Um sistema de criptografia de broadcast para navegadores de internet

Hilder Vitor Lima Pereira

Orientador Prof. Dr. Alfredo Goldman

Bacharelado em Ciência da Computação

Instituto de Matemática e Estatística

Universidade de São Paulo

São Paulo, Novembro de 2013

Sumário

Lista de Figuras

Resumo

Este trabalho propõe um sistema de criptografia de campos de páginas web que contenha também uma forma automática de criar e gerenciar as chaves. Trata-se de um sistema de uso geral, que pode servir para praticamente qualquer site.

Tal esquema de criptografia é útil para garantir uma forma simples de criptografar os dados e escondê-los até mesmo dos servidores dos sites que estão sendo acessados.

Além disto, discute detalhes técnicos sobre a implementação de tal sistema, que aqui foi implementado como uma extensão para o navegador Mozilla Firefox, e mostra alguns exemplos de uso.

1

Introdução

1.1 Motivação

O protocolo HTTPS conseguiu fornecer um bom nível de segurança nas comunicações entre terminais clientes e servidores web, protegendo, por exemplo, contra ataques do tipo *Man in the middle* e do tipo *Eavesdropping*[1]. Ou seja, os dados enviados aos servidores estão em segurança. Mas e depois que eles chegam aos servidores web?

Os últimos acontecimentos nos mostraram que garantir uma comunicação segura entre clientes e servidores não é suficiente. Há pouco tempo foi noticiado por grandes jornais, como o *The Guardian*[1] e o *The Washington Post*[2], que o governo estadunidense tinha acesso direto aos servidores de grandes empresas como Google, Yahoo! e Microsoft, por meio do programa PRISM. Isto mostrou que há a necessidade de guardar os dados sensíveis também dos servidores, o que exige uma criptografia no lado cliente antes dos dados serem processados pelo website que esteja sendo acessado.

Junto com as notícias que revelaram este esquema de espionagem internacional praticado pela *National Security Agency* (NSA)[2], agência estadunidense de segurança, houve o surgimento (e o reaparecimento) de muitos sistemas ligados à privacidade[3]. Dentre eles, destacam-se as soluções livres[4], de código aberto, como os sistemas listados no site do *PRISM Break*[5], que vão desde criptografia de e-mails, como o sistema *Mailvelope*, até sistemas de navegação privada, como o *Tor Browser Bundle*. Estas ferramentas, em conjunto, oferecem um bom nível de privacidade, mas, em geral, demandam conhecimento técnico e, em alguns casos, o próprio usuário é que fica responsável pelas chaves usadas para criptografar as mensagens. Além disto, são feitas para casos muito específicos (um para e-mail, um para chat, um para HTTPS, etc), o que acaba acarretando na necessidade de se utilizar vários sistemas em conjunto, o que aumenta a complexidade do ponto de vista do usuário e dificulta o acesso para pessoas mais leigas na área da computação.

Por isto, este trabalho propõe um sistema que criptografa campos das páginas web, funcionando no navegador, de uma forma geral, servindo para uma gama grande de sites. O sistema proposto também faz o próprio gerenciamento de chaves (criação, armazenamento, distribuição), pois isto elimina boa parte das exigências de conhecimentos técnicos dos usuários e facilita a sua utilização.

[1] http://www.theguardian.com/world/2013/aug/23/nsa-prism-costs-tech-companies-paid

[2] washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html?hpid=z1

Ademais, ele é baseado em esquemas de criptografia para sistemas de *broadcast*, o que possibilita seu uso em redes sociais, fóruns e outros sites onde mensagens são enviadas a um grupo de pessoas, ao invés de enviadas a um único destinatário.

1.2 Objetivos

Criar um sistema de criptografia no lado cliente que criptografe os dados antes de deixar os sites os processarem e que implemente algum gerenciador e distribuidor de chaves para manter a navegação dos usuários o mais próximo possível do comum (isto é, tentar fazer a criptografia "transparente" aos usuários, evitando que estes precisem se preocupar com as chaves e com todo o processo técnico envolvido).

2

O sistema

2.1 Funcionamento geral

Toda vez que um usuário quiser enviar um texto criptografado, basta apenas escrever o texto e depois digitaria as teclas de atalho (por enquanto, são CRTL + Espaço) para criptografar o texto.[1]

Por exemplo, digamos que um usuário X está usando uma rede social e ele posta a seguinte mensagem:

"Olá, amigos. O encontro de amanhã será realizado às 16h."

