se mostrasse promissor no começo, acabou sendo pouco eficiente, visto que os próprios *spammers* começaram a publicar seus registros SPF [9]. Isso mostra que protocolos de autenticação de e-mail por si só não são suficientes, tendo mais utilidade em casos de *phishing* ou e-mail *spoofing*. Um outro exemplo de técnica para burlar os mecanismos de detecção é o uso de imagens no envio das mensagens de e-mail. Isto aconteceu porque a eficiência dos mecanismos *antispam* na forma textual aumentou e então, em 2006, os *spammers* começaram a converter as mensagens em imagem [10]. De acordo com a *Ironport*, naquele mesmo ano, a quantidade de *spam* de imagem quadruplicou, representando entre 25% e 45% de todo *spam,* em alguns dias [11].

As técnicas para driblar os mecanismos *antispam* também são utilizadas para tornar ineficazes as técnicas baseadas em reconhecimento de texto. Palavras como 'viagra', por exemplo, podem se apresentar de diversas formas (e.g. V1AGR4, v.i.a.g.r.a etc). Um estudo revelou que essa mesma palavra pode ter mais de um quintilhão de variações [12], sendo praticamente impossível que os mecanismos de detecção baseados em análise de texto tenham conhecimento de todas as variações possíveis de uma única palavra. O problema atinge uma escala muito maior se for considerado que há várias palavras que podem produzir estes números significativos de variações num mesmo texto.

A criação de novas técnicas de detecção de *spam* levou os *spammers* a criarem novas técnicas de disseminação, geralmente baseadas na alteração constante do conteúdo da mensagem. De um modo geral, algumas das técnicas de detecção de *spam* existentes, envolvendo a área de reconhecimento de padrões, conseguem taxas razoáveis de classificação correta de e-mails. Entretanto, devido à grande variedade de técnicas utilizadas para o envio de *spam,* haverá detalhes específicos na mensagem que dificultarão a sua correta identificação. Assim, o *spam* é um problema que ainda está longe de ser solucionado,

visto que não existe uma técnica de detecção que seja eficiente para todas as técnicas de disseminação existentes.

Muitas ferramentas para classificação de e-mails, assim como a maioria das ferramentas de navegador, utilizam listas de remetentes "bons" (*whitelists*) e "maus" (*blacklists*). Normalmente, as *blacklists* bloqueiam o endereço IP do servidor de e-mail de origem das mensagens *spam*, ou ainda o próprio endereço de e-mail do remetente. O bloqueio do endereço IP ou domínio pode causar problemas quando o remetente utiliza o servidor de SMTP de algum provedor (e.g Yahoo, Gmail, etc), pois acaba por bloquear todos os remetentes que o utilizam. Já o bloqueio do e-mail do remetente pode ser ineficiente, visto que o mesmo pode ter sido forjado, sequestrado ou roubado de um usuário legítimo [13].

Algumas abordagens sugerem, também, a colaboração dos usuários para auxiliar o mecanismo *antispam* a classificar mensagens [14, 15]. Isto pode confundir o mecanismo de classificação de e-mails, pois um usuário pode considerar algo como *spam,* quando na verdade apenas não gosta de receber aquele tipo de e-mail por uma questão pessoal, enquanto outros usuários gostariam de recebê-lo.

As ferramentas baseadas em aprendizagem de máquina, em sua maioria acabam falhando quando um novo tipo de *spam* é recebido, visto que um novo modelo precisa ser treinado para que o novo *spam* possa ser detectado. Ou seja, até que o classificador seja treinado novamente, este novo *spam* já atingiu vários usuários. Além disto, o uso de imagens na mensagem, como comentado anteriormente, se tornou muito comum, então esta técnica passou a ser uma das mais exploradas, de acordo com a literatura [16, 17, 18, 19, 20, 21] para burlar os mecanismos *antispam*.

Uma técnica de disseminação de *spam* baseada no conteúdo da mensagem (mensagem textual), e que se tornou um problema para os classificadores baseados na ocorrência de palavras, é o ofuscamento de palavras ou caracteres. A quantidade de caracteres que podem ser trocados, visando confundir os mecanismos *antispam*, aumenta drasticamente a quantidade de palavras que podem substituir os termos originais que são comuns em mensagens de spam [12], pois para essas técnicas de classificação, se um único caractere é trocado em uma palavra, já não é computacionalmente considerado a mesma *string* de caracteres.

Este capítulo tem como objetivo apresentar as principais técnicas e trabalhos relacionados à detecção de *spam*. Porém, ao invés de classificar os trabalhos existentes pela técnica de detecção/classificação utilizada, como normalmente é feito na literatura, as propostas serão classificadas de acordo com a técnica utilizada na disseminação de *spam.* Ou seja, para cada artimanha utilizada pelos *spammers* para driblar os mecanismos de detecção, são apresentadas as soluções propostas na literatura para mitigar o problema.

Um dos problemas de apresentar as técnicas de detecção de *spam* na forma tradicional (classificando pela técnica de detecção ao invés da técnica de envio) é que, para quem não conhece o assunto a fundo, esta abordagem não revela os problemas relacionados à detecção de *spam* que devem ser resolvidos. Após a apresentação dos principais tipos de *spam*, classificando a partir das técnicas utilizadas, são apresentadas as principais técnicas de detecção propostas na literatura. Deste modo, é mais fácil entender e questionar a eficiência das técnicas de detecção para cada tipo de abordagem utilizada na disseminação das mensagens de *spam*.

4.1.2 Organização

A seção 4.2 apresenta alguns conceitos básicos sobre o funcionamento do serviço de e-mail. O objetivo dessa seção é entender como funciona o envio do e-mail, alguns aspectos importantes em relação ao protocolo SMTP (*Simple Mail Transfer Protocol*) e os principais campos da mensagem utilizados na comunicação entre servidores, aplicativos e usuários.

A seção 4.3 apresenta as principais técnicas utilizadas para o envio de *spam*. Diferente da maioria dos trabalhos, os quais classificam as abordagens a partir das técnicas de detecção propostas [4, 22, 23, 24, 25, 26], esta proposta apresenta uma visão diferenciada, que trata das técnicas de detecção sob o ponto de vista de cada técnica de envio de *spam*. Ou seja, a partir das principais técnicas de envio de *spam* são apresentadas as técnicas de detecção mais indicadas. Essa visão facilita o entendimento do problema, identificando a técnica de disseminação de *spam* e a abordagem de classificação de e-mails mais apropriada. Desse modo, se um administrador de sistemas estiver com problemas no recebimento de *spam* que utilize uma técnica específica (e.g *spam* de imagem), poderá conhecer as principais técnicas para a detecção deste tipo específico de *spam*.

Após a apresentação dos principais tipos de *spam*, ao apresentar cada técnica de detecção ficará mais fácil questionar a sua eficiência para um ou outro tipo de mensagem. Havendo o entendimento dessas técnicas de disseminação, é possível seguir para a próxima etapa (seção 4.4), onde são discutidas as principais técnicas de detecção de *spam* encontradas na literatura.

Na seção 4.5 é apresentada uma relação entre os tipos de *spam* apresentados na seção 4.3 e as técnicas de detecção apresentadas seção 4.4. A partir deste ponto, será possível inferir que nenhuma das técnicas isoladamente é capaz de conseguir ótimos resultados, sendo necessária uma combinação das mesmas para obter um mecanismo (e.g. filtro ou classificador) robusto e eficiente. Para cada tipo de *spam*, será apresentada a técnica mais recomendada. Adicionalmente, serão apresentadas as principais limitações de cada técnica e algumas alternativas utilizadas.

A última seção (4.6) apresenta as considerações finais acerca do assunto, possibilitando uma visibilidade maior sobre a complexidade do problema que o *spam* representa, incluindo os esforços mais recentes em pesquisa que buscam mitigar suas causas.

4.2. O Serviço de E-mail

O envio e recebimento de e-mails envolve dois componentes principais: o MTA (*Mail Transfer Agent*) e o MUA (*Mail User Agent*). O MTA (e.g. Postfix [27], qmail [28], Exchange [29], etc.) é o servidor responsável pelo envio e recebimento dos e-mails. Em um envio de e-mail através da Internet, na origem da mensagem, o MUA (e.g. Thunderbird [30], Microsoft Outlook [31], etc.) tem a função de coletar o e-mail do remetente e encaminhar a mensagem, através do protocolo SMTP, a um MTA de origem, que é o servidor encarregado de encaminhar a mensagem ao MTA de destino. O MTA de destino é encarregado pela entrega do e-mail ao MUA do destinatário através de algum serviço de entrega (e.g. IMAP [32] ou POP3 [33]). A Figura 4.1 ilustra um cenário onde há troca de mensagens entre dois MTAs.

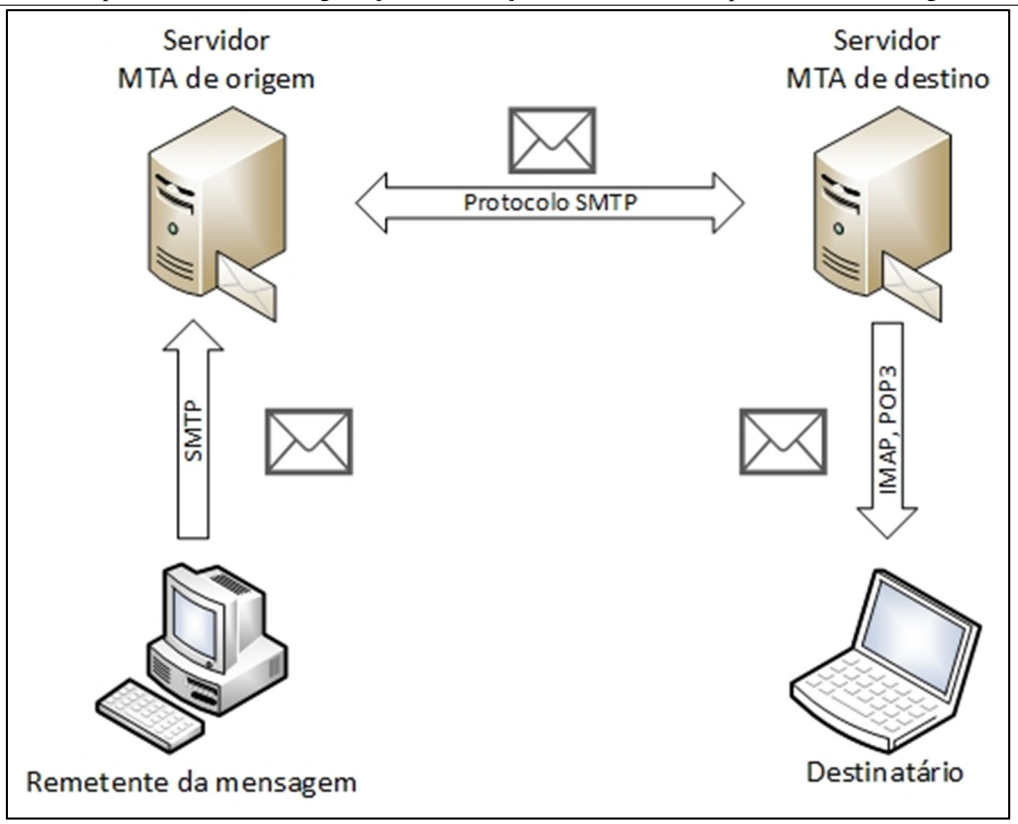


Figura 4.1. Troca de mensagens entre dois MTAs.

As regras para a troca de mensagens entre MTAs estão definidas na RFC 2821 (*Simple Mail Transfer Protocol*), que especifica que o e-mail é composto por um envelope (em inglês, *envelope)* e por um conteúdo *(content)* [1]. O envelope é composto por um endereço de origem (para onde os relatórios de erro são direcionados), um ou mais endereços de destino (destinatários) e outras informações opcionais do protocolo. O conteúdo é dividido em duas partes: cabeçalho (*header*) e o corpo *(body)*.

O cabeçalho sempre precede o corpo do e-mail, e contém informações obrigatórias, tais como os campos FROM (e-mail do remetente), TO (endereço do destinatário) e DATE (data), e outras informações opcionais, mas que geralmente são utilizadas, tais como os campos SUBJECT (assunto) e CC (do inglês, *carbon copy* – demais destinatários copiados no e-mail).

No que diz respeito ao *spam*, o próprio protocolo SMTP, por questões de projeto, possui limitações que são exploradas pelos *spammers* no envio de suas mensagens, possibilitando, por exemplo, que o endereço no campo remetente seja substituído por outro diferente do e-mail de quem está enviando a mensagem. Essas limitações são apresentadas com mais detalhes na seção 4.3.1.

O corpo contém a mensagem do e-mail, que pode ser composta por imagens e por texto com uso de recursos de hipertexto, como o HTML (*HyperText Markup Language*) - que serão interpretados pelo MUA. Embora o cabeçalho do e-mail seja essencialmente codificado no formato US-ASCII (*American Standard Code for Information Interchange*), o corpo do e-mail é estruturado conforme o formato MIME – *Multipurpose Internet Mail Extensions* [34].

A definição do MIME foi necessária pois o padrão que antecedia a RFC 2821 (RFC 822 – *Standard for the Format of ARPA Internet Text Messages*, 1982) [35], tinha a intenção de especificar apenas um formato para mensagens de texto. Mensagens em

outros formatos, tais como mensagens multimídia, que podem incluir áudio e vídeo, não foram mencionadas no padrão. Além disso, o padrão especificado pela RFC 822 é inadequado para as necessidades atuais dos usuários de e-mail, os quais utilizam idiomas que necessitam de um conjunto de caracteres mais amplo que o US-ASCII, como os caracteres de idiomas asiáticos e europeus [34]. O MIME redefine o formato das mensagens para permitir:

(1) Texto do corpo da mensagem utilizando conjuntos de caracteres diferentes do US-ASCII;

(2) Um conjunto extensível de diferentes formatos para corpos de texto em formato não-textual;

(3) Corpos de mensagem *multi-part*, ou seja, divididos em duas ou mais partes, cada uma com conjuntos de caracteres diferentes;

(4) Informação textual do cabeçalho em um conjunto de caracteres diferente do US-ASCII.

A possibilidade de uso de diversos conjuntos de caracteres, somada ao uso de imagens e recursos de hipertexto (HTML), permitiu que o corpo do e-mail se tornasse o campo da mensagem onde há o maior número de subterfúgios técnicos que podem ser utilizados para burlar os mecanismos a*ntispam*.

A seção 4.3 apresenta em detalhes as principais formas como o protocolo SMTP, o corpo do e-mail e o cabeçalho são utilizados na disseminação de *spam*, buscando a máxima eficiência, seja com o objetivo de alcançar o maior número de destinatários no menor tempo possível ou, ainda, driblando os mecanismos *antispam*.

4.3. Principais Tipos de Spam com Base na Técnica de Disseminação Utilizada

As técnicas utilizadas para a disseminação de *spam* sofreram mudanças significativas nos últimos anos. A criação de técnicas mais eficientes para a detecção de *spam* obrigou os *spammers* a criarem novas artimanhas para driblar os mecanismos de detecção. As técnicas de *spam* podem ser classificadas em duas categorias principais: i) técnicas baseadas no envio do *spam* e ii) técnicas baseadas no conteúdo do e-mail.

As técnicas baseadas no envio do *spam* correspondem à forma como o e-mail é enviado (e.g. através de uma *botnet* ou MTA legítimo). Já as técnicas baseadas no conteúdo normalmente estão associadas às artimanhas utilizadas no corpo do e-mail, para confundir os mecanismos de detecção baseados no conteúdo (textual ou não) da mensagem. Normalmente, o *spammer* utiliza uma combinação de técnicas presentes nas duas categorias (ou ainda várias técnicas da mesma categoria), maximizando as chances da mensagem passar pelos mecansimos *antispam* sem ser detectada. A possibilidade de inúmeras combinações dessas técnicas aumenta substancialmente o desafio de quem faz a classificação das mensagens (*spam* ou *não-spam*), i.e., os administradores de servidores de e-mail, que configuram os mecanismos *antispam* ou os pesquisadores que desenvolvem novas técnicas de classificação de mensagens.

A seguir será mostrado que cada uma dessas categorias principais pode abrigar várias subcategorias, como será apresentado nas próximas seções (4.3.1 – 4.3.3).

4.3.1. Técnicas Baseadas no Envio de Spam

As técnicas baseadas no envio de spam estão relacionadas com a forma como as mensagens são encaminhadas para os endereços de e-mail dos usuários. Ou seja, essas técnicas estão mais associadas com o "disparo" dos e-mails do que com o conteúdo da mensagem. Por exemplo, um *spam* pode ser encaminhado com ou sem a opção de verificação do recebimento da mensagem e fazer o reenvio em alguns casos de falha na entrega. Um *spam* poderia, também, ser encaminhado por um MTA que pertence a uma organização confiável - para fins publicitários, ou por uma origem fraudulenta, através de um mecanismo simples de envio – e.g. através de um *malware* (código malicioso desenvolvido para fins ilícitos e sem o consentimento do usuário).

Mais exemplos de técnicas baseadas no envio do *spam* e suas características são apresentadas nos itens de *a* até *e*.

a) Mecanismo simples de envio

Uma das formas típicas de enviar *spam* é através de um *mecanismo simples de envio* massivo de e-mails. Geralmente, este tipo de técnica é utilizado quando o objetivo é atingir o máximo de destinatários no menor tempo possível, sendo feito o envio de milhares de e-mails (ou mesmo dezenas ou centenas de milhares) sem se preocupar com erros de envio e confirmações de entrega das mensagens.

Em um MTA devidamente configurado, normalmente é verificado o recebimento da mensagem pelo MTA de destino. Além de problemas de rede, pode ocorrer, inclusive, verificações relacionadas à conta de e-mail do destinatário, retornando uma mensagem de erro caso a conta de usuário não exista ou esteja com a cota em disco esgotada. Em alguns casos de erro na entrega, a mensagem pode ser recolocada em uma fila para uma posterior tentativa de reenvio do e-mail. Esses controles e verificações que, entre vários propósitos, têm por objetivo a redução de erros de comunicação e maior eficiência do serviço, aumentam o custo computacional, o tempo de envio dos e-mails, e a complexidade dos softwares envolvidos no processo de envio massivo de mensagens.

