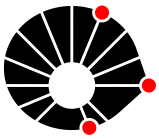

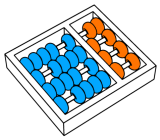

Criptografia como Software Livre

J. Alves *I. C. Garcia*

Relatório Técnico - IC-PFG-21-26
Projeto Final de Graduação
2021 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Criptografia como Software Livre

Jeferson Alves

Orientadora: Islene Calciolari Garcia

Resumo

Com avanço da internet e dos serviços que dela se originaram cresce a necessidade de proteger os dados que são inseridos na rede de computadores. A natureza dos dados varia, desde uma página Web requisitada por usuário que se conecta a uma plataforma de *e-commerce* até os dados sigilosos de seu cartão de crédito, que ele deverá inserir se desejar finalizar o pedido. Essa rotina já é uma realidade, e os desenvolvedores de aplicações precisam sempre garantir a proteção dos dados inseridos através do uso de criptografia. A criptografia em si é um conjunto de princípios e técnicas utilizadas para cifrar o conjunto de dados de origem, com o intuito de protegê-los de terceiros possivelmente mal-intencionados. Uma vez que a quantidade de dados que trafegam pela rede é enorme também é a demanda por softwares que forneçam serviços de criptografia. Dentre os mais importantes estão os sofwares livres, tais como OpenSSL, LibreSSL e GnuTLS, que oferecem maior transparência em relação ao processo criptográfico pois são desenvolvidos por comunidades abertas, tendo milhares de olhos vigilantes analisando o código fonte. Uma vez que o software livre está sempre aberto para estudo e redistribuição, de acordo com sua respectiva licença, ele proporciona maior igualdade no acesso a informação, dando suporte aos desenvolvedores de aplicações, desde o desenvolvedor *full stack* independente até grandes empresas que podem, ou não, contribuir através de doações.

Neste trabalho apresentamos os fundamentos da criptografia de chave pública, em especial RSA, tal como as vantagens da criptografia como software livre. Através do estudo das implementações de *toolkits* criptográficos baseados em código aberto, buscamos definir os pontos fortes e fracos de cada um deles tal como incentivar novas contribuições para com as comunidades.

Palavras-chave: Criptografia, Software Livre, OpenSSL, LibreSSL, GnuTLS.

1 Introdução

Em uma simples tarefa do cotidiano, como acessar um site de notícias, desencadeamos a execução silenciosa de uma série de procedimentos criptográficos cuja finalidade é a proteção da comunicação. Proteção em relação ao conteúdo do acesso, proteção em relação ao usuário que efetua o acesso e proteção em relação às mensagens que serão trocadas. Essa proteção é essencial nos dias de hoje, onde muitos serviços são realizados através da rede, que é tida como não segura e suscetível a ataques e desvios. Essa proteção é construída a partir da criptografia uma vez que para uma pessoa comum nenhum canal jamais deve ser considerado seguro.

A criptografia no senso comum se refere quase que exclusivamente à encriptação, que é o processo de converter informações comuns (chamadas de texto puro) em um texto "embaralhado" (chamado de texto cifrado). Entretanto, é necessário que haja a contraparte oferecida pela decriptação, que transforma o texto cifrado em texto puro. Ao conjunto de algoritmos que realizam as conversões do texto puro para texto cifrado e texto cifrado para texto puro damos o nome de cifra.

A necessidade de cifrar mensagens surge ainda na Antiguidade, sobretudo no trato de mensagens de conteúdo militar, diplomático ou íntimo. Acompanhando o desenvolvimento dos métodos para cifrar mensagens surge a criptoanálise, que estabelece operações de decodificação ou solução de criptogramas, o que muitas vezes alterou o curso da História. Como exemplo, temos a criptoanálise do Telegrama Zimmermann, que motivou a entrada dos Estados Unidos na Primeira Guerra Mundial e tornou bastante claro o risco associado a uma criptografia ruim.

Até a década de 1970, a criptografia foi amplamente utilizada para a proteção de governos, entretanto dois eventos tornaram essa área importante para o usuário comum: a criação de um padrão de criptografia de chave simétrica e o advento da criptografia de chave pública, como RSA, amplamente utilizada nos dias de hoje.

Neste trabalho abordaremos todo o processo criptográfico baseado em RSA, os fundamentos da segurança por ele proporcionada e as relações entre desenvolvimento de *toolkits* criptográficos e software livre, com o intuito de evidenciar a importância deste tipo de software para com a transparência e confiabilidade do serviço fornecido.

2 Conceitos Básicos

De acordo com o Manual de Segurança de Computadores da NIST o termo Segurança de Computadores é definido da seguinte forma:
"Segurança de Computadores: a proteção oferecida para um sistema de informação automatizado a fim de alcançar os objetivos de preservar a integridade, a disponibilidade e a confidencialidade dos recursos do sistema de informação (incluindo hardware, software, firmware, informações/dados e telecomunicações)". Segundo Stallings [1] essa definição evidencia que os três objetivos principais da segurança de computadores são:

- **Confidencialidade:** esse termo cobre dois conceitos relacionados:

  - **Confidencialidade de dados:** assegura que informações privadas e confidenciais não estejam disponíveis nem sejam reveladas para indivíduos não autorizados.
  - **Privacidade:** assegura que os indivíduos controlem ou influenciem quais informações relacionadas a eles podem ser obtidas e armazenadas, da mesma forma que como, por quem e para quem essas informações são passíveis de ser reveladas.

- **Integridade:** esse termo abrange dois conceitos relacionados:
  - **Integridade de dados:** assegura que as informações e os programas sejam modificados somente em uma maneira especificada e autorizada.
  - **Integridade do sistema:** assegura que um sistema execute as suas funcionalidades de forma ilesa, livre de manipulações deliberadas ou inadvertidas do sistema.

- **Disponibilidade:** assegura que os sistemas operem prontamente e seus serviços não fiquem indisponíveis para usuários autorizados.

Esses três conceitos formam o que é normalmente chamado de tríade CIA (do acrônimo em inglês para *confidentiality*, *integrity* and *availability*). Os três conceitos envolvem os objetivos fundamentais da segurança tanto para dados quanto para serviços de informação e computação. Por exemplo, os padrões FIPS 199 (padrões para categorização de segurança para as informações e sistemas de informações federais - nos EUA) da NIST listam a confidencialidade, integridade e disponibilidade como os três objetivos da segurança para informação e sistemas de informação. O FIPS 199 fornece uma caracterização muito útil para esses três objetivos em termos de requisitos e da definição de uma perda de segurança em cada categoria:

- **Confidencialidade:** preservar restrições autorizadas sobre o acesso e divulgação de informação, incluindo meios para proteger a privacidade de indivíduos e informações privadas. Uma perda de confidencialidade seria a divulgação não autorizada de informação.

- **Integridade:** prevenir-se contra a modificação ou destruição imprópria de informação, incluindo a irretratabilidade e autenticidade dela. Uma perda de integridade seria a modificação ou destruição não autorizada de informação.

- **Disponibilidade:** assegurar acesso e uso rápido e confiável da informação. Uma perda de disponibilidade é a perda de acesso ou de uso da informação ou sistema de informação.

Embora o emprego da tríade CIA para definir os objetivos da segurança esteja bem estabelecido, alguns no campo da segurança percebem que conceitos adicionais são necessários para apresentar um quadro completo. Seguem abaixo dois desses conceitos que são mais comumente mencionados.