Ele então usaria as teclas CRTL + Espaço e esta mensagem seria criptografada antes de ser enviada aos servidores da rede social. Depois, todas as pessoas que tiverem acesso a esta mensagem (servidores e usuários da rede social) veriam a mensagem criptografada, a menos que estivessem incluídos na lista de pessoas com permissão para descriptografar a mensagem (ou seja, usuários que estiverem na lista de contatos do usuário que mandou a mensagem e que, consequentemente, têm suas chaves relacionadas à chave de sessão utilizada na criptografia).

Dessa forma, a etapa de descriptografia seria "invisível" para o usuário, no sentido de que ela não mudaria em nada a experiência deste com o navegador.

Sendo um pouco mais formal, mas ainda apresentando de forma simplificada, o funcionamento é o seguinte:

- Criptografar:
    1. Um usuário X, proprietário de uma chave de sessão K, atribuída automaticamente pelo gerenciador de chaves, acessa uma página Y.
    2. X preenche um campo qualquer do formulário (pode ser um e-mail, um *chat*, uma postagem em um fórum), usa as teclas de atalho para criptografar e o envia os dados.
    3. Antes dos dados serem enviados, o sistema criptografa os dados dos campos de texto usando a chave K.
- Descriptografar:
    1. Um usuário Z acessa uma página.
    2. O sistema intercepta a página antes do navegador mostrá-la e identifica campos que foram criptografados por algum usuário (digamos, um usuário X).

[1] As teclas para o comando de criptografar ainda podem ser alteradas.

3. O sistema verifica se Z tem permissão para ver o conteúdo criptografado por X (ou seja, se Z está na rede de contatos de X)
4. Caso Z tenha permissão, o sistema descriptografa o conteúdo.
5. Caso contrário, o sistema entrega a página como ela está ao navegador.
6. O navegador mostra a página normalmente (sem se dar conta de tudo o que aconteceu, mostrando o conteúdo criptografado ou não dependendo do que aconteça no passo 3).

2.2 Tecnologias envolvidas no projeto

Esta seção disserta sobre tecnologias usadas na implementação do sistema. Algumas são apresentadas de uma forma superficial, pois já são muito conhecidas.

2.2.1 DOM

O DOM (Document Object Model, em inglês) é uma especificação da W3C [6], de Abril de 2004 (sendo que a primeira versão é de 1998), para uma interface independente de linguagem de programação ou de plataforma, que regula as formas de acessar ou alterar dinamicamente o conteúdo, a estrutura ou o estilo de páginas HTML, XML, XTHML e afins. Ele padroniza a forma de trabalhar com os elementos de um documento web, definindo os objetos, propriedades e métodos de cada elemento de uma página web.

Segundo esta padronização, o documento é representado por uma árvore na qual cada nó é um elemento da página e cada nó é filho de um outro nó se a tag correspondente aquele nó estiver no interior da *tag* correspondete a este. Esta árvore é conhecida como DOM *Tree*. Ela define uma hierarquia dentro da página, sendo que o documento está no topo (a raíz da árvore é um elemento do tipo *Document*) e os filhos do documento são o elementos que representam as tags HEAD e BODY e assim sucessivamente. Linguagens de script para a web, como o JavaScript, usam este padrão para definir as funções e métodos de manipulação das páginas.

2.2.2 JavaScript

JavaScript é uma linguagem de *scripts* da internet, orientada a objetos, interpretada pelos navegadores para executar scripts no lado cliente das aplicações e sites web [7]. Foi inventada por Brendan Eich, na época empregado da Netscape, e vem sendo utilizada na internet desde os navegadores Internet Explorer 3.0 e o Navigator 2.0. Desde 1996, ela é padronizada pela Ecma internacional, sob o nome de ECMAScript.

É uma linguagem imperativa e estruturada, com elementos de sintaxe parecidos com os da linguagem C ou os da linguagem Java, mas apresenta algumas características de linguagens funcionais, como funções aninhadas, fechamento e funções anônimas. Ela é baseada em objetos e é dinamicamente tipada.

Neste trabalho, esta é a linguagem mais utilizada, por conta do padrão imposto pela Mozilla para as extensões do Firefox.

2.2.3 Extensão Mozilla Firefox

Uma extensão, ou *add-on*, é um programa que tem por objetivo adicionar ou modificar funcionalidades de um programa maior.