Com a ausência de qualquer tipo de controle ou verificação na transmissão da mensagem (Figura 4.2), o mecanismo de envio massivo de e-mails torna-se muito mais simples, reduzindo consideravelmente o custo computacional, a complexidade do software e o tempo necessários para o envio do *spam*. O envio através de um *mecanismo simples* pode ser realizado através de um software com poucas linhas de código (inclusive algum *malware*), não sendo necessário a configuração de um servidor MTA completo. Neste caso não é feita uma nova tentativa de envio, ou seja, o servidor de origem (*spammer*) não precisa tratar os e-mails que retornam com erro, simplificando o sistema de envio do atacante.

A Figura 4.2 ilustra um exemplo no qual o *spammer* utiliza um computador pessoal para enviar *spam* para uma conta de e-mail que não existe naquele domínio (evento 1). O servidor SMTP ao receber a mensagem e constatar que não existe uma caixa de entrada para aquele endereço (usuario_inexistente@servidor_smtp.com), fornece uma mensagem de erro para o endereço do *spammer* (evento 2). A RFC 3463 mostra os códigos de estado para o serviço de e-mail [36]. O computador do *spammer*, por estar configurado como um mecanismo simples de envio de e-mails, não está preparado para tratar os e-mails de retorno, descartando a mensagem de erro (evento 3).

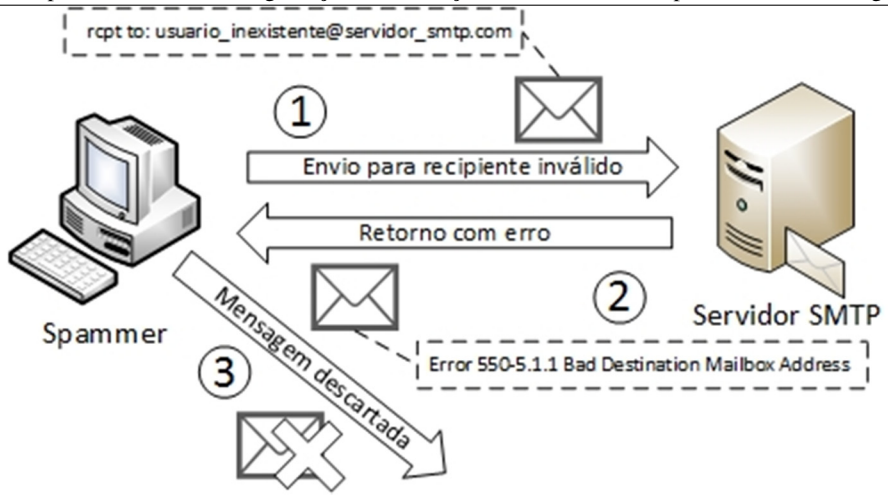


Figura 4.2. Envio através de mecanismo simples, sem tratamento de erros.

Em um *gateway* SMTP (ambiente de entrada e saída de e-mails onde pode haver um ou mais MTAs configurados) com grande volume de mensagens, é possível notar uma grande quantidade de e-mails destinados a endereços inexistentes (contas de usuário desativadas ou que nunca existiram). Isto ocorre porque os *spammers* muitas vezes se utilizam de listas de e-mails que possuem uma grande quantidade desses endereços inexistentes. Como não há tratamento desses e-mails com destinatários inválidos, o único tempo desperdiçado pelo *spammer*, nesses casos, é o do próprio envio da mensagem.

b) Envio com tratamento de erros

Neste caso, diferente do *mecanismo simples de envio*, os e-mails que retornam com erro são recolocados na fila de envio do MTA e são programados para serem retransmitidos mais tarde (Figura 4.3). Por exemplo, se ocorrer um erro temporário de rede ou no MTA de destino, o *spam* retorna ao MTA de origem e é reenviado mais tarde. O número de tentativas de reenvio e o tempo de espera para a retransmissão da mensagem é variável, podendo ser configurado no MTA de origem. Este tipo de envio se tornou mais comum à medida que foram criadas técnicas para a detecção de mensagens de *spam* enviadas através de *mecanismos simples de envio*, sendo também muito comum em servidores de e-mail que enviam *spam* com fins publicitários.

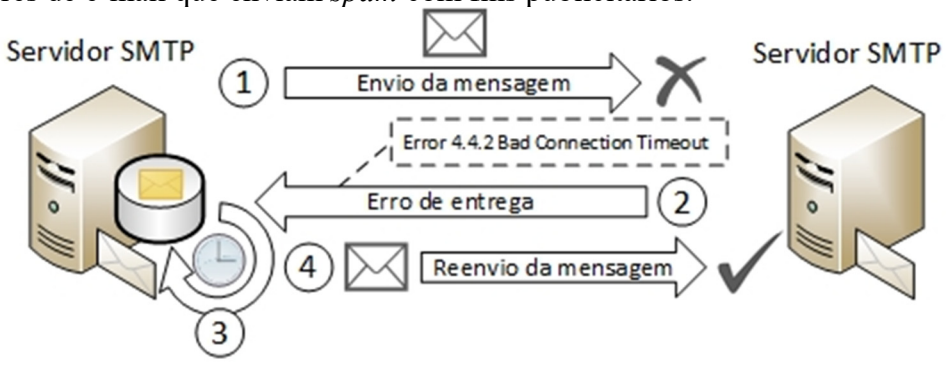


Figura 4.3. Tratamento de erros entre MTAs completos.

Na Figura 4.3, o servidor SMTP da origem da mensagem encaminha o e-mail para o servidor SMTP de destino e, por alguma razão (e.g. problemas de rede ou no servidor de destino) não consegue concluir a transferência do e-mail (evento 1). Ao constatar o erro na entrega do e-mail (evento 2), a mensagem é colocada em uma fila (evento 3) para posterior tentativa de reenvio. O tempo que a mensagem permanece na fila e o seu

descarte após várias tentativas é uma questão de configuração do MTA de origem. Na ilustração, a mensagem é entregue com sucesso após uma nova tentativa (evento 4).

Esta técnica de envio permite que o *spammer* valide e atualize os endereços de suas listas de e-mail, podendo remover aqueles e-mails que retornaram erro, evitando futuros envios sem sucesso.

Esta forma de envio também é adotada pela maioria das entidades legítimas em campanhas de marketing (e.g. sites de *e-commerce*), podendo ser de interesse parcial dos destinatários que recebem seus e-mails, dificultando ainda mais a tarefa de classificação da mensagem, uma vez que o que é *spam* para alguns usuários pode não ser considerado da mesma forma para outros. Detalhes sobre o conteúdo das mensagens de *e-mail marketing* são apresentados na seção 4.3.2.

c) Envio com substituição de remetente ou transmissor

O próprio protocolo SMTP por uma questão de projeto [1], permite que o endereço do remetente seja substituído. Assim, num ataque, além de forjar o remetente de um e-mail, também é comum que o atacante forje o endereço IP (*Internet Protocol*) do remetente (*IP Spoofing*). Essas técnicas são utilizadas para violar os mecanismos *antispam* baseados no endereço de e-mail ou IP do remetente. A Figura 4.4 ilustra este exemplo.

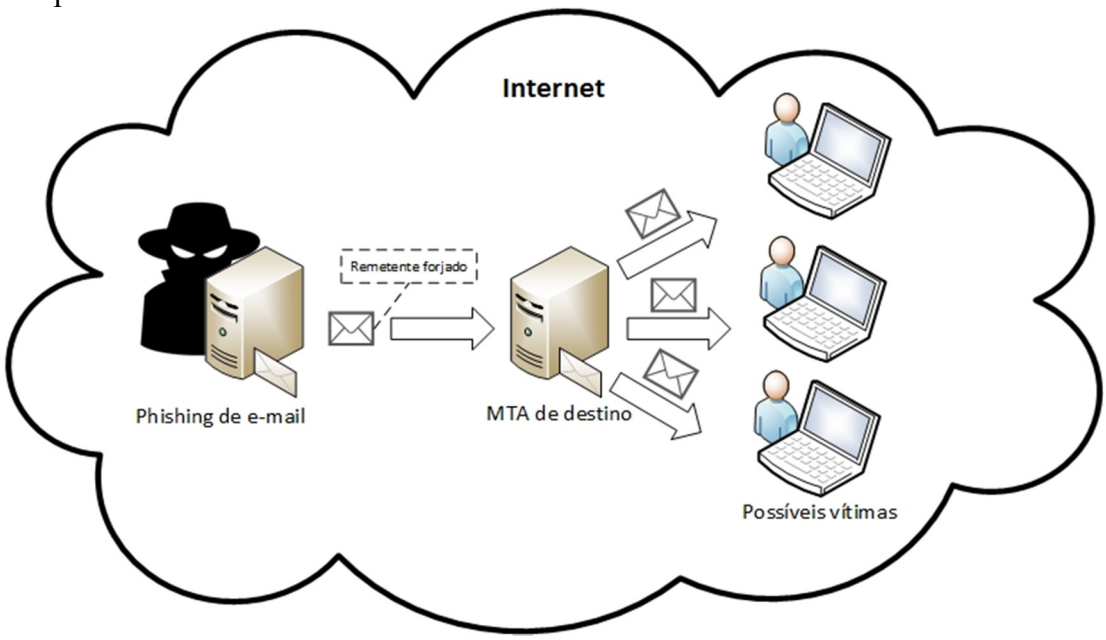


Figura 4.4. Envio com forja do remetente.

Além do intuito de burlar os mecanismos baseados no e-mail ou IP do remetente para envio de *spam*, em geral, esta prática é muito comum em casos de *phishing*[1] de e-mail. Em outras palavras, o *phishing* passa uma mensagem à vítima tentando convencê-la de que o remetente é uma fonte confiável (e.g. banco, site de *e-commerce*, órgãos governamentais etc.). Além disto, o *phishing* executa alguma ação que normalmente lhe causará algum tipo de prejuízo (e.g. cartão de crédito clonado ou senha bancária roubada)

[1] Phishing é uma forma de estelionato que usa engenharia social para fazer vítimas, enganando-as com o uso de recursos tecnológicos, normalmente com o objetivo de obter suas informações pessoais (geralmente de cunho financeiro) e causar-lhes prejuízos [2]. De acordo com o Código Penal Brasileiro, estelionato é "obter, para si ou para outrem, vantagem ilícita, em prejuízo alheio, induzindo ou mantendo alguém em erro, mediante artifício, ardil, ou qualquer outro meio fraudulento" [37].

ao usuário. Com o remetente forjado, ao abrir o e-mail, o usuário vítima do *phishing* acaba sendo convencido de que o remetente é de uma fonte confiável, sem perceber o golpe. Além de forjar o remetente, o atacante costuma usar um conteúdo da mensagem bem convincente, porém esta questão será explorada na seção 4.3.2.

d) Envio através de botnets

Uma *botnet* é uma rede de computadores comprometidos (*bots*), conectados à Internet e controlados por um atacante remoto (*botmaster*) [38]. As *botnets* utilizadas para a disseminação de *spam* geralmente são compostas por computadores comprometidos de usuários, onde foi instalado algum tipo de *malware* (código malicioso). Esses computadores são controlados remotamente, sem o consentimento do usuário, para a disseminação de *spam* (Figura 4.5).

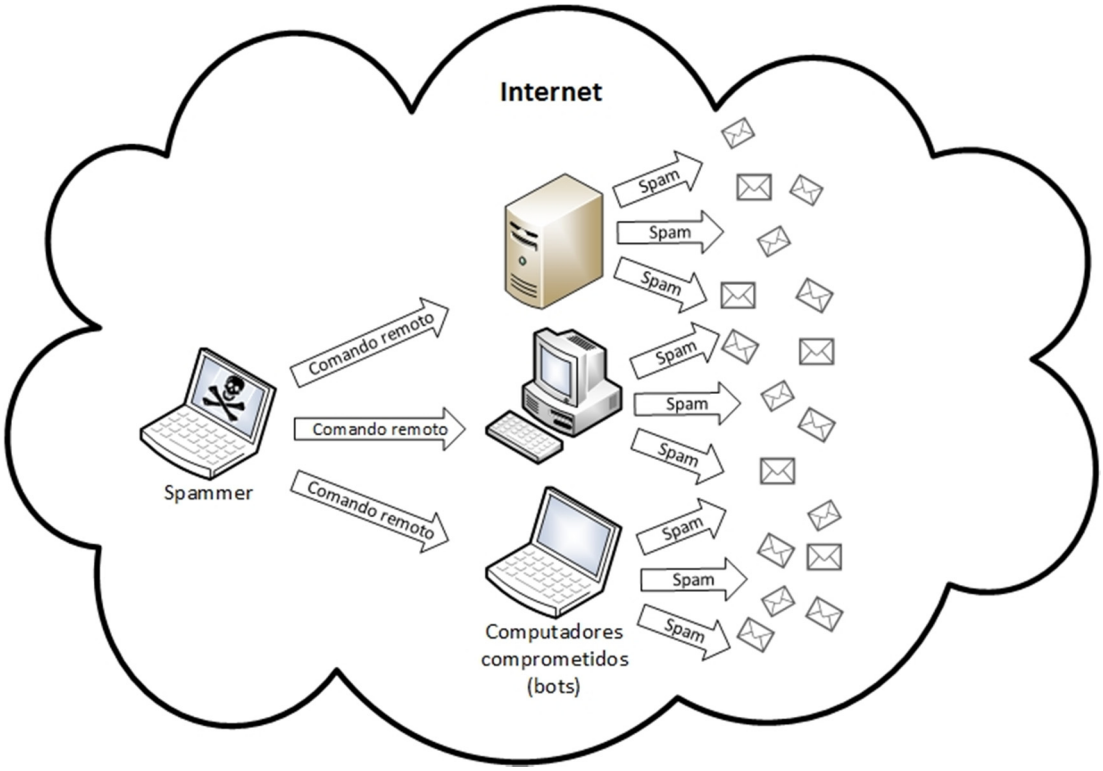


Figura 4.5. Envio através de botnets.

Como uma *botnet* tem grande escalabilidade, a dificuldade em detectar o *spam* a partir do endereço de origem é significativamente aumentada. Esta prática também facilita o anonimato do *spammer*, pois utiliza computadores de pessoas comuns, que podem estar localizados em pontos geográficos distantes, espalhados por diferentes países ou continentes, sem precisar configurar temporariamente servidores SMTP que denunciariam a sua localização. Assim, a quantidade de endereços IP de origem (computadores comprometidos) e a variedade de faixas de endereços IP (devido às diferentes localizações geográficas) são tão grandes que não há como realizar um bloqueio eficaz através dos endereços de origem.

O uso de *botnets* facilita anonimato e, portanto, é uma das técnicas mais utilizadas para a disseminação de *phishing* de e-mail, além de outras ameaças existentes na Internet. Como o *malware* presente nos computadores dos usuários costuma possuir poucas linhas de código, o *spam* enviado através de *botnets* geralmente se enquadra também na categoria de mecanismo simples de envio (seção 4.3.1.a).

e) Envio através de *open relays*

Open relays ou relays abertas são servidores SMTP onde qualquer pessoa ou sistema pode se conectar e enviar e-mails livremente através deste, sem precisar de qualquer tipo de autenticação. A conexão é feita, quase na totalidade dos casos, sem o consentimento ou conhecimento da organização responsável pelo servidor.

Durante o final da década de 90 até o início dos anos 2000, o envio de mensagens pelos *spammers* através de servidores *open relay* era uma prática muito comum. Na época, os desenvolvedores de MTAs realizaram mudanças nos códigos e na configuração padrão dos sistemas para assegurar que as instalações padrão fossem *closed relays* (relays fechadas) e tornar a criação de uma *open relay* mais difícil, de modo a permitir que os e-mails fossem enviados através do servidor somente por usuários autorizados [39]. Entre 2012 e 2013, o projeto Spamhaus registrou cerca de 4 mil registros de *open relays*, sendo que diariamente os *spammers* descobrem e exploram de 10 a 20 novas *relays abertas* [39]. A Figura 4.6 ilustra o envio de *spam* através de *open relays*.

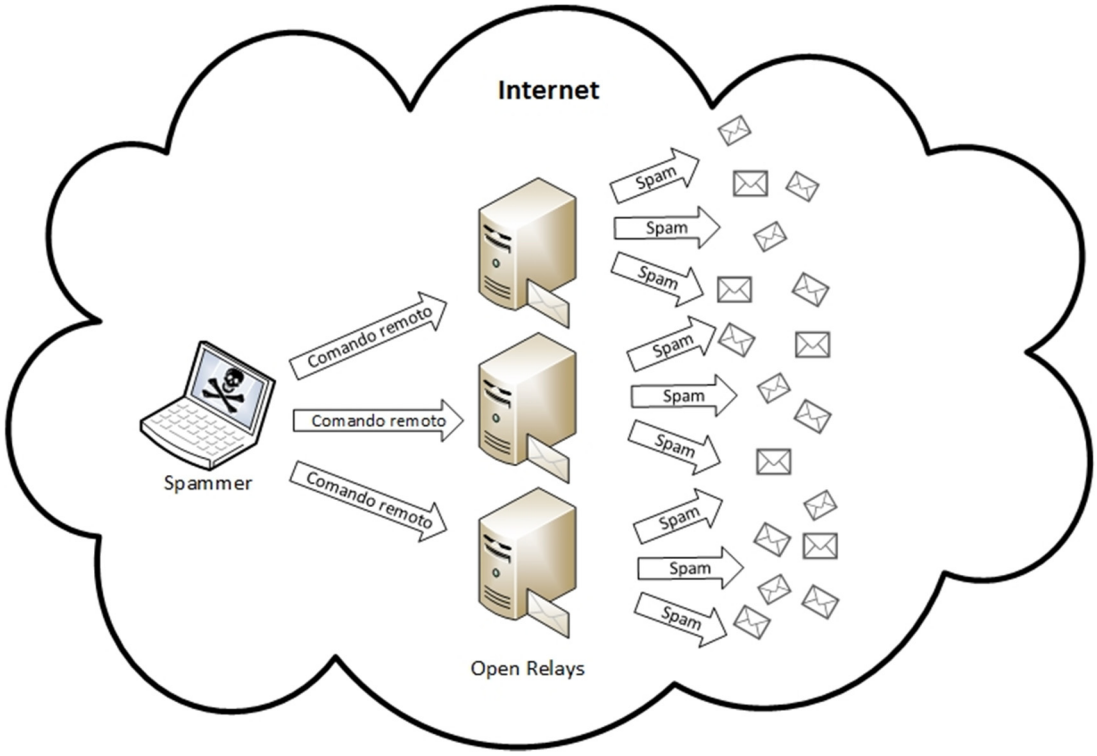


Figura 4.6. Envio de spam através de open relays.