- **Autenticidade:** a propriedade de ser genuíno e capaz de ser verificado e confiável; confiança na validação de uma transmissão, em uma mensagem ou na origem de uma mensagem. Isso significa verificar que os usuários são quem dizem ser e, além disso, que cada entrada no sistema vem de uma fonte confiável.

- **Responsabilização:** a meta de segurança que gera o requisito para que ações de uma entidade sejam atribuídas exclusivamente a ela. Isso provê irretratabilidade, dissuasão, isolamento de falhas, detecção e prevenção de intrusão, além de recuperação pós-ação e ações legais. Como sistemas totalmente seguros não são ainda uma meta alcançável, temos que ser capazes de associar uma violação de segurança a uma parte responsável. Os sistemas precisam manter registros de suas atividades a fim de permitir posterior análise forense, de modo a rastrear as violações de segurança ou auxiliar em disputas de uma transação.

A segurança de computadores apresenta **muitos desafios**. Segundo Stallings [1] existem diversas razões para isso:

- O conceito de segurança parece mas não é simples. Os requisitos aparentam ser claros e diretos; de fato, a maioria dos mais importantes requisitos para serviços de segurança pode ser autoexplicativa e identificada com rótulos de: **confidencialidade**, **autenticação**, **irretratabilidade** ou **integridade**. No entanto, os mecanismos usados para satisfazê-los talvez sejam bastante complexos e seu entendimento envolva razões bastante sutis.

- No desenvolvimento de um mecanismo ou algoritmo específico de segurança, **deve-se sempre considerar potenciais ataques a essas funcionalidades**. Em muitos casos, os ataques bem-sucedidos são projetados a fim de olhar para o problema de uma forma completamente diferente, portanto, explorando uma fraqueza inesperada no mecanismo.

- Os procedimentos usados para fornecer os serviços de segurança são muitas vezes pouco intuitivos. Normalmente, um mecanismo de segurança é complexo, e não fica óbvio na definição de seus requisitos que essas medidas são necessárias. **Só faz sentido elaborar mecanismos de segurança quando os vários aspectos de ameaças são considerados**.

- Tendo projetado vários mecanismos de segurança, é necessário decidir onde eles devem ser usados. Essa é uma verdade tanto em termos da localização física (por exemplo, em que pontos na rede são exigidos certos mecanismos de segurança) quanto do sentido lógico (por exemplo, em que camada ou camadas de uma arquitetura como TCP/IP devem ser postos tais mecanismos).

- Mecanismos de segurança normalmente envolvem mais do que um algoritmo ou protocolo em particular. Eles também requerem que os participantes possuam algumas informações secretas (como chave de encriptação), **o que levanta outras questões relacionadas à criação, distribuição e proteção delas**. Pode haver igualmente

uma dependência de protocolos de comunicação, cujo comportamento talvez complique a tarefa de desenvolver o mecanismo de segurança. Por exemplo, se o funcionamento adequado do mecanismo de segurança requer determinar limites de tempo de trânsito de uma mensagem do emissor ao destinatário, então qualquer protocolo ou rede que introduza atrasos variáveis ou imprevisíveis pode implicar na perda de sentido de utilizar esses limites de tempo.

- Segurança de computadores e redes é, essencialmente, **uma batalha de inteligência entre um criminoso que tenta encontrar buracos e o projetista ou administrador que tenta fechá-los**. A grande vantagem que o atacante possui é que ele ou ela precisa encontrar uma simples brecha, enquanto que o projetista tem que encontrar e eliminar todas as possíveis brechas para garantir uma segurança perfeita.
- Existe uma tendência natural de uma parte dos usuários e gerentes de sistemas a perceber poucos benefícios com os investimentos em segurança, até que uma falha nela ocorra.
- A segurança requer um monitoramento regular, ou até mesmo constante, e isso é algo difícil com os curtos prazos e nos ambientes sobrecarregados dos dias de hoje.
- **Segurança ainda é muito frequentemente um adendo a ser incorporado no sistema após o projeto estar completo, em vez de ser parte do processo de sua criação**.
- Muitos usuários, e até mesmo administradores de segurança, veem uma segurança forte como um impedimento à eficiência e à operação amigável de um sistema de informação ou do uso da informação.

No escopo de análise deste projeto o objeto de estudo da segurança são as técnicas de criptografia utilizadas para proteger qualquer que seja a informação. Proteção em relação a *sniffers*, empresas (que podem estar rendendo com a venda, em massa, de dados relativos a comportamentos dos usuários) e até mesmo governos.

Softwares livres, em geral, não possuem garantia de superioridade em relação a softwares proprietários no que se refere ao serviço oferecido, mas quando se trata da segurança da informação é vantajoso ter o maior número possível de pessoas olhando para o código em busca de falhas e possíveis *backdoors*, e nesse quesito eles são superiores. O código aberto também possibilita maior transparência em relação ao tratamento dos dados vindouros, uma vez que os processos pelos quais os dados serão submetidos estão "abertos" e podem ser lidos por qualquer um.

3 Criptografia de chave pública

De acordo com Stallings [1] o desenvolvimento da criptografia de chave pública é a maior e talvez a única verdadeira revolução em toda a história da criptografia. Desde o seu início até os tempos modernos, **praticamente todos os sistemas criptográficos têm sido baseados em ferramentas elementares de substituição e permutação**. Depois

de milênios de trabalho com algoritmos que, em essência, poderiam ser calculados à mão, um grande avanço na criptografia simétrica ocorreu com o desenvolvimento da máquina de encriptação/decriptação de rotor. O rotor eletromecânico permitiu a elaboração de sistemas de cifra incrivelmente complexos. Com a disponibilidade dos computadores, sistemas ainda mais complexos foram criados, e o mais importante foi o esforço Lucifer na IBM, que culminou com o *Data Encryption Standard* (DES). Mas tanto as máquinas de rotor quanto o DES, embora representando avanços significativos, ainda contavam com ferramentas básicas de substituição e permutação.

Para Stallings [1] a criptografia de chave pública oferece uma mudança radical de tudo o que foi feito antes. Por um lado, **os algoritmos de chave pública são baseados em funções matemáticas, em vez de substituição e permutação**. Mais importante, a criptografia de chave pública é assimétrica, envolvendo o uso de duas chaves separadas, ao contrário da criptografia simétrica, que utiliza apenas uma chave. O uso de duas chaves tem profundas consequências nas áreas de confidencialidade, distribuição de chave e autenticação.

3.1 Criptografia assimétrica é mais segura que a simétrica?

Segundo Stallings [1] a criptografia assimétrica não é mais segura contra a criptoanálise que a irmã simétrica. A segurança de qualquer esquema de criptografia depende do tamanho da chave e do trabalho computacional envolvido para quebrar uma cifra. Não há nada em princípio sobre a criptografia simétrica ou de chave pública que torne uma superior à outra, do ponto de vista de resistência à criptoanálise. Além disso, o advento da criptografia assimétrica não tornou a criptografia simétrica obsoleta. Por conta do *overhead* computacional dos esquemas de criptografia assimétrica atuais, parece não haver probabilidade previsível de que a criptografia simétrica será abandonada. De acordo com Diffie [2], um dos inventores da criptografia de chave pública: "a restrição da criptografia de chave pública às aplicações de gerenciamento de chave e assinatura é aceita quase universalmente".