No caso do Mozilla Firefox, uma extensão é um conjunto de arquivos JavaScript, CSS, XUL (*XML User Interface*), alguns arquivos com meta informações, como o *chrome.manifest* e o *install.rdf*, e arquivos multimídias, como logos e ícones, se necessários. Todos estes arquivos são compactados no formato *zip*, mas com a extensão *xpi*.

Os arquivos *XUL* e *CSS* servem sobretudo para criar a interface da extensão e os arquivos JavaScript são o cerne. Entre estes, existem ainda dois tipos: o *Content Script*, que pode interagir com a DOM e modificar as páginas, e o *Core*, que são os scripts que interagem com o navegador e carregam os content scripts.

A Central do Desenvolvedor de Complementos[2], site oficial do Mozilla Firefox sobre o desenvolvimento de extensões, fornece duas opções para a criação de extensões:

1. Add-on SDK: é um pacote com as APIs e com uma linha de comando, chamada de *cfx command-line tool*, com algumas ferramentas que servem para depurar, testar e empacotar. Precisa ser instalado na máquina que será usada para desenvolver e, diferente da segunda opção, não oferece controle de versão.[8]

2. Add-on Builder: é um ambiente de desenvolvimento *on-line* para criar extensões para o Mozilla Firefox. Além de fornecer uma IDE simples, ainda faz controle de versão e integração com o repositório de extensões mantido pela própria Mozilla. Além disto, ele é bem prático, pois dá a opção de instalar e testar a extensão no próprio navegador: clicando na opção *test*, a extensão é automaticamente compilada e instalada.

Em ambas as opções pode-se utilizar a API (conjunto de classes com métodos úteis para o desenvolvimento) oferecida pela Mozilla. Ela está dividida em APIs de alto nível e APIs de baixo nível, sendo que no primeiro grupo encontram-se classes úteis para fazer a interface visual da extensão ou as partes que irão interagir diretamente com o usuário, como a classe *Tab*, para manipulação de abas, ou a classe *Hotkey*, para a criação de teclas de atalhos. Na API de baixo nível encontram-se classes mais pertinentes ao desenvolvedor e que são usadas para chegar ao objetivo final, mas não como o próprio fim, como por exemplo, o módulo *io/file*, usado para se ter acesso ao sistema de arquivos da máquina em que a extensão está sendo executada ou a classe *MatchPattern*, que permite ao desenvolvedor trabalhar com expressões regulares simples.

Após finalizada, uma extensão pode ser instalada de maneira simples por qualquer usuário: basta ir no menu do Firefox, procurar a extensão pelo seu nome e escolher a opção que a instala.

2.2.4 PHP

O PHP, *PHP: Hypertext Preprocessor*, é uma linguagem de programação dinamicamente tipado, interpretada, de código aberto, orientado a objetos (desde de a versão 5, de julho de 2004) e usada principalmente para para implementar a parte dos sistemas web que interage tanto

[2]Link: https://addons.mozilla.org/pt-Br/developers/builder

com os servidores quanto com os navegadores. Em 2012, havia cerca de 244 milhões de sites usando o PHP[9].

Seu funcionamento geral é descrito na figura 2.1: páginas web com a extensão *PHP* são solicitadas pelos navegadores, o servidor web então interpreta o código php contido nela e este gera uma página dinâmica, imprimindo códigos HTML, CSS e outros tipos de linguagens *client-side*[10].

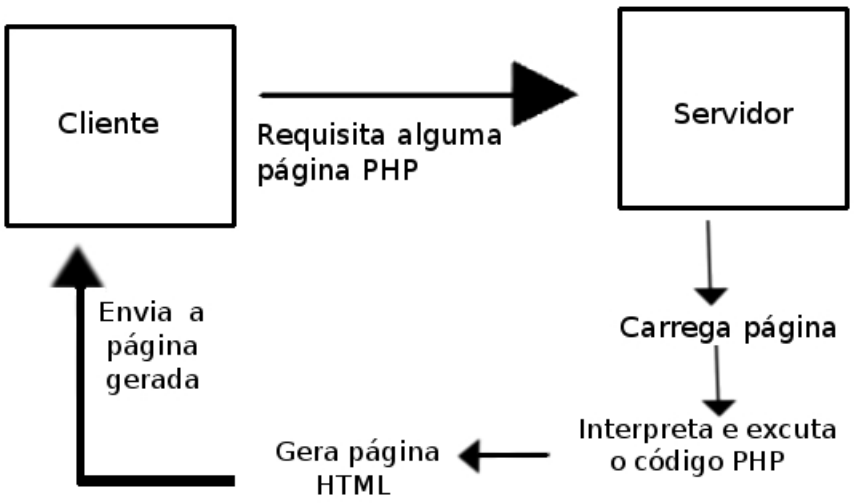

Figura 2.1: Funcionamento geral do PHP

Por exemplo, páginas de autenticação são típicamente feitas em PHP, recebendo informações como o nome do usuário e senha, acessando um banco de dados, verificando se o usuário está cadastrado e se a senha é correta, e devolvendo uma página direcionada ao usuário que está se autenticando.