O bloqueio de servidores SMTP *open relay* é complicado devido à grande quantidade desses servidores vulneráveis espalhados pela Internet e pelo surgimento constante de novas *relays* abertas a cada dia. Além disso, o bloqueio do endereço IP de origem, neste caso, também pode acarretar o bloqueio indevido dos e-mails de uma organização legítima.

4.3.2. Técnicas Baseadas no Conteúdo E-mail

Diferentemente das técnicas apresentadas na seção 4.3.1, as técnicas de envio de *spam* baseadas no conteúdo do e-mail estão, na maioria dos casos, associadas a violação de mecanismos *antispam* que se baseiam nas informações coletadas a partir do corpo da mensagem. Essas técnicas vão desde a inserção proposital de palavras, que confundem o

classificador de e-mails, até o uso de subterfúgios técnicos, como recursos da linguagem de hipertexto (HTML) incorporada ao e-mail.

A identificação dessas técnicas dará sequência às que foram apresentadas na seção 4.3.1, porém com foco exclusivo no corpo da mensagem. As principais técnicas são apresentadas nos itens entre a e f.

a) Inserção proposital de palavras

A *inserção proposital de palavras* é utilizada pelos *spammers* para confundir os classificadores de *spam*, que usam a ocorrência de palavras mais comuns para a detecção do ataque. É o caso dos classificadores bayesianos [40], que classificam as mensagens com base nas palavras que ocorrem com mais frequência, tanto para *spam* como para *não-spam*. Dependendo das palavras existentes no corpo da mensagem, o classificador gera uma probabilidade do e-mail ser *spam* ou não.

Para confundir o classificador, o *spammer* insere de forma proposital, palavras (texto comum) em e-mails legítimos no conteúdo do *spam*, fazendo com que a técnica de classificação utilizada caracterize a mensagem como *não-spam*, porque há predominância de texto legítimo no e-mail.

b) Troca ou inserção proposital de caracteres

A *troca ou inserção proposital de caracteres* (também conhecida por *ofuscação textual*) é outra técnica utilizada pelos *spammers* para violar os mecanismos de detecção baseados no texto da mensagem. Diferentemente da *inserção proposital de palavras*, em vez de inserir palavras que possam confundir o classificador, é realizada a troca, exclusão ou inserção proposital de caracteres nas palavras mais comuns. Este tipo de técnica prejudica a detecção tanto nos mecanismos baseados na probabilidade de mensagens como em filtros baseados em regras de detecção com palavra-chave (seção 4.3).

Por exemplo, a palavra 'viagra', muito comum em mensagens de *spam*, com o uso desta técnica poderia se apresentar de diversas formas (e.g. V1AGR4, v.i.a.g.r.a etc.), chegando a mais de *seiscentos quintilhões de variações* (*600.426.974.379.824.381.952*) [12]. Este número torna a detecção dessa e demais palavras e suas variações praticamente impossível porque esta quantidade fantástica de combinações deveria ser gerada em tempo real para todas as palavras ou armazenada em alguma base de dados. Em ambos os casos, os mecanismos de detecção baseados na ocorrência de palavras precisam comparar, no e-mail recebido, cada palavra do corpo da mensagem com todas as combinações possíveis (*strings* ofuscadas) de todas as palavras possíveis. O número de combinações é possível devido ao grande número de caracteres existentes no padrão *Unicode* que, em sua versão 8.0, reúne 120.672 códigos que representam caracteres de vários idiomas, ideogramas e coleções de símbolos [41].

No caso da substituição de caracteres, o problema pode aumentar mais ainda caso haja substituições de um caractere da palavra original por dois ou mais caracteres. A Tabela 4.1 mostra alguns exemplos.

A Tabela 4.1 apresenta alguns exemplos de ofuscação textual que podem ocorrer. A forma mais simples é a inserção de caracteres no meio da palavra como espaços, pontos, hifens etc. Os caracteres inseridos podem variar na mesma palavra (e.g. 'V I.A-G.R A'). Além da simples inserção de caracteres no meio de uma palavra específica, também pode ocorrer uma substituição $N \rightarrow 1$, ou seja, onde um ou mais caracteres (N) são utilizados para representar um caractere específico. É possível ainda que uma única palavra possua uma combinação de mais de um dos tipos de ofuscação apresentados na Tabela 4.1 (e.g.

'P|-|AR.M/\CE.UTI.C@L'). Com a grande possibilidade de inserções, substituições e combinações de caracteres para realizar a ofuscação textual, fica claro o tamanho da complexidade do problema.

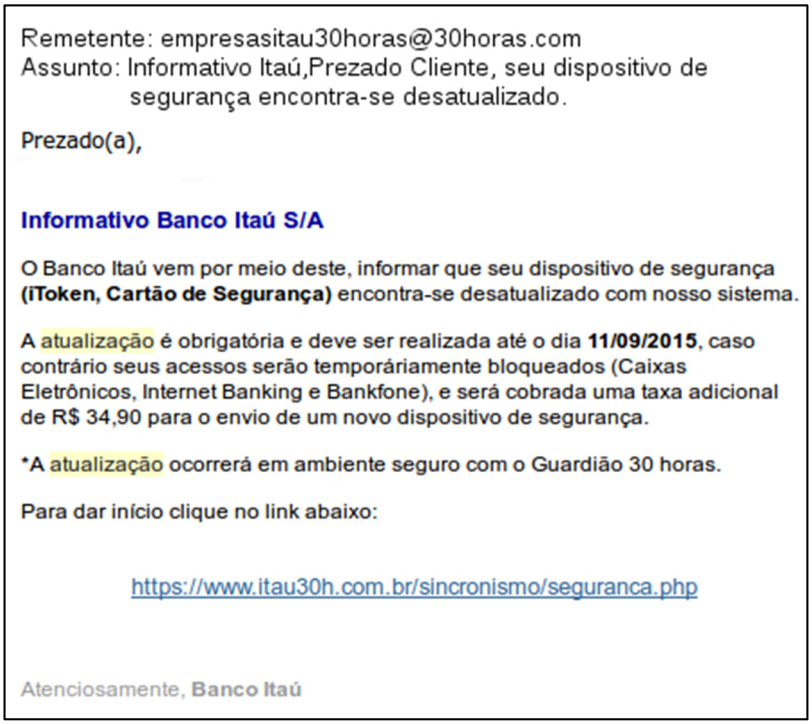
Tabela 4.1. Exemplos de ofuscação textual.

Tipo de Ofuscação	Exemplos
Inserção de caracteres	'P h a r m a c e u t i c a l', 'V.I.A.G.R.A', 'T-i-c-k-e-t', 'V I.A-G.R A'
Substituição 1⟶1	'P**h**@rmaceutic**a**1', 'V1**A**GR**4**', 'T1C**K**ET'
Substituição 2⟶1	'P**H**/\RMACEUTIC/\L', 'VI/\GR/\', 'TIC**I<**ET'
Substituição 3⟶1	'P|-|ARMACEUTIC/-\L', 'VI/-\GR/-\'

Liu e Stamm [42] realizaram experimentos para comprovar o quanto a ofuscação de palavras pode prejudicar o resultado de um classificador. Para a realização dos testes, foram substituídos termos originais (sem ofuscação) em mensagens da base de *spam* por caracteres *Unicode* que possuem semelhança visual com o caractere original. Depois de treinar a ferramenta SpamAssassin [43], foram realizados testes de classificação nas bases originais (sem ofuscamento), com ofuscamento e na base desofuscada. Os resultados obtidos demonstram que termos ofuscados têm grande impacto no resultado do classificador. O Spam Assassin atribui uma nota (*score*) para o e-mail que, quanto mais alta, maior é a probabilidade de ser *spam*. Em um dos experimentos, as mensagens de *spam* originais receberam notas de 7,9 a 21,7. Com os termos ofuscados, as notas foram de 1,9 a 5,94, ou seja, muito abaixo do que um *spam* normalmente receberia.

c) Conteúdo falso

Uma prática comum utilizada pelos *phishers* (*spammers* que disseminam *phishing*) é utilizar um texto bem convincente e muito parecido com mensagens legítimas. Mas, que na verdade ludibriam o usuário do sistema de e-mail, convencendo-o a realizar alguma ação que poderá torná-lo vítima de algum tipo de fraude.


Remetente: empresasitau30horas@30horas.com
Assunto: Informativo Itaú,Prezado Cliente, seu dispositivo de segurança encontra-se desatualizado.

Prezado(a),

Informativo Banco Itaú S/A

O Banco Itaú vem por meio deste, informar que seu dispositivo de segurança **(iToken, Cartão de Segurança)** encontra-se desatualizado com nosso sistema.

A atualização é obrigatória e deve ser realizada até o dia **11/09/2015**, caso contrário seus acessos serão temporáriamente bloqueados (Caixas Eletrônicos, Internet Banking e Bankfone), e será cobrada uma taxa adicional de R$ 34,90 para o envio de um novo dispositivo de segurança.

*A atualização ocorrerá em ambiente seguro com o Guardião 30 horas.

Para dar início clique no link abaixo:

https://www.itau30h.com.br/sincronismo/seguranca.php

Atenciosamente, **Banco Itaú**

Figura 4.7. Mensagem de um *phishing* de e-mail.

A Figura 4.7 é um exemplo de conteúdo de uma mensagem de um phishing. O texto tenta convencer a vítima que o seu dispositivo de segurança utilizado para acessar serviços bancários está desatualizado (o texto possui, inclusive, a palavra 'atualização' destacada em amarelo para chamar a atenção do usuário). Para atualizá-lo o usuário deve acessar o link informado, que o levará a baixar um malware que depois poderá lhe causar prejuízos financeiros, por exemplo, devido ao roubo de senhas bancárias.

Este tipo de e-mail acaba fugindo à regra pois, além de forjar o remetente (seção 4.3.1.c), possui características textuais muito parecidas com e-mails legítimos, tornando a mensagem ainda mais convincente aos olhos do usuário (vítima). Esses e-mails também podem conter outras artimanhas utilizadas pelos phishers, o uso de recursos da linguagem de hipertexto HTML para enganar as vítimas (assunto que será apresentado na seção 4.3.2.e). Por se parecerem com e-mails que estão presentes em bases de não-spam, os classificadores baseados no texto da mensagem muitas vezes não são eficientes contra este tipo de spam.

d) Uso de imagens

À medida que os mecanismos de detecção de *spam* textuais foram se tornando mais eficientes, os *spammers* passaram a inserir o texto da mensagem dentro de imagens, tornando os classificadores tradicionais ineficazes para este tipo de *spam* [10]. Uma das soluções para isto foi o uso de OCR (*Optical Character Recognition*) – Reconhecimento Ótico de Caracteres, que realiza a extração do texto contido nessas imagens (pré-processamento) e submetendo-o em seguida para o processamento textual. Após a adoção de técnicas de OCR como solução do problema, os *spammers* começaram a utilizar técnicas para dificultar o processamento das imagens.

Figura 4.8. Imagem com técnica de ofuscação. Adaptado de [20].

As técnicas utilizadas pelos spammers para dificultar o tratamento dessas imagens (e.g. ofuscação de imagens, Figura 4.8) aumentam consideravelmente a complexidade da classificação dos e-mails. Pela sua eficiência, esta técnica se tornou o tipo de spam mais explorado recentemente. O ofuscamento de imagens consiste em utilizar uma imagem de fundo que dificulte, de alguma maneira, o reconhecimento e a extração do texto durante o pré-processamento, mas sem prejudicar a mensagem do texto para a visão humana.

Esta prática passou a ser comum a partir de 2006 [10], sendo que no mesmo ano a quantidade de e-mail com este tipo de spam quadruplicou, representando de 25 a 45% do total, dependendo do dia [11]. Outro agravante é o tamanho médio do spam de imagem,

que é em média cerca de 10 vezes maior do que o spam textual, consumindo maior banda para trafegar na rede e mais recursos de armazenamento [16], além de necessitar de tempo e capacidade adicional de processamento no MTA de destino.

A Figura 4.8 é um exemplo de spam de imagem com técnica de ofuscação. O fundo "poluído" e a variação de cores tanto no fundo como nas letras da imagem dificultam consideravelmente a aplicação do OCR para a extração textual. Para visão humana, entretanto, é possível compreender a mensagem com um mínimo de dificuldade.

e) Uso de recursos da linguagem HTML

O uso de recursos de hipertexto (HTML) de forma mal-intencionada é muito comum em casos de estelionato, como no *phishing* de e-mail. Nesse caso, os recursos da linguagem podem ser utilizados para ocultar informações da vítima ou até mesmo confundi-la.

Um dos exemplos mais comuns do uso de recursos HTML é quando o texto âncora, ou seja, o texto visível para o usuário é uma URL (*Uniform Resource Locator*) de algum site legítimo, mas que aponta para um domínio fraudulento diferente do endereço visível para o usuário, por exemplo:

* http://www.paypal.com/login.php *

Neste exemplo, o usuário verá a URL "http://www.**pay**pal.com/login.php", porém será redirecionado para o endereço "http://**play**pal.com". Um usuário mais experiente saberá que um simples passar do ponteiro do mouse sobre o link provavelmente mostrará a verdadeira URL por trás do texto âncora, porém, esta técnica costuma funcionar com os usuários mais desatentos ou inexperientes.

Um outro exemplo poderia ser:

<img src=http://www.dpf.gov.br/logo.png> Você está intimado a comparecer em nossa delegacia! Clique aqui para saber o motivo .

Neste caso, a vítima enxergaria a mensagem "*Você está intimado a comparecer em nossa delegacia! Clique aqui para saber o motivo"*, com uma imagem do logo da organização, retirado diretamente do Portal da Polícia Federal e contendo um link para o endereço "http://badsite.com/malware.exe", que aponta para um arquivo executável que provavelmente contém algum tipo de *malware*.

No trabalho de Olivo, C. K., Santin, A. O. e Oliveira, L. S. [2] há exemplos de vários outros casos de uso do HTML que são comuns em *phishing*.

f) Campanha Publicitária (*E-mail marketing*)

Campanhas publicitárias (*e-mail Marketing* ou marketing por e-mail) são mensagens com fins publicitários que geralmente são enviadas por um MTA com considerável poder de processamento, que possui um domínio autêntico para envio de e-mails, tratamento de erros etc. É importante ressaltar que o *e-mail marketing* não é exatamente uma técnica de disseminação de *spam* baseada no conteúdo da mensagem, pois nem sempre o objetivo primário destes e-mails é causar problemas ao usuário ou administradores de servidores de e-mail. O problema dessas mensagens é sua classificação pelos usuários (que podem considerá-las mensagens não solicitadas – *spam*), esta é a razão pela qual esta categoria de e-mails está incluída nesta seção.

Diferentemente de e-mails que promovem a venda de medicamentos (e.g. Viagra, Cialis etc.) e outras categorias de *spam* que são indesejados por quase todos os usuários, este tipo de e-mail pode ser do interesse de alguns por conter assuntos de interesse específico, tais como: promoções de produtos, lançamentos etc. A maioria destes e-mails de campanhas publicitárias também oferece ao usuário a opção de remoção do seu endereço da *mailing list*, para que não receba mais o que considera *spam*.

4.3.3. Considerações sobre as técnicas de disseminação de spam

Conforme apresentado nas seções anteriores, há uma grande variedade de técnicas e tipos de *spam*. A Tabela 4.2 sumariza as principais características, objetivos e principais dificuldades de detecção para cada tipo de *spam*.

A grande diversidade de técnicas de disseminação ou tipos de *spam* aumenta consideravelmente a complexidade do problema. Sem entender a complexidade do problema, ao analisar uma técnica específica de detecção, é impossível ter a compreensão necessária da abordagem apresentada, identificando em que momento o mecanismo de detecção poderá falhar.

Tabela 4.2. Exemplos de ofuscação textual

Tipo de Spam	Características	Objetivos
Mecanismo simples de envio	• Normalmente ocorre o envio de milhares, ou mesmo dezenas de centenas de milhares de e-mails. • Envio sem tratamento de erros. • Transmissão da mensagem sem tentativa de reenvio em caso de erro. • Comum em casos de computadores comprometidos por *malwares* que disseminam *spam*. • Sistemas de envio de e-mail sem robustez. • Poucas linhas de código são necessárias para o mecanismo de envio. • Baixo custo computacional para envio da mensagem.	• Atingir o maior número possível de destinatários no menor tempo possível, sem se preocupar com erros de transmissão.
Envio com tratamento de erros	• Mecanismos de envio mais robustos. • Ocorre o tratamento de erros ou retransmissão da mensagem. • Muito comum em casos de *e-mail marketing*.	• Dificultar a detecção do *spam* contra técnicas que são eficazes contra *mecanismos simples de envio*. • Possibilitar a validação de endereços de e-mail, excluindo aqueles que retornam com erro, facilitando o controle do processo de *spamming*.
Envio com substituição de remetente ou transmissor	• Ocorre a falsificação do endereço IP ou do endereço de e-mail do remetente da mensagem. • No caso da falsificação do e-mail, a técnica é possível devido às limitações do protocolo SMTP. • Muito comum em casos de *phishing* de e-mail.	• Violar os mecanismos baseados em detecção de *spam* através do endereço de e-mail ou IP do remetente. • Enganar a vítima, fazendo-a acreditar que o e-mail foi encaminhado de uma fonte confiável.
Envio através de *botnets*	• Envio de *spam* através de computadores comprometidos por *malwares*, sem o consentimento dos seus usuários.	• Ocultar a localização do *spammer*, já que o envio é feito através de computadores comprometidos de terceiros.