3.2 O nascimento da criptografia de chave pública

Segundo Stallings [1] o conceito de criptografia de chave pública nasceu da tentativa de atacar **dois dos problemas mais difíceis** associados à encriptação simétrica:

1. O primeiro deles é a distribuição de chaves. A distribuição de chaves sob encriptação simétrica (1) requer que dois comunicantes já compartilhem uma chave, que de alguma forma foi distribuída a eles ou o (2) uso de um centro de distribuição de chaves. Whitfield Diffie, um dos descobridores da encriptação de chave pública (com Martin Hellman) raciocinou que esse segundo requisito anulava a essência da criptografia: a capacidade de manter sigilo total sobre a sua própria comunicação. Segundo Diffie [2]: "afinal, qual seria a vantagem de desenvolver criptossistemas impenetráveis, se seus usuários fossem forçados a compartilhar suas chaves com um centro de distribuição que poderia ser comprometido por roubo ou suborno?".

2. O segundo problema sobre o qual Diffie ponderou, e que estava aparentemente não

relacionado com o primeiro, foi o de **assinaturas digitais**. Se o uso da criptografia tivesse que se tornar comum, não apenas nas situações militares, mas para fins comerciais e particulares, então as mensagens e documentos eletrônicos precisariam do equivalente das assinaturas usadas nos documentos de papel. Ou seja, deveria ser criado um método para estipular que uma mensagem digital foi enviada por determinada entidade. **Diffie e Hellman construíram um método que resolvia os dois problemas e que era radicalmente diferente de todas as técnicas anteriores de criptografia**.

3.3 Criptossistemas de chave pública

Os algoritmos assimétricos utilizam uma chave para encriptação e uma chave diferente, porém relacionada, para decriptação. Eles apresentam a seguinte característica importante:

- **É computacionalmente inviável** determinar a chave de decriptação dado apenas o conhecimento do algoritmo de criptografia e da chave de encriptação.

Além disso, em alguns algoritmos, como o RSA, também existe a seguinte característica:

- Qualquer uma das duas chaves relacionadas pode ser usada para encriptação, com a outra para decriptação.

Um esquema de encriptação de chave pública possui **cinco elementos**:

1. **Texto claro:** essa é a mensagem ou dados legíveis que são alimentados no algoritmo de entrada.
2. **Algoritmo de encriptação:** realiza várias transformações no texto claro.
3. **Chaves pública e privada:** esse é um par de chaves que foi selecionado de modo que, se uma for usada para encriptação, a outra é usada para decriptação. As transformações exatas realizadas pelo algoritmo dependem da chave pública ou privada que é fornecida como entrada.
4. **Texto cifrado:** essa é a mensagem "embaralhada" produzida como saída. Ela depende do texto claro e da chave. Para determinada mensagem, duas chaves diferentes produzirão dois textos cifrados diferentes.
5. **Algoritmo de decriptação:** aceita o texto cifrado e a chave correspondente e produz o texto claro original.

3.3.1 Etapas do processo criptográfico

1. Cada usuário gera um par de chaves a ser usado para a encriptação e a decriptação das mensagens.
2. Cada usuário coloca uma das duas chaves em um registrador público ou em outro arquivo acessível. **Essa é a chave pública**. A chave acompanhante permanece privada.

3. Se o usuário B deseja enviar uma mensagem confidencial para o usuário A, ele encripta o texto claro usando a chave pública de A.

4. Quando o usuário A recebe a mensagem de B, ele a decripta usando sua chave privada. **Nenhum outro destinatário pode decriptar a mensagem**, pois somente A conhece sua chave privada. O processo como um todo é ilustrado na Figura 1.

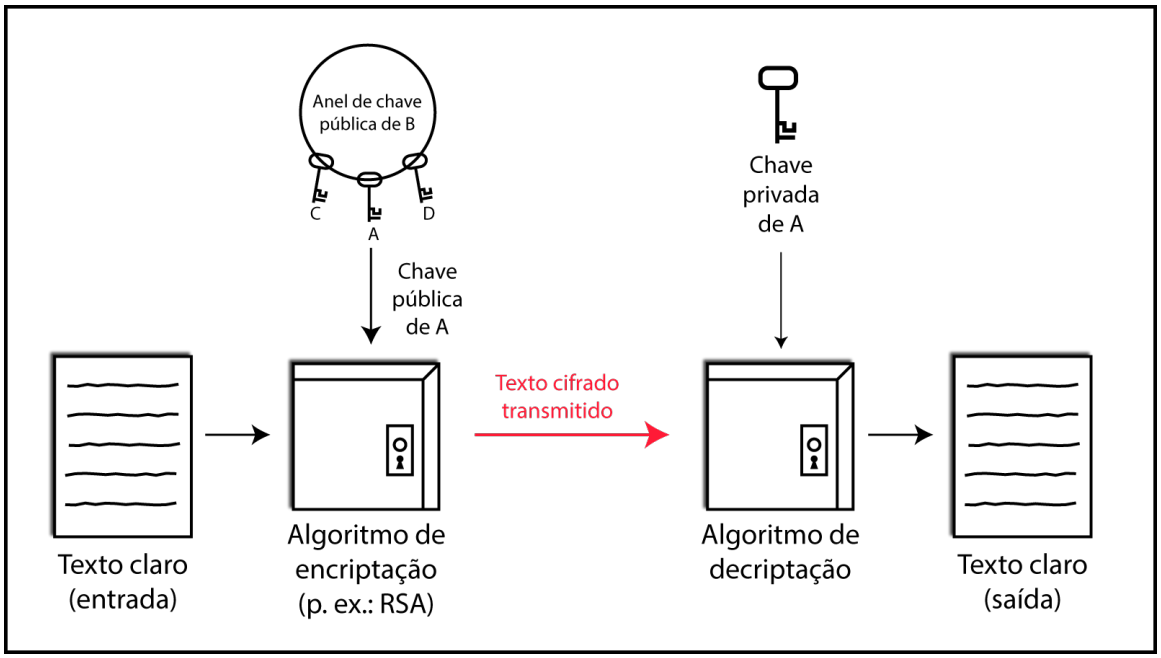

Figura 1: Processo de encriptação/decriptação com RSA. Adaptado de Stallings [1].

Com essa técnica, todos os participantes têm acesso às chaves públicas; as chaves privadas são geradas localmente por cada usuário e, portanto, não precisam ser distribuídas. Desde que a chave privada de um usuário permaneça protegida e secreta, a comunicação que chega está protegida. A qualquer momento, um sistema pode alterar sua chave privada e publicar uma chave pública correspondente para substituir sua antiga.

4 Algoritmo RSA

Diffie e Hellman [3] introduziram uma nova técnica para criptografia que, com efeito, desafiou os criptologistas a encontrar um algoritmo criptográfico que atendesse aos requisitos para os sistemas de chave pública. Uma das primeiras respostas, e a primeira a se tornar pública, foi desenvolvida em 1977 por Ron Rivest, Adi Shamir e Leonard Adleman e publicada em 1978 [4]. O esquema Rivest-Shamin-Adleman (RSA), desde essa época, tem reinado soberano como a técnica de uso geral mais aceita e implementada para a encriptação de chave pública.

O esquema RSA é uma cifra de bloco em que o texto claro e o cifrado são inteiros entre 0 e n-1, para algum n. Um tamanho típico para n é 1024 bits, ou 309 dígitos decimais. Ou seja, n é menor que 2^{1024} .