No caso deste trabalho, o PHP foi utilizado para fazer uma interface de acesso ao banco de dados remoto, pois, por questões de segurança, o acesso direto à banco de dados remotos é bem limitado nas extensões do Mozilla Firefox.

2.2.5 Banco de Dados

Para executar o algoritmo de criptografia, é necessário, além de algumas outras informações auxiliares, que cada usuário tenha um par de chaves pública e privada. E por definição de chave pública, ela precisa estar em um lugar acessível para todos os usuários, por isto, foi necessário criar uma base de dados remota para guardar as chaves públicas.

O banco de dados foi modelado usando o BrModelo, um programa no qual é possível fazer um modelo conceitual do banco de dados e depois gerar os modelos lógico e físico. O modelo conceitual da base de dados é apresentado na figura 2.2. Nele, há apenas duas entidades, a entidade Usuário, representando cada usuário do sistema, e a entidade Chave, representando as chaves públicas dos usuários. Além destas duas entidades, há ainda duas relações, a relação Amizade, da entidade Usuário com ela mesma e representando os contatos entre os usuários e a relação Ter, entre a entidade Usuário e a entidade Chave, dizendo que um usuário tem uma ou mais chaves.

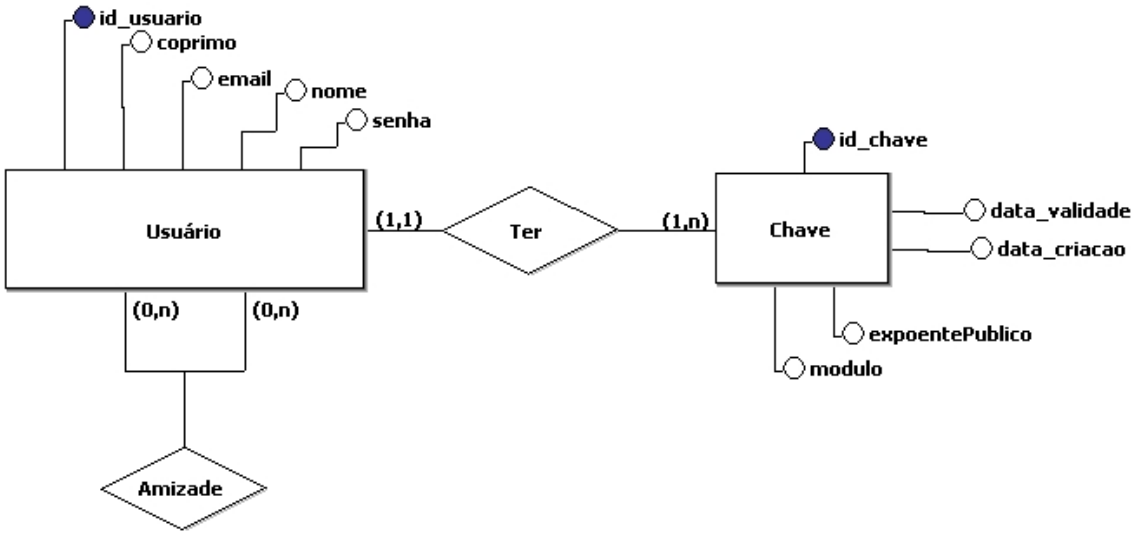

Figura 2.2: Modelo conceitual do banco de dados

Note que apenas a chave pública (o campo *expoentePublico*) fica guardada no banco de dados remoto.

O modelo lógico é apresentado na figura 2.3 e é, por definição, apenas uma tradução do modelo conceitual para uma representação menos abstrata, utilizando-se o conceito de relações (representadas por tabelas).