Tipo de Spam	Características	Objetivos
	• Grande escalabilidade da *botnet*, o que dificulta a detecção do *spam* a partir do endereço de origem. • Também se enquadra na categoria *mecanismo simples de envio*.	• Atingir o maior número possível de destinatários no menor tempo possível, através do envio massivo de e-mails, devido à alta escalabilidade das *botnets*.
Envio através de *open relays*	• Servidores SMTP que permitem que qualquer pessoa ou sistema se conecte livremente, utilizando-os para a disseminação de *spam*. • Envio realizado sem o consentimento da organização responsável pelo servidor SMTP configurado como *open relay*.	• Dificultar a localização do *spammer*, já que o envio é feito através de servidores SMTP que pertencem a terceiros. • Dificultar o bloqueio a partir do endereço IP de origem, devido à grande quantidade de servidores com esse tipo de vulnerabilidade na Internet.
Inserção proposital de palavras	• Inserção proposital de palavras comuns em e-mails que não são *spam*.	• Enganar os classificadores baseados na ocorrência de palavras mais comuns em mensagens de *spam*.
Troca ou inserção proposital de caracteres	• Técnica também conhecida como ofuscação textual. • Ocorre a troca, exclusão ou inserção proposital de caracteres em palavras comuns em *spam*. • Mesmo com a troca dos caracteres em determinadas palavras, a compreensão humana não é prejudicada. • Possibilidade de geração de inúmeras variações de uma única palavra. • Considerável complexidade de detecção.	• Enganar os classificadores baseados na ocorrência de palavras mais comuns em mensagens de *spam*.
Conteúdo falso	• Texto bem convincente, com características textuais semelhantes a mensagens legítimas. • Normalmente também é utilizado com a *forja de remetente*. • Normalmente utiliza recursos hipertexto da linguagem HTML para enganar as vítimas. • Classificadores baseados em características textuais da mensagem muitas vezes não são eficientes contra este tipo de *spam*.	• Ludibriar o usuário do sistema de e-mail, convencendo-o a realizar alguma ação que poderá torná-lo vítima de algum tipo de fraude.
Uso de imagens	• Texto da mensagem passa a ser "embutido" em imagens. • Um dos tipos de *spam* mais explorados na literatura. • Pode ocorrer o uso de técnicas de ofuscação de imagens. • Tamanho médio do *spam* de imagem é dez vezes maior do que o spam textual. • Consome mais recursos de armazenamento e gera maior tráfego na rede. • Necessita de técnicas adicionais de OCR (pré-processamento) para extrair o texto da imagem para posterior processamento textual.	• Burlar os mecanismos *antispam* baseados em características textuais. • Nos casos de ofuscamento de imagens, o objetivo é dificultar o tratamento desse tipo de *spam*.
Uso de recursos da	• Seu uso mal-intencionado é muito comum em casos de estelionato, como no *phishing* de e-mail.	• No caso de *phishing*, o objetivo é ocultar informações da vítima ou confundi-la.

Tipo de Spam	Características	Objetivos
linguagem HTML		
E-mail marketing	• Normalmente é encaminhado através de MTAs robustos. • Geralmente o envio de e-mails é realizado com o tratamento de erros. • Não há unanimidade na classificação desses e-mails por parte dos usuários (se é *spam* ou não-*spam*). • A maioria desses e-mails permite que o usuário remova seu endereço da lista de envio.	• Realizar campanhas publicitárias por e-mail, usando assuntos que podem ser do interesse de apenas alguns usuários. • Nem sempre o objetivo primário desses e-mails é causar problemas ao usuário e administradores de servidores de *e-mail*.

Com o objetivo de aumentar a compreensão sobre o problema e complexidade do *spam*, a seção 4.3 apresentou as principais técnicas de disseminação utilizadas pelos *spammers*, com o objetivo de facilitar a análise das técnicas de classificação de e-mails que serão apresentadas nas próximas seções.

4.4. Principais técnicas utilizadas para detecção de spam

Diante das técnicas mais utilizadas para a disseminação de *spam*, várias propostas foram criadas prometendo solucionar ou mitigar o problema. Esta seção apresentará as principais técnicas encontradas na literatura.

4.4.1. Técnicas de detecção de spam baseadas em regras, políticas ou protocolos

a) Whitelists e Blacklists

O exemplo mais comum de regra para bloqueio de *spam* é o uso de listas de bons (*whitelists)* e maus (*blacklists)* remetentes. Basicamente, a regra consiste em aceitar ou rejeitar todo e-mail, domínio ou IP contido nessas listas. No caso das *whitelists*, há a possibilidade de liberar o e-mail destinado a caixa de entrada do usuário sem precisar passar pelos demais mecanismos e classificadores, agilizando o processo de entrega e diminuindo a carga de processamento das mensagens no MTA de destino. Este tipo de configuração, entretanto, pode ser um risco para a segurança por aceitar qualquer mensagem de endereços que estejam na *whitelist*. Da mesma forma são utilizadas as *blacklists* para o bloqueio das mensagens. A Figura 4.9 ilustra o funcionamento dessas listas.

Na Figura 4.9 o e-mail que não é *spam* é enviado por um remetente que consta na *whitelist* do MTA de destino, sendo entregue diretamente ao MUA do usuário sem qualquer tipo de processamento da mensagem. Uma das vantagens do uso de *whitelists* é evitar o bloqueio indevido e reduzir o custo computacional para o processamento de e-mails de fontes confiáveis. Entretanto, se esses remetentes "confiáveis" estiverem comprometidos por algum tipo de *malware*, poderá representar um sério risco à segurança do servidor de seus usuários e redes as quais estiver conectado.

Na Figura 4.9 no exemplo em que um *spam* é encaminhado, o endereço do *spammer* está presente na *blacklist* e, portanto, a mensagem é bloqueada. O uso de *blacklists* é útil para o bloqueio de fontes de *spam* conhecidas.

O uso destas listas, apesar de parecer eficiente, possui várias limitações e o seu uso é recomendado somente em último caso. O bloqueio do endereço IP ou domínio pode

causar problemas quando o remetente utiliza o servidor de SMTP de algum provedor (e.g. Yahoo, Gmail etc.), pois acaba por bloquear todos os remetentes que o utilizam. Já o bloqueio do e-mail do remetente pode ser ineficiente, visto que o mesmo pode ter sido forjado (veja a seção 4.3.1.c).

No caso do *phishing* (seção 4.3.2.c) a origem da mensagem (e.g. endereço IP, URL alvo do *phishing*, e-mail forjado etc.) costuma mudar constantemente para evitar seu rastreamento. Além disso, a dificuldade de administração dessas listas pode se tornar muito complexa, pois o fluxo de mensagens pode ser muito intenso no servidor SMTP onde a filtragem é realizada. Assim, esta abordagem geralmente é ineficiente [2].

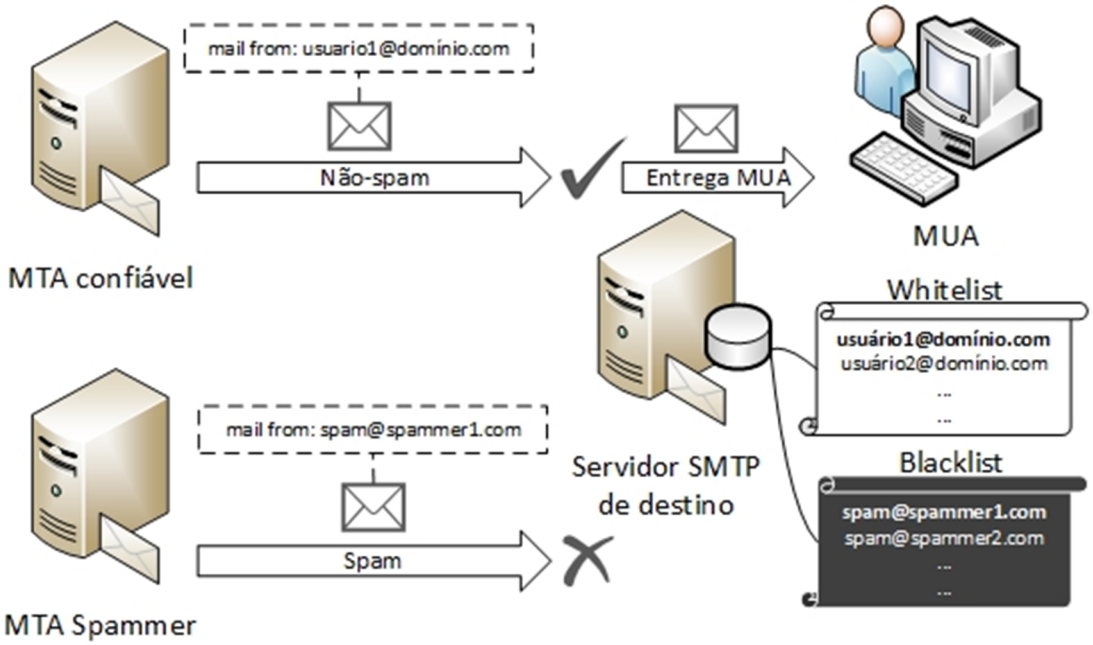


Figura 4.9. Funcionamento de whitelists e blacklists.

b) Mecanismos *antispam* baseados em palavras-chave

De modo similar as *blacklists*, também é possível bloquear e-mails que contenham determinadas palavras no corpo da mensagem. As palavras inseridas na lista de palavras-chave podem fazer uso de expressões regulares para identificar algumas variações de *strings* de caracteres.

Na figura 4.10 há um exemplo de palavras-chave utilizadas no bloqueio de *spam*. De acordo com Jargas, A. M. [44], uma expressão regular é um método formal de se especificar um padrão de texto. A expressão, quando aplicada em um texto qualquer, retorna sucesso caso este texto obedeça a todas as suas condições. Na Figura 4.10, entre colchetes estão os caracteres (expressão regular) que podem ocorrer em determinada posição da *string*. Apesar de ser um exemplo simples, a complexidade das expressões regulares pode ser muito maior, sendo muito útil no momento de compor essas listas de palavras, abrangendo um número considerável de variações de uma mesma *string* de caracteres. Entretanto, sem a perícia adequada, o uso de expressões regulares pode causar o bloqueio indevido dos e-mails.

O bloqueio a partir de palavras-chave deve ser utilizado somente em último caso ou de maneira paliativa, por exemplo, bloqueando um *spam* que não está sendo detectado pela técnica de classificação textual, de forma temporária, usando assunto da mensagem ou alguma palavra ou frase no corpo do e-mail para selecionar o alvo do bloqueio. Como esse tipo de bloqueio não considera o contexto da mensagem como um todo e não calcula

nenhum tipo de probabilidade da mensagem ser ou não *spam*, a inserção de palavras na lista de bloqueio deve ser feita com bastante cautela, a fim de evitar o bloqueio indevido de mensagens.

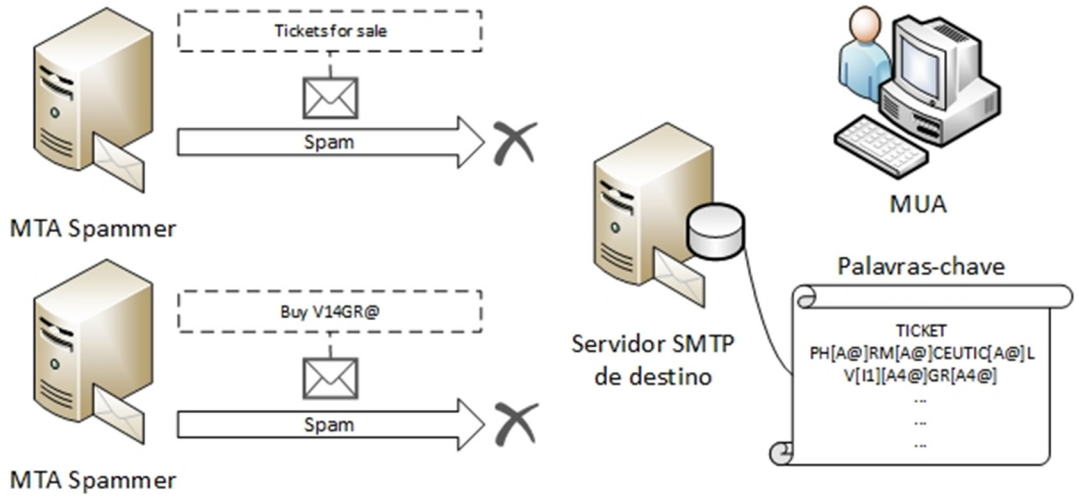


Figura 4.10. Bloqueio por palavras-chave.

c) *Greylisting*

Em configurações mais rudimentares de *spam* a maioria das mensagens é enviada através de programas pouco robustos, tentando atingir o máximo de destinatários possíveis em um determinado tempo (seção 4.3.1.a). Não dispondo de um MTA completo, as mensagens de *spam* que não fossem aceitas pelo servidor de destino não eram colocadas em fila para reenvio posterior. Isso é muito comum atualmente, em computadores comprometidos em uma *botnet* para envio de *spam* (seção 4.3.1.d). Além disso, para evitar seu rastreamento, é comum o *spammer* não possuir nenhum domínio registrado em seu nome. Esse fato levou à criação de diversas outras técnicas baseadas em listas, como o *greylisting* [45] e o SPF (*Sender Policy Framework*) [46].

A *greylisting* (Figura 4.11) consiste numa recusa inicial da mensagem recebida pelo MTA de destino (evento 1), supondo que o *spammer* não dispõe de um MTA completo que reenviará a mensagem em caso de falha na entrega. O endereço do remetente é colocado temporariamente na *greylist*, onde permanecerá por um tempo determinado aguardando o reenvio da mensagem (evento 2). O servidor SMTP de destino enviará uma mensagem de erro (evento 3) informando que a mensagem foi colocada na *greylist*. Na origem, se o MTA estiver configurado devidamente para reprocessar as mensagens, o e-mail é colocado numa fila para reenvio (evento 4). Na nova tentativa de envio da mensagem, o servidor SMTP de destino fará a checagem do endereço do remetente, que desta vez estará na *greylist*, aceitando a mensagem (evento 5). O tempo que os endereços permanecem na *greylist* e o tempo que em que ocorre uma nova tentativa de envio da mensagem, correspondendo a um detalhe de configuração dos servidores.

Levine, J. R. [47] cita várias situações em o *greylisting* pode falhar, por exemplo:

(1) Perda de e-mails legítimos, caso o MTA de origem não esteja configurado para reenviar mensagens com falha na entrega;

(2) Atraso na entrega de e-mails (tempo entre o primeiro e segundo envio), o que se agrava dependendo da configuração do MTA de origem;

(3) Máquinas de envio de *spam* mais robustas podem suportar o reenvio de mensagens, o que inviabiliza o uso desta técnica (seção 4.3.1.b).

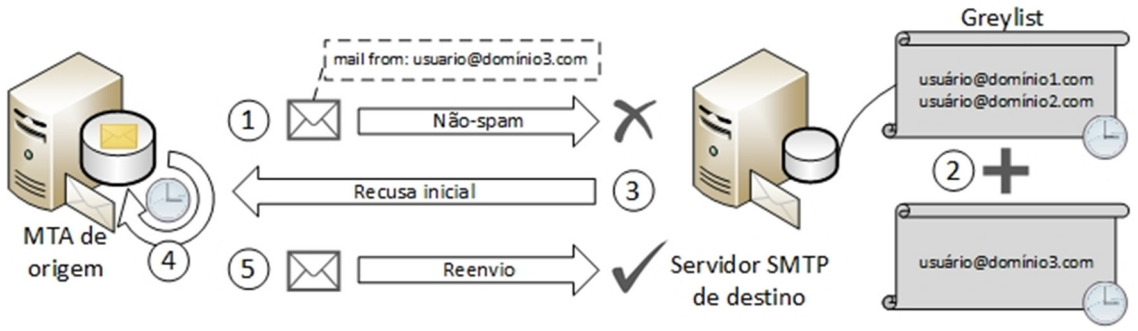


Figura 4.11. Cenário de uso de Greylisting.

d) Framework com Política de Remetente (SPF - *Sender Policy Framework*)

O SPF é um padrão aberto que especifica um procedimento para prevenir a substituição do remetente do e-mail (seção 4.3.1.c – *envio com substituição de remetente ou transmissor*). Mais especificamente, a versão atual do SPF (SPFv1 ou SPF Clássico) protege o campo *"envelope sender address"* (também conhecido como *return-path*), que é o campo utilizado pelos MTAs de origem e destino na entrega das mensagens, inclusive nos casos de erro na entrega do e-mail e retorno ao remetente [46].

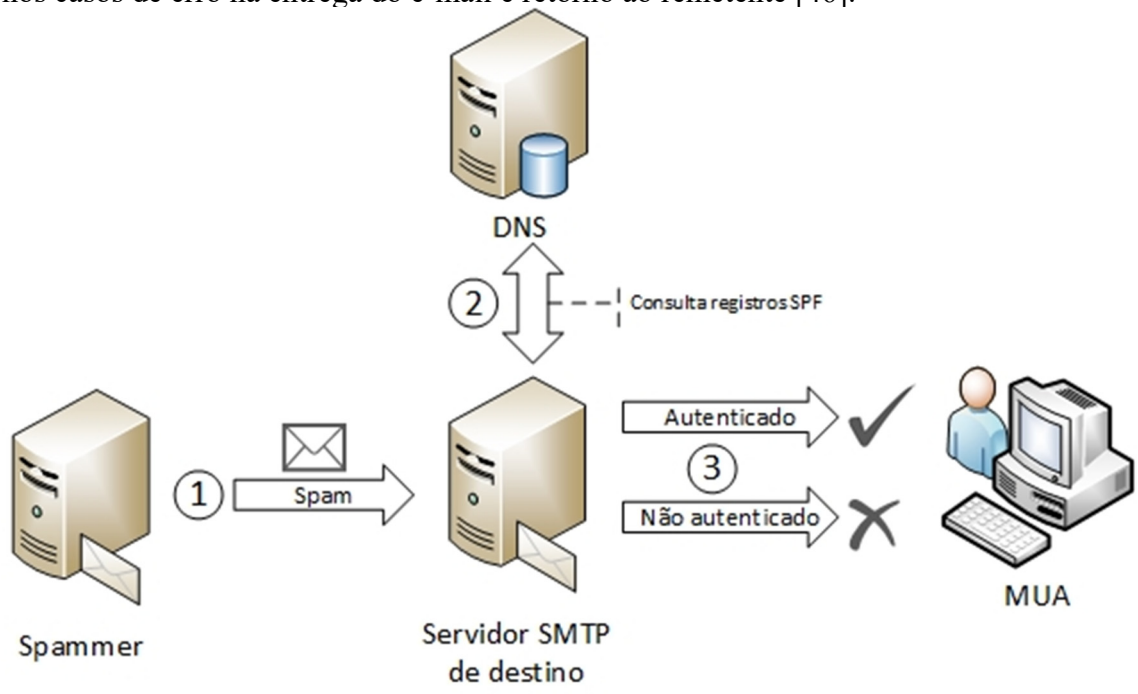


Figura 4.12. Cenário de uso de SPF.