4.1 Descrição do algoritmo

O RSA utiliza uma expressão com exponenciais. O texto claro é encriptado em blocos, com cada um tendo um valor binário menor que algum número n. Ou seja, o tamanho do bloco precisa ser menor ou igual a

$$log_2(n) + 1$$

na prática, o tamanho do bloco é de i bits, onde

$$2^i < n \leq 2^{i+1}$$

A encriptação e a decriptação têm a seguinte forma, para algum bloco de texto claro M e bloco de texto cifrado C:

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Tanto o emissor quanto o receptor precisam conhecer o valor de n. O emissor conhece o valor de e, e somente o receptor sabe o valor de d. Assim, esse é um algoritmo de encriptação de chave pública com uma chave pública $K_{pb} = \{e, n\}$ e uma chave privada $K_{pr} = \{d, n\}$. Para que esse algoritmo seja satisfatório à encriptação de chave pública, os seguintes requisitos precisam ser atendidos:

1. É possível encontrar valores de e, d e n, tais que $M^{ed} \bmod n = M$ para todo $M < n$.
2. É relativamente fácil calcular $M^e \bmod n$ e $C^d \bmod n$ para todos os valores de $M < n$.
3. Conhecendo e e n é inviável determinar d.

Logo, é necessário encontrar um relacionamento da forma

$$M^{ed} \bmod n = M$$

O relacionamento acima se mantém se e e d forem inversos multiplicativos módulo $\phi(n)$, onde $\phi(n)$ é a função totiente de Euler.

4.2 A função ϕ de Euler

Também chamada de função totiente de Euler, representada por $\phi(x)$, é definida para um número natural x como sendo igual à quantidade de números menores ou iguais a x e co-primos em relação a ele. Matematicamente, tem-se:

$$\phi(x) = \#\{n \in N | n \leq x \wedge mdc(n, x) = 1\}$$

Os valores de $\phi(x)$ são, em geral, difíceis de se calcular, exceto quando x é primo. Quando x é primo, tem-se $\phi(x) = x - 1$. Além disso, a função ϕ é multiplicativa, portanto para p e q primos $\phi(pq) = \phi(p) * \phi(q) = (p - 1)(q - 1)$.

4.3 Encriptação

O relacionamento entre e e d pode ser expresso como:

$$ed \bmod \phi(n) = 1 \tag{1}$$

Que é equivalente a:

$$ed \equiv 1 \bmod \phi(n) \tag{2}$$

$$d \equiv e^{-1} \bmod \phi(n) \tag{3}$$

Ou seja, e e d são inversos multiplicativos módulo $\phi(n)$. De acordo com as regras da aritmética modular, isso é verdadeiro somente se d (e portanto, e) for relativamente primo a $\phi(n)$, o que pode ser expresso como $mdc(\phi(n), d) = 1$. Portanto, o RSA necessita dos seguintes elementos:

1. p, q primos (privados, escolhidos)
2. $n = pq$ (público, calculado)
3. e, com $\text{mdc}(\phi(n), e) = 1$; $1 < e < \phi(n)$ (público, escolhido)
4. $d \equiv e^{-1} \mod \phi(n)$ (privado, calculado)

A chave privada consiste em $\{d, n\}$, e a chave pública em $\{e, n\}$. Suponha que o usuário A tenha publicado sua chave pública e que o usuário B queira enviar a mensagem M para A. Para realizar a comunicação, B calcula $C = M^e \bmod n$ e transmite C. Ao receber esse texto cifrado, o usuário A decripta-o calculando $M = C^d \bmod n$.

5 Protocolo SSL/TLS

O *Transport Layer Security* (TLS), assim como seu antecessor *Secure Sockets Layer* (SSL), é um protocolo de segurança projetado para fornecer segurança nas comunicações sobre uma rede de computadores. Várias versões do protocolo encontram amplo uso em aplicações como navegação na web, email, mensagens instantâneas e voz sobre IP. Os sites podem usar o TLS para proteger todas as comunicações entre seus servidores e navegadores web [6].

O protocolo TLS visa principalmente fornecer privacidade e integridade de dados entre dois ou mais aplicativos de computador que se comunicam. Quando protegido por TLS, conexões entre um cliente (por exemplo, um navegador Web) e um servidor (por exemplo *ic.com.br*) devem apresentar uma ou mais das seguintes propriedades:

- A conexão é privada (ou segura) porque a criptografia simétrica é usada para criptografar os dados transmitidos. As chaves para essa criptografia simétrica são geradas exclusivamente para cada conexão e são baseadas em um segredo compartilhado que foi negociado no inicio da sessão. **O servidor e o cliente negociam os detalhes de qual algoritmo de criptografia e chaves criptográficas usar** antes que o primeiro *byte* de dados seja transmitido. A negociação de um segredo compartilhado é

segura (o segredo negociado não está disponível para bisbilhoteiros e não pode ser obtido, mesmo por um invasor que se coloque no meio da conexão) e confiável (nenhum invasor pode modificar as comunicações durante a negociação sem ser detectado).

- A identidade das partes em comunicação pode ser autenticada usando criptografia de chave pública. Essa autenticação pode ser opcional, mas geralmente é necessária para pelo menos uma das partes (geralmente o servidor).

- A conexão é confiável porque cada mensagem transmitida inclui uma verificação de integridade de mensagem usando um código de autenticação de mensagem para evitar perda não detectada ou alteração dos dados durante a transmissão.

Além das propriedades acima, a configuração cuidadosa do TLS pode fornecer propriedades adicionais relacionadas à privacidade, como sigilo de encaminhamento, garantindo que qualquer divulgação futura de chaves de criptografia não possa ser usada para descriptografar as comunicações TLS registradas no passado [15].

6 Software livre e criptografia

Um aspecto importante do desenvolvimento de software é como ele será licenciado. Uma licença de software estabelece como o código pode ser usado e distribuído pelos usuários finais, o que impacta na escala de adoção da tecnologia implementada. A maior parte dos softwares modernos é vendida sob uma licença proprietária, que permite ao criador manter os direitos de propriedade intelectual do software.

Contudo, existe um ponto de vista alternativo que sustenta que esse tipo de licença coloca um controle desnecessário nas mãos dos criadores de software. Ao impedir que usuários finais copiem e alterem o código fonte do projeto os criadores sufocam a inovação e impedem o potencial crescimento de novas tecnologias. Esse pensamento inspirou a criação de novos tipos de licença que permitem aos usuários finais estudar, alterar e compartilhar o código fonte do software de acordo com sua preferência. Os softwares licenciados dessa maneira são geralmente conhecidos por um de dois nomes: "software livre" (*free software*) ou "software de código aberto" (*open-source software*).

Em geral os dois termos se referem à mesma coisa: software com poucas restrições a respeito de como podem ser usados. Da perspectiva dos proponentes, tanto software livre quanto o de código aberto são mais seguros, mais eficientes e funcionam de maneira mais confiável do que a contraparte proprietária. Neste trabalho, os dois termos serão usados como sinônimos, embora eles apresentem diferenças históricas, de acordo com Drake [13].

6.1 OpenSSL

O OpenSSL disponibiliza um *toolkit* em código livre, que implementa o protocolo SSL e vários algoritmos e primitivas criptográficas de uso comum, incluindo algoritmos de troca de chaves, funções *hash*, algoritmos simétricos e assimétricos. O *toolkit* se apresenta na forma de duas bibliotecas e um conjunto de programas que implementam as rotinas por

elas disponibilizadas. Os mecanismos do SSL estão implementados na *libssl* e os outros algoritmos estão implementados na *libcrypto*.