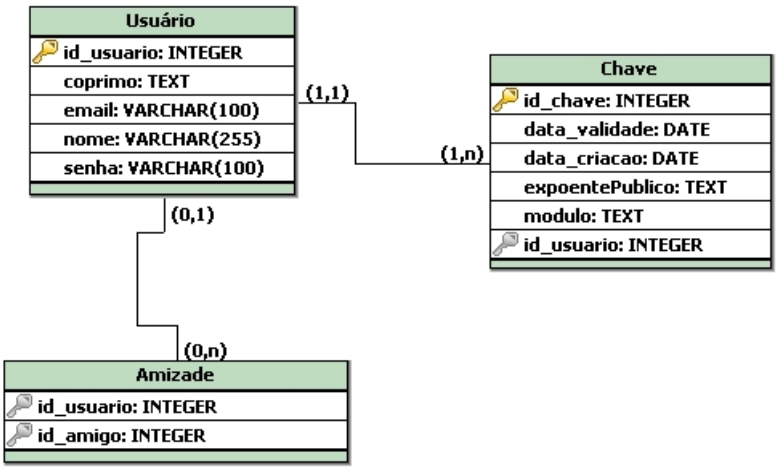

Figura 2.3: Modelo Lógico do banco de dados

O modelo físico consiste apenas do código SQL para criar esta base de dados.

O banco de dados é simples, pois o objetivo é guardar o mínimo de informação necessária sobre os usuários (já que o propósito do sistema é garantir mais privacidade).

Além desta base de dados remota, o sistema também trabalha com uma base local, em SQLite, que, como o próprio site define, é uma biblioteca auto-contida que implementa um gerenciador de banco de dados SQL livre de servidor, transacional e auto-configurável[11]. É usado em grandes projetos como o Skype, o Firefox e softwares da Adobe e do Google.

Esta base de dados serve principalmente para armazenar as chaves privadas no computador do próprio usuário e também para armazenar informações a respeito dos contatos dos usuários, eliminando a necessidade de fazer requisições ao servidor remoto a cada início de sessão.

Para utilizar o banco de dados SQLite a partir da extensão foi utilizada a biblioteca sqlite-jetpack[3], uma biblioteca disponibilizada sob uma licensa de software livre. Além de utilizá-la, acabei fazendo algumas modificações, que foram aceitas pelo mantenedor e agora estou na lista de colaboradores desta biblioteca.

2.3 Detalhes teóricos

2.3.1 Teorema Chinês do Resto

Este teorema é utilizado durante a fase de inicialização de sessão, conforme descrito na próxima seção. Além disto, é um teorema muito utilizado na criptografia, inclusive algumas implementações do *RSA* o utilizam como uma forma de otimizar alguns cálculos[12].

Enunciado

Sejam $I_M = \{1, 2, .., M\}$ e $n_1, n_2, .., n_M$ inteiros positivos tais que $\text{mdc}(n_i, n_j) = 1$ para todo $i \neq j$ em I_M. Então, dado $(a_1, a_2, .., a_M) \in \mathbb{Z}^M$ existe $x \in \mathbb{Z}$ tal que

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \\ \vdots \\ x \equiv a_M \pmod{n_M} \end{cases}$$

Demonstração:

Seja $N = \prod_{i=1}^{M} h_i$ e $N_i = \frac{N}{n_i}$, para cada $i \in I_M$.

Como $\text{mdc}(N_i, n_i) = 1$, existe b_i tal que $N_i \cdot b_i \equiv 1 \pmod{n_i}$ (implicação direta do Teorema de Bézout), logo, $a_k \cdot b_k \cdot N_k \equiv a_k \cdot 1 \equiv a_k \pmod{n_k}$.

Além disto, $a_k \cdot b_k \cdot N_k \equiv 0 \pmod{n_i}$ se $i \neq k$, pois n_i divide N_k. Desta forma, temos que

$$a_1 \cdot b_1 \cdot N_1 + a_1 \cdot b_1 \cdot N_1 + .. + a_M \cdot b_M \cdot N_M + \equiv 0 + 0 + .. + a_i + 0 + .. + 0 \equiv a_i \pmod{n_i}$$

e isto vale para todo $i \in I_M$. Logo, temos que

$$x = \sum_{i \in I_M} a_i \cdot b_i \cdot N_i$$

é solução para o sistema de equações modulares.

[3] https://github.com/julianceballos/sqlite-jetpack

2.3.2 Criptografia de Broadcast

As redes sociais e outros sites em que usuários interagem com grupos de usuários ao invés de interagir com apenas um usuário por vez, como é, tipicamente, o caso do e-mail, estão cada vez mais populares. O Facebook, por exemplo, já é o segundo site mais acessado do mundo[4].