O SPF funciona de modo que o proprietário de um domínio possa especificar quais servidores são utilizados para enviar e-mails. Para que o procedimento funcione, primeiramente o proprietário do domínio remetente deve publicar um registro SPF no DNS (*Domain Name System*), o qual contém o endereço de um servidor autorizado a enviar e-mail em nome do remetente. Além disso, o servidor de destino deve fazer a checagem desse registro, rejeitando a mensagem caso essa não seja originada em um endereço especificado na política SPF. Uma vez que a autenticidade do domínio do servidor do remetente é confirmada, este poderá ser adicionado a alguma lista ou sistema de reputação [46]. A Figura 4.12 ilustra o funcionamento desta técnica.

Atualmente, é sabido que boa parte dos *spammers* possui os recursos de um MTA completo (seção 4.3.1b), com a possibilidade de reenvio da mensagem em caso de falha

(ou na recusa da primeira mensagem pela *greylisting*), ou mesmo publicando o registro SPF do seu servidor de e-mail (de onde o *spam* é enviado), tornando esses mecanismos ineficazes na maioria dos casos [9, 47]. Um bom exemplo da limitação da proposta do SPF é o *"spam* comercial" enviado por empresas de *e-commerce*, onde as mensagens publicitárias, na maioria das vezes, são encaminhadas através de servidores de e-mail completos (que podem, inclusive, possuir registros SPF publicados). Além disso, nem todos os servidores de domínios confiáveis (e.g. bancos, órgãos governamentais, empresas, etc.) possuem seus registros SPF publicados, não impedindo que terceiros mal-intencionados enviem e-mails em nome desses domínios.

e) Técnicas Baseadas em Assinatura

Várias técnicas baseadas na assinatura das mensagens foram desenvolvidas para mitigar o problema do *spam*. Uma das mais conhecidas, o DKIM (*Domain Keys Identified Mail*), define um mecanismo pelo qual as mensagens de e-mail podem ser assinadas digitalmente, permitindo que um domínio assinante reivindique a responsabilidade pela introdução da mensagem no fluxo de e-mails. Os destinatários da mensagem podem verificar a assinatura solicitando a chave pública diretamente ao domínio assinante e assim confirmar que a mensagem foi verificada por alguém em posse da chave privada para o domínio remetente [48].

Em outras palavras, o DKIM é uma especificação do IETF (*Internet Engineering Task Force*) que define um mecanismo para autenticação de e-mail baseado em criptografia de chaves públicas. Através do uso do DKIM (Figura 4.13) uma organização assina digitalmente as mensagens que envia, permitindo ao receptor confirmar a autenticidade das mensagens que recebe. Para verificar a assinatura digital, a chave pública é obtida por meio de consulta ao DNS do domínio do remetente. Ao contrário do SPF, que verifica somente o envelope, o DKIM verifica o cabeçalho da mensagem. Assim, o DKIM impõe um custo computacional adicional para processar cada mensagem, tanto para o MTA remetente quanto para o receptor [49].

O DKIM, além de verificar a assinatura da mensagem, também possibilita a verificação da integridade do conteúdo do e-mail. A autenticação da mensagem auxilia no controle de *spam* e *phishing* de e-mail [50].

Antes do DKIM, houve outros quatro padrões IETF para assinatura de e-mails [50], como listado a seguir:

- PEM - *Privacy Enhaced Mail* (1987) [51];
- PGP – *Pretty Good Privacy* (1991), padronizada mais tarde como OpenPGP [52];
- MOSS - *MIME Object Security Services* (1995), originado do PEM [53];
- S/MIME – *Secure MIME*, desenvolvido de forma independente pela RSA Security e mais tarde padronizado pela IETF [54].

De modo geral, as técnicas de autenticação de e-mails baseadas em assinatura poderiam ser a solução definitiva para problemas como o *phishing* de e-mail. Entretanto, a falta de padronização da técnica de autenticação utilizada no serviço de e-mail e a não-obrigatoriedade do uso da mesma impede a solução do problema. Na verdade, o próprio protocolo SMTP deixa a desejar quando tolera a ausência de uma técnica de autenticação. Essas técnicas consumem recursos computacional e configurações adicionais no sistema de e-mail e, portanto, nem sempre são adotadas. Além disso, um remetente (MTA de origem) que não assina suas mensagens não necessariamente é uma fonte não-confiável, mas a recusa de mensagens desta fonte nem sempre é possível. O mesmo vale para as

mensagens autenticadas, que mesmo que assinadas digitalmente, não são necessariamente de fontes confiáveis. A impossibilidade de verificação da autenticidade de todas as mensagens, somada ao fato que a assinatura em uma mensagem não significa que a mesma não seja *spam*, faz com que as técnicas de autenticação não representem a solução definitiva para o problema.

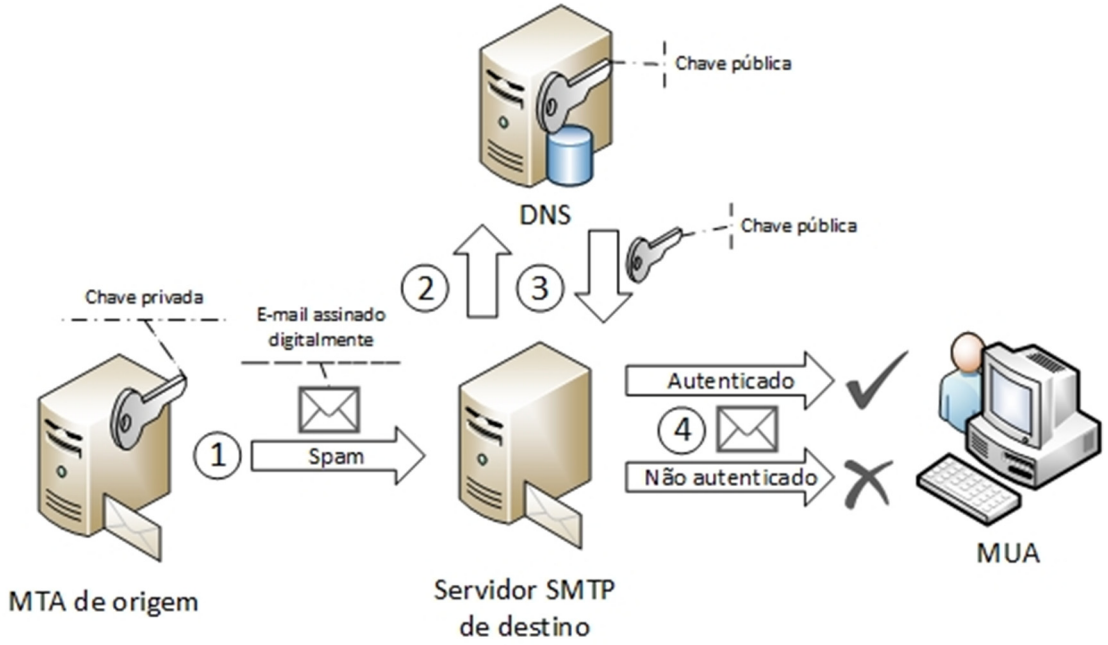


Figura 4.13. Cenário de uso de DKIM.

4.4.2. Técnicas de detecção de spam baseadas reconhecimento de padrões

Reconhecimento de padrões consiste na descoberta automática de padrões nos dados, usando algoritmos computacionais, que permitem classificar os dados em diferentes categorias [55]. Técnicas de reconhecimento de padrões são muito utilizadas no combate ao *spam*. Em se tratando de detecção de *spam*, algumas das técnicas ou classificadores mais utilizados são as abordagens Bayesianas, Redes Neurais, Árvores de Decisão e SVM (*Support Vector Machines*).

De um modo geral, as abordagens propostas para detecção de *spam* podem ser divididas em dois tipos:

(1) As que utilizam essencialmente a frequência de ocorrência de palavras como característica;

(2) As que se utilizam de várias características presentes no e-mail como informações de cabeçalho, uso de recursos específicos da linguagem HTML etc.

Também é possível encontrar propostas que combinam ambas [56]. Dentre as propostas que utilizam a frequência de palavras para a classificação de e-mails, os filtros bayesianos estão entre os mais comuns. A teoria da decisão bayesiana é uma abordagem estatística fundamental para o problema de classificação de padrões [40], sendo o classificador *Naïve Bayes* (NB) um dos mais utilizados para a classificação textual [57] (inclusive para a classificação de e-mails). Uma das razões é a sua simplicidade, o que o torna fácil de ser implementado e com boa taxa de acerto, comparando-se a outros algoritmos mais elaborados de aprendizagem de máquina [58]. O algoritmo NB tem sido um dos mais utilizados em propostas comerciais de filtros *antispam* [59].

Chen, C. e seus colegas [59] fizeram uma comparação de três propostas que utilizam NB [60, 61, 62] e mais quatro abordagens tradicionais (SVM, C4.5, NB e KNN). De um modo geral, a proposta apresenta alguns aprimoramentos no algoritmo NB, tentando comprovar que é melhor utilizar o algoritmo NB aprimorado do que métodos tradicionais de classificação (incluindo o próprio NB). Porém, ou autores não apresentam algumas informações essenciais, tais como taxa de falsos positivos.

Drucker, H., Wu, S., e Vapnik, V. N [63] apresentaram um trabalho que, embora seja antigo para área, apresenta vários conceitos muito utilizados em técnicas baseadas na detecção por frequência de ocorrências de caracteres, tais como TF (*Term Frequency*) – quantidade de vezes que uma palavra aparece em um documento, TF-IDF (*Term Frequency – Inverse Document Frequency*) – que define a importância de um termo dentro de uma coleção de documentos, e *Stop List* – lista de termos que não devem aparecer no vetor de características. Adicionalmente, também apresentam várias técnicas muito utilizadas para avaliar a performance do classificador e validação dos resultados, além de fazer um estudo do uso do classificador SVM em comparação com outros algoritmos de classificação (*Ripper*, *Rocchio* e *Boosting Decision Trees*).

Além das abordagens mais tradicionais de reconhecimento de padrões utilizadas para a detecção de spam, algumas propostas buscam a solução dos problemas sob uma outra perspectiva. Ma, W. e seus colegas [64], por exemplo, exploram a detecção de *spam* através de "seleção negativa", ou seja, uma linha de reconhecimento de padrões para casos em que não houve uma etapa de aprendizado (AMO – *Artificial Immune Systems*). Uma das vantagens seria a pro-atividade na detecção, sem necessitar uma etapa de treinamento, o que quase sempre é necessário nas abordagens *antispam* tradicionais.

Outra técnica que visa facilitar a etapa de aprendizado é o *co-training*, que permite a construção de um classificador com um número relativamente pequeno de amostras rotuladas [65]. Após a construção desse primeiro classificador de taxa de acerto "fraca", esse mesmo classificador vai se tornando mais robusto com e-mails não rotulados. O princípio base desta técnica é considerar que um classificador "fraco", feito com amostras rotuladas, pode encontrar amostras muito similares em uma base não rotulada. Então, vão sendo adquiridas novas amostras rotuladas para alimentar a base e refazer seu treinamento. Kiritchenko e Matwin [66] utilizaram *co-training* para a classificação de e-mails, realizando 50 treinamentos na base, à medida que mais e-mails eram classificados a taxa de acerto aumentava. No primeiro treinamento foi possível classificar corretamente 90% das mensagens com o classificador SVM, mas após 50 iterações de treinamento o resultado aumentou para 94%.

Dentro desta mesma área, alguns trabalhos propuseram soluções para combater as técnicas de ofuscação textual utilizadas pelos *spammers*. Braga e Ladeira (2007) propuseram uma abordagem que considera a dinamicidade do spam, ou seja, a quantidade de variações que costumam surgir visando burlar os mecanismos de detecção [67]. A abordagem é composta por três módulos, com função de pré-processamento, classificação e adaptação. A etapa de pré-processamento transforma a mensagem em valores numéricos, fazendo uso de árvores de Huffman adaptativas (árvores FGK) e um algoritmo de ordenação das mensagens com base na representação vetorial das palavras que a compõe. A vantagem de utilizar uma árvore adaptativa é a sua capacidade de acrescentar novas folhas sem precisar criar uma nova árvore desde o início. Na etapa de classificação foi utilizado o classificador SVM. A adaptação das mensagens é feita através de uma técnica chamada envelhecimento exponencial. Para retreinar o classificador no tempo i+1, são utilizadas novas mensagens que chegaram até o tempo i+1, porém nem todas as

mensagens do tempo *i* são escolhidas, dando maior importância às mensagens mais recentes.

O número de mensagens escolhidas em cada conjunto treinado decresce exponencialmente, e a periodicidade com que ocorre cada treinamento pode ser ajustada. O interessante dessa abordagem é que a mesma considera que as mensagens de *spam* mudam ao longo do tempo, possibilitando que, a cada novo treinamento, seja dado uma maior importância às mensagens mais recentes. Além disso, a abordagem também faz uso de um modelo numérico (Árvore de Huffman Adaptativa - FGK), o que acelera todo o processo se comparado ao uso de texto na aplicação das técnicas propostas.

Uma outra proposta parecida foi apresentada por Zhou et. Al [69], também fazendo uso de árvores FGK [68]. Os resultados mostram que o modelo com Árvore de Huffman Adaptativa levou vantagens sobre outras sete técnicas de classificação em oito dentre dez testes realizados. Seguindo a mesma linha de utilizar métodos de compressão para a detecção de *spam* em geral, alguns trabalhos propuseram o uso do Modelo de Markov [69, 70, 71, 72]. Modelos de compressão estatísticos podem ser utilizados para estimar a probabilidade de uma certa sequência de caracteres, podendo ser aplicados como classificadores Bayesianos.

Lee e Ng [74] utilizaram um método que faz uso do Modelo de Markov para o desofuscamento de palavras em *spam,* o qual eles chamam de Árvore Léxica do Modelo Escondido de Markov (LT-HMM – Lexicon Tree Hidden Markov Model) [73]. A abordagem integra ao Modelo de Markov um dicionário (léxico) e informações de contexto. Por exemplo, para que a palavra ofuscada *mortg3ge* seja convertida para *mortgage* (hipoteca), é necessário saber que *mortgage* é uma palavra do idioma inglês. O dicionário é construído através do modelo proposto, ou seja, da árvore léxica que na verdade é uma árvore de prefixos. Os resultados apresentados comprovam que a técnica é eficiente para desofuscar palavras, porém a abordagem possui um ponto fraco que é utilizar apenas os caracteres da tabela ASCII, deixando de lado os milhares de caracteres que estão na tabela Unicode. Além disso, a LT-HMM possui um número muito grande de estados (110.919 estados), o que não é muito conveniente para aplicações práticas.

Sculley et al. [75] buscaram solucionar o problema da ofuscação textual com o uso de algoritmos de busca aproximada. Uma das motivações para o uso desse tipo de técnica é que elas têm sido aplicadas com sucesso em áreas de biologia computacional com dados genômicos, já que as *strings* formadas por esse tipo de dados possuem inserções, substituições e exclusões de caracteres causadas por motivos evolucionais, fazendo semelhança com o que ocorre em termos comuns em *spam.* A técnica utilizada para fazer a busca aproximada de s*trings* é conhecida como *k-mers*. O problema do uso desse tipo de técnica é o custo computacional necessário para realizar a busca. Para resolver esse problema, foi proposto o uso de variantes desse método que utilizam os *kernels gappy* e *wildcard*, juntamente com o classificador *perceptron with margins*. Os resultados apresentados, apesar de interessantes, foram obtidos em uma base de *spam* que não possuía muitas variações de caracteres, se comparado ao grande número de possibilidades dentro do padrão Unicode (segundo o artigo, havia somente 25 variações da palavra 'viagra' com uma variedade mínima de caracteres visualmente semelhantes). Além disso, apesar de ser mencionado a importância de um bom desempenho para esse tipo de aplicação, não é apresentado nenhum resultado que mostre a sua aplicabilidade em um ambiente real de produção.

As técnicas de reconhecimento de padrões estão entre as mais utilizadas e mais pesquisadas na literatura relacionada ao combate ao *spam.* Devido à variedade de técnicas existentes, há vários trabalhos que analisam ou testam várias dessas técnicas na

classificação de e-mails [76, 77]. Embora bastante promissores, os classificadores mais utilizados no combate ao *spam* passam a ser do conhecimento dos *spammers*, que podem explorar determinadas limitações em etapas importantes da classificação, tal como a aprendizagem. Por exemplo, um *spammer* pode enviar mensagens que contenham propositalmente palavras frequentes em e-mails que não são *spam,* deste modo proporcionalmente estas palavras gerariam falsos positivos, se essas mensagens forem incluídas na base de treinamento. Esta técnica conhecida como *evasion* [78], costuma ser utilizada em outras aplicações relacionadas à segurança, como sistemas IDS (*Intrusion Detection System*) [79].

Embora não haja um consenso sobre qual é a melhor técnica de reconhecimento de padrões para classificação de e-mails, porque mensagens de *spam* não se enquadram em alguns casos muito específicos (e.g. técnicas da seção 4.2), várias abordagens desta área conseguem classificar corretamente a maioria das mensagens, justificando sua aplicação nas ferramentas disponíveis atualmente.

As técnicas baseadas em reconhecimento de padrões geralmente são indiferentes em relação às técnicas de *spam* apresentadas na seção 4.3.1, porém estão fortemente relacionadas às técnicas da seção 4.3.2. Como a aprendizagem de máquina se tornou algo popular entre as ferramentas *antispam*, várias das técnicas da seção 4.3.2 foram criadas especialmente para prejudicar a performance dos classificadores, aumentando o número de falsos positivos e falsos negativos.

4.4.3. Técnicas de detecção de spam baseadas em redes sociais

Algumas abordagens mais recentes de detecção de spam vêm explorando o sucesso crescente das redes sociais para obter informações que possam ajudar na classificação de mensagens. Li e Shen [80] sugerem que as informações obtidas em redes sociais (e.g. assuntos de interesse, esportes, religião, política etc) podem servir de guia para decidir se as informações contidas em um determinado e-mail devem ser consideradas *spam* para um usuário ou não. Um e-mail com propaganda usa as palavras-chave "perca peso", este tema pode ser considerado *spam* para um grupo e muito interesse para outro grupo com obesos, por exemplo. Assim, o interesse manifestado nas redes sociais ajudaria a separar os dois grupos e classificar corretamente os e-mails para cada caso.