6.1.1 RSA no OpenSSL

Para a analisar a implementação do RSA no OpenSSL precisamos primeiro localizar os subdiretórios *openssl/crypto/bn* e *openssl/crypto/rsa*, que serão suficientes para uma análise completa de todo o processo de encriptação/decriptação [14]. O processo de encriptação utilizando RSA é bastante simples como explicado anteriormente, mas o problema se torna complexo quando nos perguntamos "Como eu gero um par de chaves RSA?" e "Quão grandes os números precisam ser para haver segurança?". As respostas a essas perguntas aumentam a complexidade do processo criptográfico baseado em RSA.

6.1.2 Geração de chave RSA

Para gerar os números n, e e d é necessário gerar números primos e números coprimos. Para gerar n, é necessário encontrar dois números p_1 e p_2 (primos grandes aleatórios) e multiplicar um pelo outro. As chaves de encriptação e e decriptação d precisam ser coprimas com $(p_1 - 1)(p_2 - 1)$. Isso significa que elas precisam satisfazer a equação:

$$ed \bmod (p_1 - 1)(p_2 - 1) = 1$$

Gerar um número primo aleatório para uma chave RSA é um processo de *guess-and-check*. Os arquivos *openssl/bn/bn_prime.** implementam um crivo para escolher primos. A abordagem mais rápida de crivo faz a pergunta "Esse número é primo?" e recebe apenas as respostas "não" e "talvez". O OpenSSL gera números aleatórios e então executa uma função que testa o número múltiplas vezes para remover quaisquer falso-positivos. Se o teste falha, o número aleatório é descartado e o processo reinicia. **Não há garantia de que o número gerado é realmente primo, mas as chances são altas**.

A chave de encriptação escolhida e geralmente é uma constante, um número primo para o protocolo (tal como o SSL). Assim sendo, apenas o número n precisa ser enviado pela internet. Encontrar a chave de decriptação d é um pouco mais difícil porque é preciso encontrar o módulo inverso. Esse processo envolve calcular o maior divisor comum utilizando o algoritmo de Euclides (no OpenSSL é no arquivo *openssl/crypto/rsa/rsa_gen.c* que encontram-se as ferramentas que tornam isso possível).

A geração de chaves RSA pode ser feita através da linha de comando utilizando *openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:keysize -out file*, onde a *flag* "algorithm" seleciona o algoritmo de encriptação/decriptação, "rsa_keygen_bits" habilita a definição do tamanho da chave dado por "keysize" e "out" define o nome do arquivo de saída.

6.1.3 Números muito grandes!

Uma implementação segura do RSA precisa ser capaz de trabalhar com números grandes. A maioria dos protocolos padrão recomenda chaves com tamanhos de 1024 bits, as quais processadores de 32-bits não conseguem operar com os tipos nativos. Para contornar esse

problema o OpenSSL implementa um novo tipo de dados chamado *BIGNUM*. Essa estrutura e métodos associados estão implementados em *openssl/crypto/bn/bn.**.

Um computador genérico pode manipular números de 1024 bits apenas se eles estiverem armazenados como *arrays* de números menores, com 32 bits. A variável *d* na estrutura *BIGNUM* armazena um ponteiro para um *array* alocado dinamicamente. O comprimento do *array* é armazenado em *dmax* para que as rotinas que manipulam variáveis do tipo *BIGNUM* possam evitar erros de acesso a *buffer*.

Humanos trabalham a adição de números com múltiplos dígitos em pedaços menores adicionando dígitos isolados na mesma coluna da direita para a esquerda. Computadores podem adicionar números grandes da mesma maneira. Se separarmos cada dígito em um elemento separado de *array* é possível implementar adição de acordo com o Algoritmo 1.

```
#define BASE 10

void add(int *a, int *b, int *c, int top) {
  int i, carry;
  for (i=carry=0; i < top; i++) {
    c[i] = a[i] + b[i] + carry;
    if (c[i] >= BASE) {
      carry = 1;
      c[i] -= BASE;
    } else {
      carry = 0;
    }
  }
}
```

Algoritmo 1: Processo de adição entre *BIGNUMS*. Adaptado de Tandon [14].

6.1.4 Licença OpenSSL

O OpenSSL é coberto por uma de duas licenças, dependendo de qual versão está envolvida. Para a versão 3.0.0, e posteriores que desta derivam, aplica-se *Apache License v2*.

Para qualquer versão feita antes do OpenSSl 3.0.0 (ou seja, 1.1.1, 1.1.0, 1.0.2, e todas as versões anteriores atualmente não suportadas), a licença dupla OpenSSL e SSLeay se aplica. É importante observar que isso também é verdadeiro para quaisquer atualizações dessas versões.

6.2 LibreSSL

Após a vulnerabilidade conhecida como *heartbleed* ter sido descoberta no OpenSSL, em 2014, o time do OpenBSD auditou o código fonte e definiu que era necessário um *fork* do código fonte para que pudesse ser feita a remoção de códigos perigosos. Na primeira semana de desenvolvimento mais 90.000 linhas de código em C foram removidas [12].

Nasce então o LibreSSL, como um *fork* do OpenSSL de 2014. Essa bifurcação foi feita com os seguintes objetivos, de acordo com os próprios autores:

- Remover *features* obsoletas ou quebradas.
- Melhorar a base de código do OpenSSL e torná-la mais fácil de auditar, entender e reparar.
- Usar e encorajar a incorporação de interfaces de programação seguras em sistemas operacionais.
- Fornecer alternativas seguras em sistemas operacionais que ainda não tenham interfaces de programação seguras disponíveis.
- Aplicar as melhores práticas nos processos de desenvolvimento: revisão de código, *releases* frequentes, processo de desenvolvimento aberto.

6.2.1 Licença LibreSSL

O LibreSSL deriva do OpenSSL através de *fork*, portanto a licença precisa se adequar aos mesmos termos da licença do OpenSSL, tal como proposto na subseção anterior.

6.2.2 Adoção do LibreSSL

A adoção do LibreSSL foi lenta desde o começo e atualmente a situação está pior. O OpenSSL fornece todo o aparato de baixo nível para diversas funções criptográficas importantes como: implementação dos protocolos TLS e SSL e numerosos utilitários para funções como geração e assinatura de chaves. A maioria dos programas que precisam se comunicar com segurança através da rede acabam aderindo ao OpenSSL mesmo este tendo um posicionamento estranho dentro do mundo linux devido à sua licença especial, que contém um requisito de publicidade considerado incompatível com a *GNU General Public Licence* (GPL). Para contornar este problema, muitos programas licenciados pela GPL incluem uma exceção especial que permite a vinculação ao OpenSSL [23].

O *heartbleed* mostrou como muitos dos sistemas atuais dependem de componentes cruciais que quase ninguém mantém, como era o caso do OpenSSL. Uma das abordagens para consertar esse problema foi direcionar mais recursos para o desenvolvimento e manutenção destes componentes através do estabelecimento da *Core Infrastructure Initiative* (CII) pela *Linux Foundation*. O CII diminuiu um pouco ao longo dos anos, mas teve sucesso em trazer desenvolvedores para projetos como OpenSSL e corrigir muitos de seus piores problemas com infraestrutura sem suporte [23].

O projeto OpenBSD teve sua própria abordagem e o resultado foi a bifurcação LibreSSL. Algumas semanas após o anúncio do *heartbleed* os desenvolvedores do LibreSSL afirmaram ter removido cerca de metade do código bifurcado do OpenSSL. A distribuição do OpenBSD mudou para a nova biblioteca, e o projeto parecia estar começando. Por quase seis anos, LibreSSL permaneceu um projeto viável, produzindo lançamentos regulares conforme avançava. Desde o lançamento 2.9.0 ao final de 2018, o projeto fundiu 1554 *patches* de 36 desenvolvedores, mostrando que está definitivamente vivo e trabalhando. Entretanto, o OpenSSL fundiu mais de 5000 *patches* durante aproximadamente o mesmo período; esse

trabalho veio de 276 desenvolvedores. Tão importante quanto, muito desse trabalho é apoiado por organizações que dependem do OpenSSL (grandes contribuidores incluem Oracle, Siemens, Akamai, Red Hat, IBM, VMware, Intel e Arm - junto com a própria OpenSSL Software Foundation). Este nível de suporte permitiu que o projeto OpenSSL resolvesse muitos de seus problemas de longa data e em 2016 o projeto estava em uma base muito mais estável [23].

Pode-se tentar argumentar que o *fork* do LibreSSL falhou, mas claramente não é o caso, ele está em desenvolvimento ativo e é usado por pelo menos algumas variantes do BSD. Pode-se argumentar, porém, que esse *fork* foi feito muito cedo, este projeto foi criado apenas algumas semanas após a divulgação do *heartbleed*, o que parece muito cedo para concluir que os problemas com o OpenSSL em si não poderiam ser resolvidos sem um *fork*. O OpenSSL acabou não sendo tão impossível de consertar quanto parecia, e o OpenBSD arca com o custo de carregar sua própria bifurcação ao invés de se beneficiar do esforço rejuvenescido do OpenSSL [23].

Talvez algum dia o OpenSSL encalhe novamente e o mundo fique feliz que haja um projeto LibreSSL por aí para recorrer. As alternativas são boas, e a capacidade de criá-las é um dos grandes pontos fortes do software livre, mas o suporte de alternativas também pode ser caro, e parece que a comunidade linux decidiu que essa alternativa em particular não vale o preço [23].

6.3 GnuTLS

O GnuTLS é uma implementação de software livre dos protocolos SSL e TLS. Sua proposta é oferecer uma API (interface de programação de aplicativos) para que aplicações implementem protocolos de comunicação seguros em sua camada de transporte de rede.

O projeto se esforça para fornecer um *backend* de comunicações seguro, simples de usar e integrado com o resto das bibliotecas linux básicas. O *backend* é projetado para funcionar e ser seguro desde o início, mantendo a complexidade de TLS e PKI (Public Key Infrastructure) fora do código do aplicativo.

6.3.1 RSA no GnuTLS

Algoritmos de chave pública como RSA, DSA e ECDSA são acessados usando a API de chave abstrata. Essa é uma API de alto nível que manipula chaves na memória de maneira transparente [20].

Chaves armazenadas na memória podem ser importadas utilizando funções como *gnutls_privkey_import_x509_raw*, enquanto chaves em smart cards ou HSMs (Hardware Security Modules) devem ser importadas com *gnutls_privkey_import_url* [20].

6.3.2 Geração de chave RSA

Todos os tipos de chave disponíveis (incluindo RSA, DSA, ECDSA, Ed25519, Ed448) podem ser geradas com o GnuTLS. As chaves são geradas através de chamada ao método *gnutls_privkey_generate* ou com o mais avançado *gnutls_privkey_generate2*. Essas funções geram chaves privadas aleatórias. A *flag* GNUTLS_PRIVKEY_FLAG_PROVABLE instrui

o processo de geração de chaves a utilizar algoritmos como Shawe-Taylor (FIPS PUB186-4) que gera parâmetros prováveis a partir de uma *seed* para chaves RSA e DSA [21].

6.3.3 Licença GnuTLS

A biblioteca principal é licenciada sob a *GNU Lesser General Public License* versão 2.1 (LGPLv2.1+). A licença LGPL é compatível com uma ampla gama de licenças gratuitas e até permite que você use o GnuTLS em programas **proprietários não-livres** [24].

A principal diferença entre a GPL e a LGPL é que esta permite também a associação com programas que não estejam sob as licenças GPL ou LGPL, incluindo software proprietário [19].

7 A importância de criptografia como software livre

Com o aumento da introdução de dados na rede a informação armazenada de forma eletrônica é mais suscetível de ser vazada para indíviduos não autorizados de maneira intencional ou não intencional. Uma das principais causas para essas brechas de segurança tem sido atribuída a softwares proprietários que mantém o código fonte apenas para a empresa que o produziu. Dessa forma, não temos garantias de que o software proprietário pré-compilado está livre de possíveis *backdoors* que ajudariam um indivíduo a invadir seu computador ou sua rede. A filosofia por trás dos softwares livres e de código aberto, em contrapartida, fornece a qualquer um a oportunidade de analisar o código-fonte para checar quaisquer vulnerabilidades, mudar e compilar de acordo com sua necessidade [5].

De acordo com Schneier [8] um dos pontos de discussão quando se está analisando a segurança de um algoritmo é a exposição dele ao escrutínio. O autor afirma explicitamente que **ninguém deveria confiar em um algoritmo proprietário**. Ele discute que, com poucas exceções, os únicos algoritmos relativamente seguros são aqueles que resistiram ao teste do tempo enquanto eram desconstruídos e analisados por milhares de criptoanalistas. Esse modo de pensar se assemelha muito ao pensamento das comunidades de software livre quando se refere ao tópico da segurança.

Existem muitos debates a respeito de qual tipo de software é melhor - o de código aberto ou o de código proprietário (fechado) - mas **no mundo da criptografia é de conhecimento comum que o software aberto é sempre o melhor**. Segundo Miessler [17] ter qualquer medida da segurança de um algoritmo com base no fato de que é um segredo é geralmente uma coisa ruim. As chaves é que devem tornar o sistema seguro e não o próprio algoritmo (sendo um segredo).

7.1 Segurança usando obscuridade

Essa é a primeira abordagem de segurança que um iniciante no mundo da criptografia pode pensar em implementar, pois se baseia na ideia de que se o código está "escondido" ele está seguro. Na área de *security engineering* a segurança através de obscuridade ou segurança por obscuridade é uma dependência em relação ao sigilo do projeto ou imple-

mentação, como sendo o principal método de proporcionar segurança para um sistema ou componente de um sistema [9].

Um sistema ou seu componente que depende da obscuridade para proporcionar uma camada de segurança pode ter vulnerabilidades teóricas ou reais, mas seus desenvolvedores ou proprietários acreditam que, se as falhas não são conhecidas, o sistema ou componente está "seguro" [9]. Tal paradigma pode funcionar em situações do cotidiano: (1) se você guardar seu carro na garagem, ele estará seguro ou (2) se você guardar seu dinheiro em um cofre, ele estará mais seguro. Entretanto essa não é uma boa abordagem quando se trata de software, pois cedo ou tarde o "segredo" implementado será revelado.