A abordagem aplicada em MailRank [81] não utiliza uma rede social específica para a coleta de informações, mas propõe a criação de uma espécie de "rede de usuários de e-mail", que seria uma forma de identificar quem se comunica com quem, formando uma espécie de rede social de comunicação baseada e-mails. Através de um algoritmo de reputação, um servidor central identifica quais usuários são ou não são *spammers*. Por exemplo, se um usuário possui uma reputação alta (não é *spammer*) e encaminha um e-mail para um outro usuário, o destinatário pode ganhar uma pontuação que o faz ser rotulado como não-*spammer*. Embora interessante, o ponto fraco da proposta é que os resultados apresentados são baseados em um cenário simulado. Como esse ambiente não simula alguns problemas que podem existir em um cenário real (e.g. propagação de *malwares*, *bots* etc.) os resultados podem não ser muito conclusivos.

Do mesmo modo que as informações obtidas através de redes sociais podem ser úteis no combate ao *spam*, *spammers* ou *phishers*, pode-se fazer uso dessas informações para realizar ataques direcionados. Informações simples como orientação sexual e até mesmo data de aniversário do usuário podem disparar um e-mail com felicitações e propagar spam ou *malwares*. Um estudo realizado na Universidade de Michigan utilizou informações de 7 mil usuários do Facebook, identificou que muitos usuários que

possuíam um perfil com informações restritas, possuem perfis em outras redes sociais que disponibilizam publicamente algumas informações. Os resultados mostraram que um *spammer* poderia atingir 85% dos usuários com um ataque direcionado, iniciando pelo Facebook e utilizando outras redes sociais, conforme for o seu interesse [82].

Embora as redes sociais tenham informações relevantes para o combate ao *spam*, ainda são muito recentes se comparado ao serviço de e-mail, que se tornou essencial à maioria das pessoas há muito tempo. Nem todos os usuários da Internet possuem cadastro em redes sociais, assim em muitos casos não é possível a coleta dessas informações. Além disso, a grande variedade de redes sociais, com diferentes finalidades, diferentes formas de funcionamento e identificação na mesma, isto tudo pode se tornar um complicador quando for se buscar essas informações.

4.4.4. Técnicas de detecção de spam de imagem

Conforme apresentado na seção 4.3.2, o *spam* de imagem é uma técnica utilizada pelos *spammers* para disfarçar mensagens de texto usando imagens e, portanto, tornando impossível a sua interpretação por classificadores textuais. Uma das soluções para esse tipo de problema é o uso de técnicas de OCR (*Optical Character Recognition*) – Reconhecimento Ótico de Caracteres, que faz a conversão de imagens em texto (com uma fase pré-processamento) e depois as analisa usando classificador textual.

As técnicas utilizadas para tratar *spam* de imagem também estão relacionadas à área de reconhecimento de padrões (seção 4.4.2), porém utilizam técnicas mais específicas. Estas técnicas estão relacionadas principalmente ao pré-processamento das imagens, que deve ser eficiente, no intuito de facilitar o reconhecimento dos caracteres que serão entrada de um classificador de *spam* em formato textual.

Uma das técnicas mais utilizadas para a detecção de spam de imagem é o histograma [10, 17, 18, 19], representação gráfica da distribuição dos dados no espectro das tonalidades da imagem. Em imagens de *spam* poderia ser utilizado um histograma de cores, por exemplo, identificando padrões de histogramas para imagens de *spam* e para imagens de e-mails legítimos. Há ainda abordagens que exploram outros atributos bem específicos das imagens, como será apresentado a seguir.

Li et al. [10] apontam como principal problema a detecção de imagens com um fundo mais complexo (e.g. texto sobreposto a uma fotografia), pois nesses casos as características globais da imagem, tais como cor, textura e formato, são similares a imagens de e-mails legítimos e, portanto, diminuindo a taxa de detecção. De acordo com os autores, embora as características globais possam mudar depois do uso das técnicas de ofuscamento utilizadas pelos *spammers,* as características locais (SIFT - *Scale-Invariant Feature Transform,* MSER - *Maximally Stable Extremal Regions, Gabor wavelet, etc)*, permanecem inalteradas. Logo, as características locais que focam em detalhes da extração de características da imagem são essenciais para a detecção de *spam* de imagem.

Biggio, B. e seus colegas [20] consideram que o texto contido em uma imagem de *spam* possui a mesma essência de um texto contido em um *spam* textual. Logo, se for resolvido o problema de extração do texto da imagem, na sequência é possível aplicar técnicas de detecção de *spam* textual (e.g. filtro bayesiano) que a taxa de acerto será boa. A abordagem proposta possui uma etapa de pré-processamento onde o texto inserido nas imagens é convertido em texto puro. A etapa de treinamento é realizada após o pré-processamento – imagens convertidas em texto. Os resultados mostram que esse tipo de estratégia é eficiente somente nos casos em que as imagens não tiveram algum tipo de ofuscamento. Para as mensagens com ofuscamento, a detecção do *spam* é feita

considerando que a imagem que oferece algum tipo de dificuldade para o OCR é *spam.* O parâmetro utilizado para detectar a dificuldade imposta ao OCR foi medido através da "complexidade perimétrica" da imagem.

Os autores Biggio, B. e seus colegas [21], em outro artigo, consideram que as abordagens da literatura para *spam* de imagem estão focadas somente no tratamento das imagens do *spam,* não se preocupando na identificação das imagens ofuscadas. A proposta apresentada é uma continuação do trabalho anterior [20], onde o foco maior está na identificação dessas imagens. A identificação é feita com base num valor de "ruído", linearizando numa escala de 0 (sem ruído) a 1 (com muito ruído). A novidade em relação à abordagem anterior é que a complexidade perimétrica, que geralmente é utilizada para uma imagem inteira, passa a ser utilizada em diversos pontos da imagem, sendo possível identificar caracteres quebrados ou agregados ao ruído. Foram identificados padrões para (i) caracteres sem ruído; (ii) caracteres quebrados com pouco ruído no fundo; (iii) dois ou mais caracteres conectados pelo ruído e (iv) imagens nas quais o texto é colocado em cima de um fundo irregular.

Por ser um problema complexo, existe uma grande variedade de propostas para solucionar o *spam* de imagem, cada uma tentando e explorando algum aspecto específico do problema. Existem ainda, abordagens que propõem a exploração de ambas as características, textuais e das imagens [17]. De um modo geral, pode-se dizer o *spam* de imagem (seção 4.3.2) é um dos tipos mais explorados por *spammers* e estudados atualmente. As justificativas para isso podem ser o grande percentual de ocorrência desses e-mails em relação ao total global de *spam* (alta relevância do problema) e a grande variedade de técnicas de reconhecimento de padrões utilizadas para o processamento de imagens. Além disso, diferente de outras técnicas utilizadas na área, as quais visam a detecção de *spam* em geral, as abordagens existentes para a detecção deste tipo de *spam* geralmente buscam combater uma técnica bem específica de disseminação de *spam* (seção. 4.3.2.d).

4.4.5. Outras técnicas de detecção de spam

Há ainda algumas abordagens onde a principal contribuição não está totalmente relacionada a uma das categorias apresentadas anteriormente. Por exemplo, abordagens onde o principal fator para a detecção está relacionado a uma questão de infraestrutura ou baseado na colaboração de usuários ou outros servidores, em alguns casos utilizando uma das técnicas apresentadas nas seções anteriores.

Liu e seus colegas [83] propuseram uma infraestrutura *antispam* baseada em *grid* computacional. A abordagem considera que um e-mail é *spam* somente se for enviado para um número grande de destinatários, contabilizando o número de pessoas que receberam o e-mail em uma base denominada *CopyRank.* Um grupo de servidores é configurado para atrair mensagens de *spam*, que passam por um filtro bayesiano distribuído que encaminha as informações aos computadores clientes. Supõe-se que a taxa de falsos positivos será baixa uma vez que somente e-mails com um *CopyRank* alto poderão ser classificados como *spam.* A *grid* funciona com um *plugin* instalado no MUA (*Mail User Agent*), que se conecta com o servidor mais próximo da *grid* toda vez que recebe um e-mail. O cliente encaminha um *checksum* do e-mail recebido, e com base nele o servidor atribui um *CopyRank.* Além do *CopyRank*, o filtro bayesiano faz uma análise que indica se o e-mail é ou não *spam.* Os servidores trabalham de forma cooperada para manter uma tabela de *CopyRanks.*

Além de propostas como a que foi apresentada por Liu et al., utilizando reconhecimento de padrões, uma base de reputação e *grid computacional*, há outras

abordagens que são essencialmente baseadas em redes P2P e na colaboração dos usuários para reportar *spam* [14, 15].

Apesar destas técnicas serem inovadoras, assim como as apresentadas anteriormente, tem bom desempenho enquanto não forem conhecidas pelos *spammers*. Depois que se tornam publicas as técnicas perdem eficácia porque os *spammers* inventam maneiras de burlar seus mecanismos de identificação do *spam*. No caso de técnicas baseadas em reconhecimento de padrões, por exemplo, os modelos podem ser regerados, porém na prática a situação parece ser uma batalha interminável entre os desenvolvedores de técnicas de detecção de *spam* e os *spammers*.

4.4.6. Considerações acerca das técnicas de detecção de spam

Conforme apresentado nas seções 4.4.1 a 4.4.5, existe uma grande variedade de técnicas que buscam o combate ao *spam*. Essas técnicas, com exceção daquelas que são um objeto de pesquisas científicas mais recentes, normalmente estão ou podem estar incorporadas a ferramentas *antispam* utilizadas pelos administradores de sistema de e-mail. O *SpamAssassin*, uma das ferramentas *antispam* mais conhecidas, utiliza redes neurais e métodos estatísticos de classificação bayesiana para atribuir um *score* que representa a probabilidade de um e-mail ser *spam* ou não [43]. A decisão é baseada num limiar padrão (*threshold*) que também pode ser ajustado pelo administrador do serviço de e-mail. Adicionalmente, também são utilizadas regras para auxiliar a classificação dos e-mails.

A Tabela 4.3 resume as principais vantagens e desvantagens das técnicas de detecção de *spam* apresentadas nesta seção. As vantagens e desvantagens apresentadas na tabela não estão relacionadas somente à eficiência dessas técnicas em relação as técnicas de disseminação de *spam* apresentadas na seção 4.3. Mas, podem apresentar vantagens e desvantagens relacionadas a aspectos funcionais, como dificuldade de gerenciamento ou custo computacional.

Após já terem sido abordadas as principais técnicas utilizadas pelos *spammers* e as principais técnicas disponíveis para detecção do *spam*, já é possível perceber que nenhuma técnica de detecção é capaz de funcionar bem isoladamente. Uma análise mais detalhada sobre a relação de técnicas de disseminação de spam e as técnicas de detecção será apresentada na seção 4.5.

Tabela 4.3. Resumo das técnicas de detecção de spam

Técnica	Vantagens	Desvantagens
Whitelists e Blacklists	• Pode agilizar o processo de entrega da mensagem em casos de não-*spam*. • Pode agilizar o processo de recusa da mensagem em casos de *spam*. • Reduz o custo computacional necessário para analisar a mensagem. • É eficiente contra fontes conhecidas de *spam*.	• As *whitelists* podem significar um risco para a segurança do servidor. • Seu uso possui muitas limitações. • É ineficiente quando o remetente ou IP de origem da mensagem é substituído. • Impõe dificuldade de gerenciamento em servidores com grande fluxo de e-mails.
Filtros baseados em palavras-chave	• Útil em casos em que o classificador falha em classificar um determinado *spam* de e-mail. • Sua eficiência pode ser potencializada com o uso de expressões regulares.	• Uso deve ser feito com cautela, a fim de evitar o bloqueio indevido de mensagens. • Uso da técnica deve ser considerado somente em último caso ou de maneira paliativa.

Técnica	Vantagens	Desvantagens
Greylisting	• Técnica eficiente quando o *spammer* não realiza o reenvio dos e-mails com falha na entrega.	• Perda de e-mails legítimos, quando o MTA de origem não faz o reenvio de mensagens. • Possibilita atrasos na entrega de e-mails (entre o primeiro e segundo envio). • É ineficiente contra máquinas de envio de *spam* mais robustas.
SPF	• Útil contra alguns casos de *phishing* de e-mail.	• Ineficiente quando o *spammer* possui um registro SPF. • Frequente ausência de registros SPF publicados no DNS.
Técnicas baseadas em assinatura	• Permitem confirmar a autenticidade da mensagem. • Útil contra alguns casos de *phishing* de e-mail. • Algumas técnicas permitem atestar a integridade do conteúdo do e-mail.	• Falta de padronização da assinatura no serviço de e-mail. • Indisponível em vários servidores de e-mail, dado que o protocolo SMTP, por padrão, não exige o uso de uma técnica baseada em assinatura. • Aumenta o custo computacional para envio do e-mail tanto na origem quanto no destino da mensagem. • Não garante que uma mensagem autenticada seja de fonte confiável.
Técnicas baseadas em reconhecimento de padrões	• Estão entre as técnicas mais utilizadas na detecção de *spam* e mais pesquisadas na literatura. • Apresenta variedade de técnicas que podem ser utilizadas. • Conseguem bons resultados na classificação, se comparado à outras técnicas.	• Os *spammers* costumam explorar as carências de algumas das etapas da classificação para não serem detectados por esse tipo de técnica. • A maioria das técnicas necessita de uma etapa de aprendizagem.
Técnicas baseadas em redes sociais	• Podem explorar o sucesso crescente das redes sociais para auxiliar a detecção de *spam*. • Algumas abordagens possibilitam a classificação das mensagens com base na percepção e interesse do usuário sobre determinados assuntos.	• Nem todos os usuários possuem cadastros em redes sociais, impossibilitando a coleta de informações em alguns casos. • Variedade de redes sociais aumenta a complexidade da busca por informações. • Tipo de técnica pouco explorada se comparada às demais.
Técnicas de detecção de spam de imagem	• Também estão relacionadas à área de reconhecimento de padrões (área muito explorada na Ciência da Computação). • Estão entre as técnicas mais exploradas na literatura.	• Focam somente em um tipo específico de *spam*. • Estão sujeitas a técnicas de ofuscamento de imagens.
Outras técnicas de detecção de spam	• Flexibilidade na criação de novas técnicas. • Podem utilizar aspectos de infraestrutura, tais como redes P2P, *grid* computacional, colaboração de usuários na classificação etc.	• Podem ser facilmente burladas pelos *spammers*, inviabilizando sua proposta, uma vez que normalmente não podem ser facilmente adaptadas quando há mudança nas técnicas de *spam*.

4.5. A eficiência das técnicas de detecção

Esta seção apresenta uma análise da relação entre as técnicas de disseminação de *spam* e as estratégias que podem ser utilizadas para mitigar o problema. Conforme mencionado anteriormente, o objetivo é fazer essa análise sempre a partir da técnica de disseminação em vez de utilizar as técnicas de detecção como ponto de partida, em contraposição ao que é convencionalmente feito na literatura [4, 22, 23, 24, 25, 26].

Esta análise, a partir do ponto de vista da técnica de disseminação, visa oferece vantagens em relação a abordagem tradicional. Pois, quando a análise é realizada a partir da técnica de detecção, normalmente não é apresentado previamente a grande variedade de técnicas que os *spammers* costumam utilizar no envio dos seus e-mails, causando a falsa impressão de que a técnica de detecção conseguirá mitigar o problema como um todo. Porém, quando a análise é realizada da ótica da técnica utilizada pelo *spammer*, é possível identificar quais técnicas *antispam* existentes podem ser eficientes contra aquele *spam* e quais são ineficazes ou podem falhar.

4.5.1 Análise das técnicas

As técnicas de disseminação (TD) mais importantes serão analisadas a seguir, considerando-se as técnicas *antispam* (TA) que podem ser utilizadas em cada caso. A relação entre as técnicas é avaliada com um *score* que vai de 1 (☆) até 5 (☆☆☆☆☆), sendo o seguinte o significado dos *scores.*

Tabela 4.4. Scores utilizados na avaliação das técnicas *antispam*.

☆	A TA é totalmente ineficiente em relação a TD.
☆☆	A TA é pouco eficiente em relação a TD.
★★★	A TA é neutra em relação a TD.
☆☆☆☆	A TA é muito eficiente em relação a TD.
☆☆☆☆☆	A TA é totalmente eficiente em relação a TD.

A Tabela 4.5 sumariza a relação entre as técnicas de disseminação (TD) apresentadas na seção 4.3 às técnicas *antispam* (TA) apresentadas na seção 4.4.

Em alguns casos, a TA não tem influência positiva nem negativa em relação à técnica de disseminação do *spam*. Esses casos (destacados com as estrelas sólidas) receberam três estrelas. Por exemplo, as técnicas baseadas em reconhecimento de padrões (e neste caso inclui-se a detecção de *spam* de imagem) não tem foco na forma de envio da mensagem, mas receberam três estrelas (avaliação neutra) devido a sua eficiência na detecção do conteúdo da mensagem, independente da forma de envio. Ou seja, a avaliação com uma ou duas estrelas (TA totalmente ou pouco ineficiente) seria imprecisa, pois haverá casos em que esta será bem-sucedida, independente da técnica de envio. Já a técnica *greylisting*, por exemplo, não tem foco no conteúdo do e-mail mas poderá ser eficiente dependendo da técnica de envio.

Há outros casos em que a técnica baseada no conteúdo normalmente está associada à forma de envio. Por exemplo, e-mails com o conteúdo falso normalmente estão associados às formas de envio que facilitam o anonimato (e.g. através de *botnets*). Nessas situações, as técnicas são avaliadas caso a caso ao invés de simplesmente receberem a avaliação neutra.

a) **Mecanismo simples de envio**

Os sistemas utilizados para disseminação de *spam* que se enquadram nesta categoria se caracterizam pelo envio de e-mails através de mecanismos pouco robustos, sem tratamento de erros ou reenvio de mensagens em caso de falha na entrega. Estas características fazem com que o *greylisting* seja muito eficiente contra este tipo de *spam*. Contudo, essa técnica não recebeu cinco estrelas pois está sujeita a falhas em algumas situações. O *greylisting* pode falhar por haver MTAs de domínios de organizações confiáveis que não estão configurados devidamente para tratar o reenvio no caso de uma recusa inicial de um e-mail. Isto causaria o não recebimento de um e-mail que não é *spam*.