7.2 Filosofia de software livre a respeito da segurança

Uma filosofia que vem sendo amplamente divulgada é a de que os softwares devem possuir código fonte aberto, podendo ser alterados e assim melhorados, tornando-se também mais seguros. De fato, softwares proprietários podem ser lançados sem sequer terem sido revisados ou tendo sido revisados por um pequeno grupo de desenvolvedores; isso, é claro, não impede que um *backdoor* tenha sido introduzido de maneira intencional no código fonte, como aconteceu com o software FrontPage Web Server da Microsoft em 2000, descoberto quatro anos depois [7]. Se o código fonte fosse aberto não haveria esse tipo de comportamento, pois *backdoors* intencionais não seriam permitidos.

Uma abordagem semelhante de abertura há muito tem sido aplicada a algoritmos criptográficos. Na criptografia existe o preceito de que a segurança de um algoritmo não deve depender de sua obscuridade. Logo, algoritmos famosos como RSA, SSL, entre outros, não são segredos, todos conhecem os algoritmos e apenas as chaves são secretas pois elas irão promover a segurança desejada. O teste aplicado aos algoritmos criptográficos pode ser resumido em três etapas [5]:

1. Publica-se o algoritmo e o código fonte.
2. Programadores analisam o algoritmo tentando encontrar erros nele ou no código.
3. Somente depois de algoritmo e código terem sido completamente revisados e nenhuma vulnerabilidade encontrada é que ele é aprovado.

Softwares livres passam pelo mesmo tipo de crivo que os algoritmos criptográficos, ou seja, o código fonte está disponível para todos e todos podem analisar e corrigir. O resultado de tal escrutínio é uma peça de software mais segura.

7.3 Vantagens da criptografia como software livre

Embora existam muitas, alguns pontos devem ser enfatizados [5]:

1. Considere o caso do software proprietário sendo instalado em seu computador para o qual nenhum código-fonte foi fornecido. Em tal situação, não há garantias de que não há "buracos" deixados para possibilitar espionagem intencional da sua privacidade ou não intencional devido a um *design* ruim. Mesmo se o código-fonte estiver "escondido" ele ainda pode ser desmontado e submetido a engenharia reversa por *crackers* e

invasores para descobrir possíveis *exploits*. Em vez disso, considere um código aberto, o qual qualquer um pode revisar. É bastante óbvio que ninguém deixaria um *backdoor* no software para que todos possam descobrir. Consequentemente, há uma garantia implícita de que o código está livre de quaisquer *backdoors* intencionais. Além disso, quaisquer outros *bugs* também são removidos como resultado do processo de revisão coletiva.

2. Quando um código é aberto para ser visto e revisado por todos, o desenvolvedor e programador toma todo o cuidado para fazê-lo bem, limpo e seguro, porque sua reputação está em jogo. Se existem erros no código, toda a comunidade ficará ciente destes erros.

3. Se o código do software é secreto, apenas *crackers* e invasores poderão descobrir os buracos no código e eles raramente publicam este tipo de informação. Assim sendo, erros não divulgados não proporcionam as correções necessárias.

4. Encontrar e consertar vulnerabilidades e erros em softwares populares é uma forma de programadores de código aberto ganharem o respeito de seus colegas e da comunidade. Portanto, fornece uma motivação para examinar código-fonte e improvisar algo novo a partir dele.

5. O código-fonte de um software de código aberto pode ser examinado de duas formas:

   (a) O código inteiro é examinado para qualquer vulnerabilidade. Se o software tem milhões de linhas de código esse pode ser um processo longo e tedioso.

   (b) Se existe uma vulnerabilidade conhecida sendo explorada, o *exploit* pode ser submetido a engenharia reversa para se encontrar a vulnerabilidade. Nesse processo a vulnerabilidade é descoberta indiretamente e depois consertada. Esse processo é mais fácil que o anterior.

6. Uma vez que são abertos a mudanças feitas por qualquer um, softwares de código aberto são muito diversos em sua natureza. Por exemplo, do Unix derivam muitos outros sistemas operacionais, tais como Linux, Solaris, OpenBSD, FreeBSD, entre outros, como mostra a Figura 2. O Linux em si possui diversas distribuições próprias e dada sua variedade ele não é um alvo fácil para *crackers* e programadores de *malwares*. Isso não acontece no software do Windows (proprietário) para o qual surgem novos *malwares* frequentemente.

7. Empresas que tentam manter seus códigos em segredo correm o risco de ter alguma vulnerabilidade descoberta e explorada sem nunca saberem disso. Ou de até ficarem sabendo e sofrerem certo embaraço, como foi em 2005 com a *Harvard Business School* e seu Portal de Admissão que continha uma falha na qual candidatos puderam espiar o *status* de suas admissões [18]. Outro bom exemplo de quão falha é a segurança por obscuridade aconteceu com a rede corporativa da *Cisco Systems Inc.* que fora comprometida e da qual foram roubados 800MB de código-fonte, o que causou grande rebuliço na comunidade de TI visto que os roteadores da Cisco são responsáveis por gerenciar grande parte do tráfego da Internet [18].

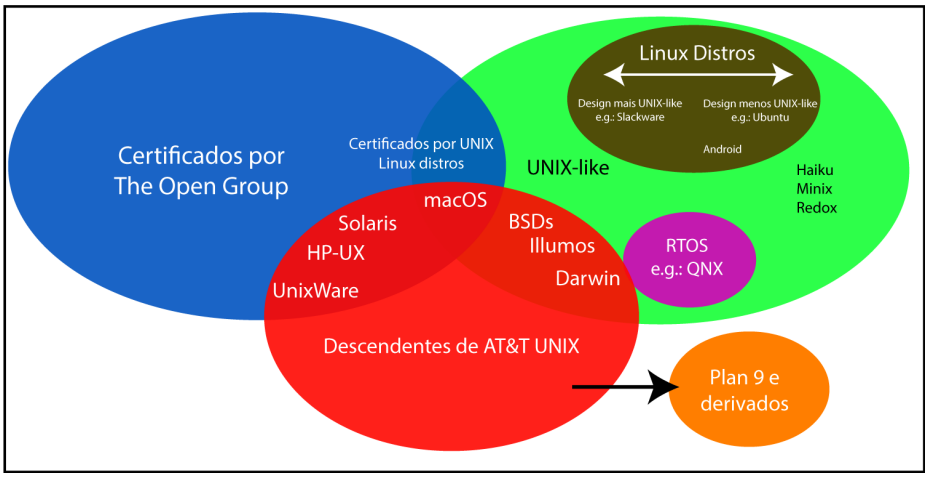

Figura 2: Diversidade em UNIX. Adaptado de Awesome Unix [10].

7.4 Possíveis *trade-offs* do software livre

Embora a filosofia de software livre apresente muitos benefícios se comparada com o paradigma de código secreto, é preciso destacar alguns *trade-offs*:

1. A depender da organização do grupo de trabalho - e claro, da quantidade de membros -, pode parecer que todos do grupo revisaram questões de segurança voltadas ao software quando na verdade ninguém o fez, ou poucos fizeram. Um caso importante envolvendo criptografia aconteceu com o OpenSSL em 2014, no qual a vulnerabilidade que recebeu o nome *heartbleed* possibilitava que invasores tivessem acesso a dados sensíveis como nomes de usuário e senhas. A vulnerabilidade estava presente em milhares de servidores Web, incluindo servidores de grandes empresas, como Yahoo [16].

2. Os integrantes das comunidades de código aberto podem não estar motivados a examinar um código particular para encontrar e consertar erros. Mesmo que existam milhares de pessoas interessadas em encontrar e consertar "buracos" no *kernel* do Linux, pode não haver pessoas motivadas o suficiente para encontrar e consertar falhas em um software menos famoso.

3. Mesmo com o benefício de erros e vulnerabilidades sendo consertados, existe a possibilidade de membros mal intencionados inserirem vulnerabilidades no código. Assim sendo, como precaução, projetos de código aberto apenas aceitam código de fontes confiáveis. Essa confiança é construída da colaboração durante grandes períodos de tempo em projetos diversos. O uso de ferramentas para controle de versão também ajuda a registrar qual programador fez qual alteração. Como exemplo recente, em abril de 2021 pesquisadores da Universidade de Minnesota (UMN) em uma tentativa deliberada de inserir código malicioso no *kernel* do Linux acabaram sendo descobertos no processo de revisão e o resultado foi o banimento da universidade da comunidade

em questão, logo *commits* de endereços *@umn.edu* não serão mais aceitos. O banimento causou polêmica, mas em geral teve mais apoios do que críticas, visto que a intenção dos pesquisadores era publicar um artigo expondo como projetos de código aberto podem ser suscetíveis ao recebimento de contribuições que introduzem vulnerabilidades [11]. Vale lembrar que comunidades distintas possuem processos de revisão distintos tal como medidas punitivas distintas.

8 Conclusão

O desenvolvimento deste projeto possibilitou uma análise dos pilares da criptografia moderna baseada em RSA e, portanto, também de cifras de chave pública. Além disso, permitiu evidenciar o estreito relacionamento entre criptografia e desenvolvimento de software de código aberto, uma vez que a melhor criptografia é aquela que resiste ao teste do tempo, que publica os algoritmos que realizam a encriptação, de forma transparente, e baseia a segurança da aplicação nas chaves e não na obscuridade do código fonte.

Como observado, o desenvolvimento de ferramentas de criptografia e a filosofia de software livre se relacionam em perfeita harmonia, pois código aberto para inspeção e modificação acelera não só o processo de desenvolvimento mas também o processo de testes, sobretudo no que se refere à segurança, que é essencial em ferramentas deste tipo. Qualquer algoritmo criptográfico que baseie sua segurança em obscuridade não pode garanti-la de maneira formal, pois cedo ou tarde o código compilado será descoberto.

Uma vez considerada a importância do desenvolvimento de algoritmos criptográficos para a proteção do volume crescente de dados que trafegam pela rede, torna-se necessária a exploração de formas descentralizadas de desenvolvimento, que agilizem a revisão de código e sejam transparentes com o usuário final. As comunidades de software livre possuem exatamente essa finalidade, e mais: fornecer software de qualidade e com transparência. Mas elas vão além, propiciando o aumento na variedade de software especializado disponível, pois a abertura para modificação gera novas ferramentas através das predecessoras, como aconteceu com Unix, que deu origem aos principais sistemas operacionais que temos hoje.

Por conseguinte, o compêndio do desenvolvimento de software de código aberto aumenta a cada dia e ensina as boas práticas de maneira inclusiva, a todos que estejam dispostos a ler o código. A produção de software livre auxilia na formação ética dos programadores, pois tudo é visível à comunidade, assim ela deve ser encorajada, para que a segurança possa ser pensada como um elemento do projeto desde a sua concepção, assegurando a integridade dos dados tal como a proteção da comunicação.

Referências

[1] Stallings, W. *Criptografia e segurança de redes: princípios e práticas* 6a edição, Pearson, 2014.

[2] Diffie, W. *The First Ten Years of Public-key Cryptography* Proceedings of the IEEE, 1988.

[3] Diffie, W.; Hellman, M. *Multiuser Cryptographic Technique* IEEE Transactions on Information Theory, 1976.

[4] Rivest, R.; Shamir, A.; Adleman, L. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems* Communications of the ACM, 1978.

[5] Farooq-i-Azam, M. *Role of Free and Open Source Software in Computer and Internet Security*, Institute of Information Technology Lahore, Pakistan, 2005.

[6] Dierks, T., Rescorla E. *The Transport Layer Security (TLS) Protocol Version 1.2*, 2008.

[7] Whitlock, W. N. *The security implications of open source software*, IBM, 2001.

[8] Schnier, B. *Schneier on Security*, 2008.

[9] Cesar, J. *Segurança através da obscuridade*, disponível em *www.infosec.com.br/seguranca-atraves-da-obscuridade*, acesso em 25/05/2021.

[10] Barnes, H. *Disambiguation: AT&T UNIX, UNIX Certification, UNIX-Like, and Linux*, disponível em *github.com/sirredbeard/Awesome-UNIX*, acesso em 31/05/21.

[11] Alecrim, E. *Equipe do Linux bane universidade por enviar código malicioso de propósito*, disponível em *tecnoblog.net/435028/universidade-minnesota-banida-envio-codigo-malicioso-kernel-linux/*, acesso em 29/05/21.

[12] Seltzer, L. *OpenBSD forks, prunes, fixes OpenSSL*, disponível em *www.zdnet.com/article/openbsd-forks-prunes-fixes-openssl*, acesso em 24/05/2021.

[13] Drake, M. *The Difference Between Free and Open-Source Software*, disponível em *www.digitalocean.com/community/tutorials/free-vs-open-source-software*, acesso em 24/04/21.

[14] Tandon, J. *Exploring RSA Encryption in OpenSSL*, disponível em *www.linuxjournal.com/article/6826*, acesso em 11/05/2021.

[15] Duncan, R. *SSL: Intercepted today, decrypted tomorrow*, disponível em *news.netcraft.com/archives/2013/06/25/ssl-intercepted-today-decrypted-tomorrow.html*, acesso em 11/05/2021.

[16] Fruhlinger, J. *What is the Heartbleed bug, how does it work and how was it fixed?*, disponível em *www.csoonline.com/article/3223203/what-is-the-heartbleed-bug-how-does-it-work-and-how-was-it-fixed.html*, acesso em 29/05/21.

[17] Miessler, D. *Criptography and Open Source*, disponível em *danielmiessler.com/blog/cryptography_opensource*, acesso em 27/04/21.

[18] Zhen, J. *Insecurity through obscurity*, disponível em *www.computerworld.com/article/2555480/insecurity-through-obscurity.html*, acesso em 26/05/2021.

[19] GNU *Lesser General Public License*, disponível em *www.ufrgs.br/soft-livre-edu/wiki/GNU_Lesser_General_Public_License*, acesso em 08/06/2021.

[20] GnuTLS Community *Public Key Algorithms*, disponível em *www.gnutls.org/manual/html_node/Public-key-algorithms.html*, acesso em 19/06/2021.

[21] GnuTLS Community *Abstract Key API*, disponível em *www.gnutls.org/manual/html_node/Abstract-key-API.html*, acesso em 19/06/2021.

[22] LibreSSL Community *LibreSSL Goals*, disponível em *www.libressl.org/goals.html*, acesso em 26/04/21.

[23] Corbet, J. *LibreSSL languishes on Linux*, disponível em *lwn.net/Articles/841664*, acesso em 22/06/21.

[24] GnuTLS Community *The GnuTLS Transport Layer Security Library*, disponível em *www.gnutls.org/index.html*, acesso em 24/05/2021.

[25] Arch Linux Wiki *OpenSSL*, disponível em *wiki.archlinux.org/title/OpenSSL*, acesso em 22/06/2021.

[26] OpenSSL Community *License*, disponível em *www.openssl.org/source/license.html*, acesso em 25/05/2021.