O SPF não poderá ser utilizado de maneira isolada para bloquear a mensagem, que pode ser de uma origem confiável, provavelmente por não ter seus registros publicados no DNS. Da mesma forma, o uso de assinaturas digitais também não é totalmente eficiente, pois depende que o remetente utilize tal recurso. Além disso, o SPF e a assinatura digital possuem foco na autenticação da mensagem, e não em características específicas dos mecanismos simples de envio.

O bloqueio através de *blacklists* funcionará somente após a identificação da origem do *spam*, e o bloqueio por palavras-chave só será útil após o conhecimento do conteúdo da mensagem, que poderá ser alterado propositalmente pelo *spammer* em pouco tempo.

b) **Envio com tratamento de erros**

Quando o sistema de envio de *spam* é capaz de reenviar mensagens em caso de falha na entrega, técnicas como o *greylisting* se tornam totalmente ineficientes. No caso do SPF, alguns servidores podem possuir seus registros publicados no DNS, inviabilizando o uso da técnica. *Blacklists* poderiam ser muito eficientes contra as fontes conhecidas de *spam*, uma vez que nesta categoria os servidores de origem do *spam* não costumam mudar com tanta frequência, porém, pode haver exceções. O bloqueio por palavras-chave só terá utilidade após o conhecimento do conteúdo da mensagem, o que ocorre somente depois que boa parte do *spam* é recebido. No caso das assinaturas digitais, como alguns servidores podem pertencer a organizações confiáveis, nada impede que o e-mail seja assinado digitalmente pela empresa responsável pelo envio, o que não muda a possibilidade da mensagem ser *spam*.

c) **Substituição de remetente**

Quando o *spammer* utiliza artifícios como a substituição (forja) do remetente, técnicas como *blacklists* que utilizam o nome do domínio do remetente ou o endereço completo do e-mail do remetente, são totalmente ineficientes, uma vez que o *spammer* pode trocar o endereço do remetente quantas vezes quiser e utilizar um domínio de uma entidade confiável (que não deve ser bloqueado) para parecer um e-mail legítimo. O *greylisting* também é indiferente em relação a esta técnica, podendo falhar em muitos casos.

Por outro lado, técnicas como assinatura digital e SPF podem ser muito eficientes contra este tipo de *spam*, falhando apenas em alguns casos. Por exemplo, se o MTA de destino recebe um *phishing* que se apresenta como sendo de uma organização confiável (e.g. Bancos, órgãos governamentais etc.), a TA pode atestar a validade do remetente da mensagem através da chave pública ou do registro SPF que foi publicado no DNS. Contudo, estas técnicas não receberam cinco estrelas, pois dependem que o remetente (ou o domínio de uma organização confiável, no caso do *phishing*) assine as mensagens, ou tenha seus registros SPF publicados. Além disso, as vítimas de *phishing* também podem

ser enganadas quando é utilizado um domínio muito parecido com o legítimo (e.g. playpal.com em vez de paypal.com).

O bloqueio por palavras-chave, assim como no *envio com tratamento de erros*, só terá utilidade após o conhecimento do conteúdo da mensagem, normalmente depois que o MTA de destino já recebeu boa quantidade do *spam*. Neste caso, entretanto, a gravidade é bem maior pois pode se tratar de uma ameaça como o *phishing.*

d) **Envio através de *botnets***

O envio através de *botnets* tem muitas semelhanças com o mecanismo simples de envio, porém, com um agravante "o bloqueio por *blacklists* se torna ainda menos eficiente visto que a origem das mensagens pode mudar mais ainda". Isto acontece porque uma *botnet* é uma rede de grande escalabilidade, formada por computadores comprometidos de usuários espalhados pela Internet. O bloqueio por palavras-chave só terá utilidade após o conhecimento do conteúdo da mensagem, após muito conteúdo *spam* ter sido recebido. Por outro lado, a técnica *greylisting* é muito eficiente contra esse tipo de *spam*, visto que os mecanismos de envio utilizados pelos *spammers* possuem as mesmas características dos mecanismos simples de envio, ou seja, não tratam o reenvio da mensagem.

O uso de assinaturas digitais e SPF pode possuir boa eficiência contra este tipo de envio, que é característico de mensagens como o *phishing*, onde o *spammer* encaminha seus e-mails fraudulentos através de computadores comprometidos que estão espalhados pela Internet, com o objetivo de dificultar a sua localização.

e) **Envio através de *open relays***

Quando se trata de *spam* enviado através de *open relays*, o uso de *blacklists* se torna complicado pois o bloqueio de um IP do MTA de origem pode ocasionar também o bloqueio de todas as mensagens de uma organização confiável. O bloqueio de palavras-chave, como de costume, possui pouca eficiência se utilizada isoladamente e só deverá ser considerado quando todas as outras técnicas falharem. O *greylisting* terá pouca eficiência visto que a *open relay*, que é um servidor que normalmente pertence a uma organização confiável, normalmente é capaz de reenviar a mensagem após uma recusa inicial. O SPF e a assinatura digital só terão utilidade nos casos já previstos anteriormente, quando há a substituição de remetente e todos os pares da comunicação utilizam tais recursos.

f) **Inserção proposital de palavras em mensagens de e-mail**

Quando o *spammer* utiliza recursos textuais para enganar os classificadores de e-mails, normalmente o envio das mensagens é mal-intencionado. Ou seja, não se trata de um *spam* que permite que o usuário opte por não receber ou de uma simples divulgação com fins publicitários, feita por uma organização confiável. Nesse caso, a técnica de *spam* baseada no conteúdo dificilmente utiliza somente recursos textuais ou visuais, podendo vir acompanhada de alguma característica que permita a variação do endereço de origem da mensagem, a fim de dificultar ainda mais a sua detecção. Assim, o uso de *blacklists* passa a ser pouco eficiente em quase todos os casos em que o *spammer* explora os recursos disponíveis no corpo da mensagem. O bloqueio por palavras-chave só terá utilidade após o recebimento (conhecimento) do conteúdo da mensagem.

As técnicas baseadas em detecção de imagem são totalmente ineficientes contra este tipo de técnica, visto que neste caso o *spammer* usa subterfúgios técnicos para burlar os mecanismos *antispam*, usando aspectos textuais da mensagem. Já as técnicas de reconhecimento de padrões com foco no texto da mensagem passam a perder sua eficiência quando este tipo de técnica é utilizado.

g) **Troca ou inserção de caracteres na mensagem de e-mail**

A troca ou inserção intencional de caracteres no e-mail provoca uma queda ainda maior na performance dos classificadores textuais baseados em reconhecimento de padrões. Pois, as palavras que são utilizadas como características para determinar se um conteúdo é *spam*, normalmente são aquelas que são modificadas pelos *spammers*.

A existência desse tipo de técnica de inserção intencional de caracteres, não faz com que a área de reconhecimento de padrões seja desconsiderada na classificação textual das mensagens. A redução na sua eficiência ocorre principalmente nos classificadores tradicionais (e.g. classificadores bayesianos e redes neurais), mas há trabalhos que buscam o aprimoramento dessas técnicas contra este tipo específico de *spam* [67, 68, 69, 73, 74, 75].

h) **Conteúdo falso**

No caso de e-mails com conteúdo falso, como o *phishing*, as técnicas tradicionais de classificação textual são pouco eficientes, pois o conteúdo da mensagem se parece muito com uma mensagem real. Alguns trabalhos buscam utilizar outros aspectos não-textuais da mensagem como característica, objetivando a classificação desse tipo de *spam* [2, 84, 85, 86, 87] através de técnicas de reconhecimento de padrões.

No caso do *phishing*, técnicas como o SPF e uso de assinatura digital podem ser muito eficientes contra este tipo de *spam* se as organizações confiáveis, às quais o *phisher* tenta personificar na sua mensagem, fazerem uso de tais recursos. Nesse caso, o receptor da mensagem poderia atestar que o e-mail não foi enviado pela organização que consta como rementem no e-mail.

i) **Uso de imagens**

O uso de texto embutido em imagens inviabiliza qualquer tipo de técnica baseada no texto da mensagem. Por esse motivo, os filtros baseados em palavras-chave passam a ser totalmente ineficientes. As técnicas de reconhecimento de padrões baseadas no processamento textual também passam a ser totalmente ineficientes, exceto se houver uma etapa de pré-processamento que extraia o texto da imagem para então submetê-lo às técnicas de classificação textual.

Já as técnicas específicas para a detecção de *spam* de imagem são muito eficientes contra este tipo de técnica. Os trabalhos presentes na literatura apresentam abordagens que vão desde o pré-processamento da mensagem (inclusive para casos de ofuscação da imagem), para posterior classificação textual através de técnicas tradicionais, até o reconhecimento do *spam* pelas características identificadas na própria imagem que contém o texto embutido.

j) **Uso de recursos HTML**

O uso de recursos da linguagem HTML pode ser mal-intencionado em muitos casos (e.g. *phishing*). Assim, a eficiência das técnicas de reconhecimento de padrões é a mesma da TD conteúdo falso. Alguns trabalhos na literatura exploram características de e-mails no formato HTML para realizar a classificação das mensagens [2, 84, 85, 86, 87].

Quando há o uso mal-intencionado de recursos HTML, normalmente há também outras características que ocorrem em e-mails fraudulentos, como a falsificação do remetente da mensagem. Técnicas como o SPF com a assinatura digital possuem muita eficiência nesses casos.

k) **E-mail marketing**

O e-mail marketing é um tipo de *spam* muito complicado de classificar. Porque normalmente é originado em servidores MTA de organizações confiáveis com fins publicitários (que normalmente não mudam de endereço com frequência ou tentam esconder a sua localização); o uso de *blacklists* pode ser um pouco mais eficiente contra esse tipo de e-mail. Porém, alguns usuários podem ter interesse em receber os e-mails de algumas organizações (e.g. sites de e-commerce) e o uso de *blacklists* deve ser considerado em último caso.

O bloqueio por palavras-chave passa a ser totalmente ineficiente, pois há o agravante de determinadas palavras presentes em algum *spam* específico também estarem presentes em um e-mail publicitário, que é do interesse dos usuários, causando o bloqueio indevido da mensagem. Técnicas baseadas na autenticação da mensagem (SPF, *greylisting* e assinaturas digitais) são pouco eficientes, já que são recursos que também podem ser utilizados pelo *spammer*.

As técnicas baseadas em reconhecimento de padrões, especificamente aquelas baseadas nas características textuais, costumam ter bons resultados na classificação dos e-mails. Há ferramentas que conseguem, inclusive, classificar boa parte dos e-mails em três categorias: *não-spam*, *spam*, e *e-mail marketing*, deixando a decisão quanto aos e-mails desta última categoria a critério do usuário final. Técnicas baseadas em redes sociais também podem ser de grande utilidade, já que essas redes podem fornecer informações sobre os assuntos de interesse do usuário.

Tabela 4.5. Resumo da relação ente TD e TA.

TA / TD		Blacklists	Palavras-chave	Greylisting	SPF	Assinatura Digital	Reconhecimento de Padrões	Redes Sociais	Spam de Imagem
Técnicas baseadas no envio do *spam*	**1. Mecanismo simples de envio**	☆☆☆	☆☆	☆☆☆☆	☆☆☆	☆☆☆	★★★	★★★	★★★
	2. Envio com tratamento de erros	☆☆☆	☆☆	☆	☆☆☆	☆☆☆	★★★	★★★	★★★
	3. Substituição de remetente	☆	☆☆	☆☆☆	☆☆☆☆	☆☆☆☆	★★★	★★★	★★★
	4. Envio através de *botnets*	☆☆	☆☆	☆☆☆☆	☆☆☆☆	☆☆☆☆	★★★	★★★	★★★
	5. Envio através de *open relays*	☆☆	☆☆	☆☆	☆☆☆	☆☆☆	★★★	★★★	★★★

TA / TD		Blacklists	Palavras-chave	Greylisting	SPF	Assinatura Digital	Reconhecimento de Padrões	Redes Sociais	Spam de Imagem
Técnicas baseadas no conteúdo do e-mail	6. Inserção proposital de palavras	☆☆	☆☆	★★★	★★★	★★★	☆☆☆	☆☆	☆
	7. Troca ou inserção de caracteres	☆☆	☆☆	★★★	★★★	★★★	☆☆	☆☆	☆
	8. Conteúdo falso	☆☆	☆☆	★★★	☆☆☆☆	☆☆☆☆	☆☆☆	☆☆	☆
	9. Uso de imagens	☆☆	☆	★★★	★★★	★★★	☆	☆☆	☆☆☆☆
	10. Uso de recursos HTML	☆☆	☆☆	★★★	☆☆☆	☆☆☆	☆☆☆	☆☆	☆
	11. E-mail marketing	☆☆☆	☆	☆☆	☆☆	☆☆	☆☆☆☆	☆☆☆☆	☆

4.5.1 Considerações importantes sobre as técnicas analisadas

Um fato importante em relação ao *spam* é que não existe uma técnica de detecção perfeita, mesmo que a técnica busque combater um tipo específico de *spam*, pois sempre haverá um caso em que a técnica poderá falhar. Por exemplo, o *greylisting*, técnica amplamente utilizada, é eficaz contra os mecanismos de envio de *spam* que não tratam o reenvio da mensagem. Entretanto, o *greylisting* pode ocasionar a recusa indevida da mensagem, mesmo de MTAs legítimos que, por uma questão estratégica ou má configuração, acabam não tratando o reenvio.

Na avaliação de qualquer tipo de técnica *antispam*, além de avaliar a detecção das mensagens de *spam*, também se faz necessário a avaliação do comportamento da técnica em relação a e-mails que não são *spam*. As técnicas de detecção podem falhar, tanto na classificação do *spam* como não-*spam*, quanto na classificação de um e-mail não-*spam* como *spam*. Para que uma técnica possa ser considerada eficiente, mesmo que seja apenas contra um tipo específico de *spam*, deverá possuir uma possibilidade nula de ocorrência de falsos positivos e falsos negativos, ao mesmo tempo que uma taxa de altíssima de verdadeiros positivos e verdadeiros negativos. Por esse motivo, nenhuma das técnicas foi classificada como totalmente eficiente.

Algumas técnicas não chegaram a ser avaliadas sequer como muito eficientes, como no caso das *blacklists* e palavras-chave. Isto não desqualifica essas técnicas a ponto do seu uso ser desconsiderado. Porém, essas técnicas podem ser utilizadas como ferramentas auxiliares na detecção de *spam*, o que deve ser feito com bastante cautela. Já técnicas baseadas na autenticação das mensagens, como assinaturas digitais e SPF, que em alguns casos não receberam cinco estrelas porque sua eficácia depende da sua adoção

por terceiros (e não somente pelo MTA de destino), podem ser úteis quando todos os pares da comunicação utilizam a técnica.

Para as técnicas baseadas em informações extraídas de *redes sociais*, abordagem muito recente e ainda com poucas evidências de sucesso, a avaliação foi considerada *pouco eficiente* em quase todos os casos, exceto para o item 11 da tabela (*e-mail marketing*), visto que as informações obtidas nessas redes podem dizer muito a respeito dos assuntos de interesse do usuário.

A análise das técnicas baseadas em reconhecimento de padrões é a que deve ser interpretada com mais cautela. Apesar de ter sido classificada como muito eficiente somente em um dos casos, ainda é a mais utilizada e uma das mais eficientes na detecção de *spam*, com inúmeras possibilidades de aplicação ainda não exploradas dentro do problema em questão. Ou seja, suas técnicas estão entre as mais eficientes para a detecção do *spam* em geral, apresentando uma ampla gama de possibilidades de técnicas que ainda podem ser aplicadas para mitigar o problema. Ao analisar os *scores* atribuídos em cada avaliação, deve-se considerar que a TD foi avaliada em relação a TA que especificamente visam violar técnicas baseadas em reconhecimento de padrões.

Enfim, o uso das técnicas *antispam* não deve ser de executada de maneira isolada. Ou seja, é necessário que haja uma combinação de técnicas para que o problema seja mitigado da melhor maneira possível. Essa combinação pode ser feita para atender alguma necessidade específica, otimizando os resultados da classificação dos e-mails, ou ainda para adequar o ambiente de detecção a capacidade computacional do servidor de e-mails.

4.6. Conclusões

O envio indiscriminado de mensagens sem o consentimento de seus destinatários, prática conhecida como *spam,* é um problema que ainda está longe de ser solucionado, apesar do e-mail estar presente na vida da maioria das pessoas atualmente. Além do aborrecimento dos usuários, o *spam* pode causar problemas como a geração de custo computacional adicional e despesas com tecnologia e infraestrutura para a detecção desse conteúdo. Após a criação de novas técnicas para a detecção de *spam*, os *spammers* costumam desenvolver novas artimanhas para burlar os mecanismos de classificação dos e-mails, gerando uma situação de competição que parece não ter fim.

Este capítulo apresentou a análise das técnicas *antispam* sob uma nova perspectiva. Primeiramente foram apresentadas as principais técnicas utilizadas na disseminação de *spam* de e-mail. Em seguida foram apresentadas as principais técnicas de detecção existentes na literatura. Com este conhecimento já foi possível avaliar a eficiência de cada técnica *antispam* que foi apresentada. Por último, foram apresentados alguns aspectos relacionados a cada técnica utilizada pelos *spammers* e a eficiência das abordagens para combatê-las. Para cada técnica de detecção de *spam* foi atribuído uma nota (*score*) que representa a sua eficiência em relação a um tipo específico de técnica de disseminação de *spam*.

A análise realizada apresentou diversos resultados interessantes. Por exemplo, ficou evidente que nenhuma técnica *antispam* é eficiente sozinha, pois sempre haverá situações que levarão à falha. As técnicas baseadas em reconhecimento de padrões, observadas nas ferramentas *antispam*, são o alvo comum de ataque dos *spammers* devido ao seu uso e eficiência.

4.7. Referências

[1] Klensin, J. “RFC 2821 - Simple Mail Transfer Protocol”, disponível em: <http://www.ietf.org/rfc/rfc2821.txt>. Acessado em: 29/09/2015.

[2] Olivo, C. K.; Santin, A. O.; Oliveira, L. S. “Obtaining the Threat Model for E-mail Phishing”. em: Applied Soft Computing, vol. 13, issue 12, p.4841-4848. 2013.

[3] Kleiner, K. “Happy Spamyversary! Spam Reaches 30”, disponível em: <http://www.newscientist.com/article/dn13777-happy-spamiversary-spam-reaches-30.html>. Acessado em: 29/09/2015.

[4] Hoanca, B. “How good are our weapons in the spam wars?”, em: IEEE Technology and Society Magazine, vol. 25, Issue 1, p.22-30. 2006.

[5] Whitworth, B.; Whitworth, E. “Spam and the social technical gap”, em: IEEE Computer, vol. 37, Issue 10, p.38-45. 2004.

[6] Symantec “January 2011 Intelligence Report”, disponível em: <http://www.messagelabs.com/mlireport/MLI_2011_01_January_Final_en_us.pdf>. Acessado em junho de 2012.

[7] Symantec “May 2013 Intelligence Report”, disponível em: <http://www.symantec.com/content/en/us/enterprise/other_resources/b-intelligence_report_05-2013.en-us.pdf>. Acessado em: 29/09/2015.

[8] Symantec “Symantec Internet Security Threat Report – volume 19”, disponível em: <www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf>. Acessado em 29/09/2015.

[9] Leyden, J. “Spammers embrace e-mail authentication”, disponível em <http://www.theregister.co.uk/2004/09/03/email_authentication_spam/>. Acessado em: 29/09/2015

[10] Li, P.; Yan, H.; Cui, G.; Du, Y. “Integration of Local and Global Features for Image Spam Filtering”, em: Journal of Computational Information Systems, vol. 8, p.779-789. 2012.

[11] The New York Times, “Spam Doubles, Finding New Ways to Deliver Itself”, disponível em: <http://www.nytimes.com/2006/12/06/technology/06spam.html>. Acessado em: 29/09/2015.

[12] “There are 600,426,974,379,824,381,952 ways to spell Viagra”, disponível em: <http://cockeyed.com/lessons/viagra/viagra.html>. Acessado em: 30/09/2015.

[13] Olivo, C. K.; Santin, A. O.; Oliveira, L. E. S. “Avaliação de Características para Detecção de Phishing de E-mail”, Pontifícia Universidade Católica do Paraná, Curitiba – PR, Brasil. 2010.

[14] Damiani, E.; Vimercati, S.; Paraboschi, S.; Samarati, P. “P2P-based collaborative spam detection and filtering”, em: Proceedings of the Fourth International Conference on Peer-to-Peer Computing, IEEE, p. 176-183. 2004.

[15] Dimmock, N.; Maddison, I., “Peer-to-peer collaborative spam detection”, em: ACM Crossroads Magazine, vol. 11, issue 2, p. 4-4. 2004.

[16] Liu, Q.; Qin, Z.; Cheng, H.; Wan, M., “Efficient Modeling of Spam Images”, em: 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, IEEE, p. 663-666. 2010.

[17] Soranamageswari, M.; Meena, C., “A Novel Approach Towards Image Spam Detection”, em: International Journal of Computer Theory and Engineering, vol. 3, p. 84-88. 2011.

[18] Xu, C.; Chiew, K.; Chen, Y.; Liu, J. , “Fusion of Text and Image Features: A New Approach to Image Spam Filtering”, em: Practical Applications of Intelligent Systems, Advances in Intelligent and Soft Computing, Springer, vol. 124, p. 129-140. 2012.

[19] Gao, Y.; Yang, M.; Zhao, X.; Pardo, B. Wu, Y.; Pappas, T.N.; Choudhary, A., “Image Spam Hunter”, em: International Conference on Acoustics, Speech and Signal Processing, IEEE, p. 1765-1768. 2008.

[20] Biggio, B.; Fumera, G.; Pillai, I.; Roli, F., “Image Spam Filtering Using Visual Information”, em: 14th International Conference on Image Analysis and Processing, IEEE, p. 105-110. 2007.

[21] Biggio, B.; Fumera, G.; Pillai, I.; Roli, , “Image spam filtering by content obscuring detection”, em: Fourth conference on e-mail and antispam. 2007.

[22] Caruana, G.; Li, M., "A survey of emerging approaches to spam filtering", em: ACM Computing Surveys (CSUR), vol. 44, Issue 2. 2012.

[23] Gansterer, W.; Ilger, M.; Lechner, P.; Neumayer, R.; Straub, J., "Anti-Spam Methods – State-of-the-Art", disponível em: <security.taa.univie.ac.at/files/FA384018-1.pdf>. Acessado em 30/09/2015.

[24] Blanzieri, E.; Bryl, A., "A survey of learning-based techniques of email spam filtering", em: Journal Artificial Intelligence Review, vol. 29, Issue 1, p. 63-92. 2008.

[25] Goodman, J.; Cormack, G.; Heckerman, D. "Spam and the ongoing battle for the inbox", em: Communications of the ACM, vol. 50, Issue 2, p. 24-33. 2007.

[26] Wang, X.; Cloete, I., "Learning to classify email: a survey", em: Proceedings of the Fourth Conference on Machine Learning and Cybernetics, vol. 9, p.5716-5719.

[27] “The Postfix Home Page”, disponível em: <http://www.postfix.org/>. Acessado em 30/09/2015.

[28] Bernstein, D. “qmail: Second Most Popular MTA on the Internet”, disponível em: <http://qmail.linorg.usp.br/top.html>. Acessado em 30/09/2015.

[29] “Secure Enterprise Email Solutions for Business | Exchange”, disponível em: <https://products.office.com/pt-br/exchange/email>. Acessado em 30/09/2015.

[30] “Thunderbird – Software Made to Make E-mail Easier”, disponível em: <https://www.mozilla.org/pt-BR/thunderbird/>. Acessado em 30/09/2015.

[31] “Software de E-mail e Calendário Microsoft Outlook”, disponível em: <https://products.office.com/pt-br/outlook/email-and-calendar-software-microsoft-outlook>. Acessado em 30/09/2015.

[32] Crispin, M. “RFC 3501 – Internet Message Access Protocol – Version 4rev1”, disponível em: <https://tools.ietf.org/rfc/rfc3501.txt>. Acessado em 30/09/2015.

[33] Myers, J.; Rose, M. “RFC 1939 – Post Office Protocol – Version 3”, disponível em: <https://www.ietf.org/rfc/rfc1939.txt>. Acessado em 30/09/2015.

[34] Freed, N.; Borenstein, I., “RFC 2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies”, disponível em: <http://tools.ietf.org/rfc/rfc2045.txt>. Acessado em 30/09/2015.

[35] Crocker, D. “RFC 882 – Standard for the Format of ARPA Internet Text Messages”, disponível em: <http://tools.ietf.org/rfc/rfc822.txt>. Acessado em 30/09/2015.

[36] Vaudreuil, G. “RFC 3463 – Enhaced Mail System Status Codes”, disponível em: <https://tools.ietf.org/rfc/rfc3463.txt>. Acessado em 30/09/2015.

[37] "Código Penal Brasileiro", Título II, Cap. VI, Art. 171, disponível em: <http://www.planalto.gov.br/ccivil_03/Decreto-Lei/Del2848.htm>. Acessado em 30/09/2015.

[38] Wang, P.; Sparks, S.; Zou, C. "An Advanced Hybrid Peer-to-Peer Botnet", em: IEEE Transactions on Dependable And Secure Computing, vol. 7, nº 2. 2010.

[39] Bianchi, N. M., "The Return of the Open Relays", disponível em: <http://www.spamhaus.org/news/article/706/the-return-of-the-open-relays>. Acessado em: 30/09/2015.

[40] Duda, R.; Hart, P.; Stork D., "Pattern Classification", 2ª edição, Wiley-Interscience. 2000.

[41] "The Unicode Standard – Technical Introduction", disponível em: <http://www.unicode.org/standard/principles.html>. Acessado em 30/09/2015.

[42] Liu, C.; Stamm, S. "Fighting Unicode-Obfuscated Spam", em: Proceedings of the Anti-Phishing Working Group - 2nd Annual eCrime Researchers Summit, p. 45-59, ACM. 2007.

[43] "SpamAssassin – The #1 Enterprise Open-Source Spam Filter", disponível em: <http://spamassassin.apache.org/>. Acessado em 24/09/2015.

[44] Jargas, A. M. "Shell Script Profissional", 1ª edição, Editora Novatec LTDA. 2008.

[45] Harrys, E. "The Next Step in Spam Control War: Greylisting", disponível em: <http://projects.puremagic.com/greylisting/whitepaper.html>. Acessado em: 30/09/2015.

[46] "Sender Policy Framework", disponível em: <http://www.openspf.org/>. Acessado em: 30/09/2015.

[47] Levine, J. R. "Experiences with Greylisting", em: Second Conference on e-mail and Anti-Spam. 2005.

[48] Allman, E.; Callas, J.; Delany, M.; Libbey, M.; Fenton, J.; Thomas, M., "RFC 4871 - DomainKeys Identified Mail (DKIM) Signatures", disponível em: <http://www.rfc-editor.org/rfc/rfc4871.txt>. Acessado em: 30/09/2015.

[49] Antispam.br, "Domain Keys Identified Mail (DKIM)", disponível em: <http://antispam.br/admin/dkim/>. Acessado em 30/09/2015.

[50] Hansen, T.; Crocker, D.; Hallam-Baker, P. "RFC 5585 - DomainKeys Identified Mail (DKIM) Service Overview", disponível em: <http://tools.ietf.org/rfc/rfc5585.txt>. Acessado em 30/09/2015.

[51] Linn, J. "Privacy Enhancement for Internet Electronic Mail", disponível em: <https://tools.ietf.org/rfc/rfc989.txt>. Acessado em 30/09/2015.

[52] Callas, J.; Donnerhacke, L.; Finney, H.; Shaw, D.; Thayer, R. "RFC 4880 - OpenPGP Message Format", disponível em: <https://tools.ietf.org/rfc/rfc4880.txt>. Acessado em 30/09/2015.

[53] Crocker, S.; Freed, N.; Galvin, J.; Murphy, S., "RFC 1848 - MIME Object Security Services", disponível em: <https://tools.ietf.org/rfc/rfc1848.txt>. Acessado em 30/09/2015.

[54] Ramsdell, B., "RFC 3851 - Secure/Multipurpose Internet Mail Extensions (S/MIME) - Version 3.1 - Message Specification", disponível em: <https://tools.ietf.org/rfc/rfc3851.txt>. Acessado em 30/09/2015.

[55] Bishop, C. "Pattern Recognition and Machine Learning", 1ª edição, Springer. 2007.

[56] Karthika, D.; Hamsapriya, T.; Raja, M.; Lakshmi, P. "Spam Classification Based on Supervised Learning Using Machine Learning Techniques", em: International Conference on Process Automation, Control and Computing (PACC), IEEE, p. 1-7. 2011.

[57] Schneider, K. "A comparison of event models for Naive Bayes anti-spam e-mail filtering", em: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1, ACM, p. 307-314. 2003.

[58] Androutsopoulos, I.; Paliouras, G.; Michelakis, E. "Learning to Filter Unsolicited Commercial E-Mail", em: NCSR "Demokritos" Technical Report, nº 2004/2, disponível em: <http://nlp.cs.aueb.gr/pubs/TR2004_updated.pdf>. Acessado em: 30/09/2015.

[59] Chen, C.; Tian, Y.; Zhang, C. "Spam Filtering with Several Novel Bayesian Classifiers", em: 19th International Conference on Pattern Recognition, IEEE, p. 1-4. 2008.

[60] Frank, E; Hall, M.; Pfahringer, B. "Locally Weighted Naive Bayes", em: Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence, ACM, p. 249-256. 2002.

[61] Zhang, H; Jiang, L.; Su, J. "Hidden Naive Bayes", em: Proceedings of the 20th national conference on Artificial intelligence - Volume 2, ACM, p. 919-914. 2005.

[62] Webb, G.; Boughton, J.; Wang, Z. "Not so Naive Bayes: Aggregating One-Dependence Estimators", em: Machine Learning, vol. 58, issue 1, p.5-24. 2005.

[63] Drucker, H.; Wu, S.; Vapnik, V. N. "Support Vector Machines for Spam Categorization", em: IEEE Transactions on Neural Networks, vol. 10, issue 5, p. 1048-1054. 1999.

[64] Ma, W.; Tran, D.; Sharma, D. "A Novel Spam e-mail Detection System Based on Negative Selection", em: Fourth International Converence on Computer Science and Convergence Information Technology, IEEE, p. 987-992. 2009.

[65] Blum, A.; Mitchell, T. "Combining labeled and unlabeled data with co-training", em: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, ACM, p.92-100. 1998.

[66] Kiritchenko, S.; Matwin, S. "E-mail Classification with Co-training", em: Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, p. 8. 2001.

[67] Braga, I.; Ladeira, M. "Um modelo adaptativo para a filtragem de spam", em: VI Encontro Nacional de Inteligência Artificial, Rio de Janeiro – RJ, Anais do XXVII Congresso da Sociedade Brasileira de Computação, p. 1381-1390. 2007.

[68] Zhou, Y.; Mulekar, M.; Nerellapalli, P. "Adaptive Spam Filtering Using Dynamic Feature Space", em: 17th IEEE International Conference on Tools with Artificial Intelligence. 2005.

[69] Bratko, A.; Filipič, B. "Spam Filtering using Character-level Markov Models: Experiments for the TREC 2005 Spam Track", em: Proceedings of the 14th Text Retrieval Conference. 2005.

[70] Chhabra, S.; Yerazunis, W.; Siefkes, C. "Spam Filtering Using a Markov Random Field Model with Variable Weighting Shcemas", em: Fourth IEEE International Conference on Data Mining, p. 347-350. 2004.

[71] Ndumiyana, D.; Sakala, L. “Hidden Markov Models and Artificial Neural Networks for Spam Detection”, em: International Journal of Engineering Research & Technology, vol. 2, issue 4. 2013.

[72] Bratko, A.; Cormack, G.; Filipič, B.; Lynam, T.; Zupan, B. “Spam Filtering Using Statistical Data Compression Models”, em: Journal of Machine Learning Research, vol. 7, p. 2673-2698. 2006.

[73] Lee, H.; Ng, A. “Spam Deobfuscation Using a Hidden Markov Model”, em: Second Conference on Email and Anti-Spam. 2005.

[74] Lee, S.; Jeong, I.; Choi, S. “Dynamically Weighted Hidden Markov Model for Spam Deobfuscation”, em: Proceedings of the 20th International Joint Conference on Artificial Intelligence, p. 2523-2529, Morgan Kaufmann Publishers Inc. 2007.

[75] Sculley, D.; Wachman, G.; Brodley, C., “Spam Filtering Using Inexact String Matching in Explicit Feature Space with On-line Linear Classifiers”, em: Proceedings of the 15th Text Retrieval Conference. 2006.

[76] Kiran, R. S. S.; Atmosukarto, I. “Spam or Not Spam – That is the Question”, em: Technical Report, University of Washington. 2005.

[77] Guzella, T.; Caminhas, W. “A Review of Machine Learning Approaches to Spam Filtering”, em: Expert Systems with Applications, Elsevier, vol. 36, issue 7, p.10206-10222. 2009.

[78] Nelson, B.; Rubinstein, B.; Huang, L.; Joseph, A.; Tygar, J. “Classifier Evasion: Models and Open Problems”, em: Privacy and Security Issues in Data Mining and Machine Learning, vol. 6549, Lecture Notes in Computer Science, p. 92-98, Springer. 2011.

[79] Barreno, M.; Nelson, B.; Sears R.; Joseph, A.; Tygar, J. “Can Machine Learning be Secure?”, em: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, p. 16-25. 2006.

[80] Li, Z.; Shen, H. “SOAP: A Social Network Aided Personalized and Effective Spam Filter to Clean Your E-mail Box”, em: IEEE INFOCOM, p. 1835-1843. 2011.

[81] Chirita, P.; Diederich, J.; Nejdl, W. “MailRank: Using Ranking for Spam Detection”, em: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, p. 373-380. 2005.

[82] Brown, G.; Howe, T.; Ihbe, M.; Prakash, A.; Borders, K. “Social Network and Context Aware Spam”, em: Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work, p. 403-412. 2008.

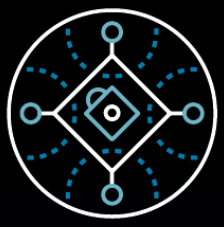
[83] Liu, P.; Chen, G.; Ye, L.; Zhong, W. “Anti-spam grid: a dynamically organized spam filtering infrastructure”, em: Proceedings of the 5th WSEAS International Conference On Simulation, Modeling And Optimization, p. 61-66. 2005.

[84] Chen, J. e Guo, C. “Online Detection and Prevention of Phishing Attacks”, em: Communications and Networking in China, p.19-21. 2006.

[85] Cook, D., Gurbani, V. e Daniluk, M. “Phishwish: A Stateless Phishing Filter Using Minimal Rules”, em: Lecture Notes in Computer Science, p.182-186. 2008.

[86] Fette, I., Sadeh, N. e Tomasic, A. “Learning to Detect Phishing emails”, em: International World Wide Web Conference, p.649-656. 2007.

[87] M. Chandrasekaran, K. Narayanan, S. Upadhyaya “Phishing email Detection Based on Structural Properties”, em: Cyber Security Symposium. 2006.



SBSEG15

XV SIMPÓSIO BRASILEIRO
EM SEGURANÇA DA INFORMAÇÃO
E DE SISTEMAS COMPUTACIONAIS

9 A 12 DE NOVEMBRO | FLORIANÓPOLIS/SC

DIAMANTE | OURO | BRONZE

PATROCÍNIO: nic.br Núcleo de Informação e Coordenação do Ponto BR; cgi.br Comitê Gestor da Internet no Brasil | intel | FAPESC | CAPES | CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico | Google

APOIO: FLORIPA CONVENTION & VISITORS BUREAU; INSTITUTO FEDERAL SANTA CATARINA Câmpus São José; INFOTV

PROMOÇÃO: SBC Sociedade Brasileira de Computação

REALIZAÇÃO: UNIVALI; UFSC

ISBN 978-85-7669-304-8