Este tipo de interação "um para N" exige um sistema de criptografia diferente dos clássicos modelos de criptografia nos quais uma mensagem é criptografada para um único destinatário. Esta é a idéia principal da criptografia de *broadcast*: uma mensagem deve ser criptografada para um grupo de usuários e enviada por meio de algum canal que permita que a mensagem seja enviada a todos os contatos de uma só vez. Este canal é chamado de canal de *broadcast*. Sistemas de rádio, televisão e muitos outros usam um canal de *broadcast* e o navegador de internet pode ser entendido como tal, já que a mensagem é enviada uma única vez e redistribuída aos destinatários pelo site que receber a mensagem.

Para entender um pouco a complexidade deste problema, imagine que uma mensagem precise ser criptografada para um grupo de M destinatários. Uma solução (ingênua) para isto seria atribuir um par de chaves pública e privada para cada usuário, criptografar a mensagem M vezes, uma para cada chave pública, concatenar as M mensagens criptografadas e enviar pelo canal de broadcast. Mas note que dessa forma a mensagem foi criptografada várias vezes e concatenar M mensagens criptografadas faz o texto enviado ser muito maior do que o texto original, fenômeno conhecido como expansão do texto criptografado (do inglês *ciphertext expansion*).

Uma outra solução (também ingênua) é a seguinte: o usuário remetente tem uma chave privada que é distribuída para os M contatos. Após isto, a mensagem é criptografada com esta chave e enviada pelo canal de *broadcast*. Este esquema, apesar de exigir que a mensagem fosse criptografada apenas uma vez, apresenta outro problema: a chave privada precisou ser enviada à cada destinatário. Mas o que acontece se um usuário pertence ao conjunto de destinatários de mais de um usuário? Então ele precisará guardar mais do que uma chave privada. De maneira geral, cada usuário precisará armazenar a chave privada de cada usuário que possa lhe enviar uma mensagem. Ou seja, se um usuário está inserido na lista de destinatários de N elementos, então ele precisa armazenar $N + 1$ chaves, contando com a própria chave.

Outro problema com este esquema é que não é fácil atualizar os grupos: dado um usuário V e o conjunto de destinatários $D(V)$, se um usuário $U \in D(V)$ deixa de ser um destinatário de V, então é preciso gerar uma nova chave privada para V (senão U vai continuar conseguindo descriptografar as mensagens) e redistribuir para todos cada usuário $W \in D(V)$.

Para resolver o problema de criptografia de broadcast, este trabalho utiliza um protocolo de criptografia baseado em [13], combinando algoritmos de criptografia simétrica e assimétrica da seguinte forma:

Considere um usuário u cujo conjunto de destinatários é $D(u)$ (conjunto de usuários que podem descriptografar as mensagens enviadas por u). Além disto, para cada usuário v do sistema, considere um inteiro N_v tal que N_v é coprimo de N_z para qualquer $z \neq v$.

Dado isto, o funcionamento pode ser dividido em três partes:

Inicialização: No início da sessão de u, o sistema gera uma chave de sessão K_u. Esta chave

[4] `http://www.alexa.com/siteinfo/facebook.com`

é então criptografada usando algum algoritmo de chave pública e as chaves públicas dos destinatários. Um sistema de equações modulares é criado com estes valores criptografados e o resultado é calculado a partir do Teorema Chinês do Resto. Este resultado é chamado de *locker*.

Ou seja, considerando que cada contato $c_i \in D(u)$ tem como chave pública o valor pub_i, ele faz

$$\forall c_i \in D(u), R_i = E(\text{pub}_i, K_u)$$

onde E é algum algoritmo de criptografia assimétrica. Então acha o *locker* X tal que

$$\begin{cases} X \equiv R_1 \pmod{N_1} \\ X \equiv R_2 \pmod{N_2} \\ \vdots \\ X \equiv R_M \pmod{N_M} \end{cases}$$

onde $M = |D(u)|$.

Note que, pelo Teorema Chinês do Resto, tal X existe.

Criptografia: para criptografar uma mensagem m escrita pelo usuário u, o sistema utiliza um algoritmo de chave privada usando a chave de sessão K_u e concatena com o *locker* X e o *id* de U. Ou seja, faz

$$C = E_s(K_u, m)$$

e envia

$$(\text{id}, X, C)$$

onde K_s é algum algoritmo de criptografia simétrica.

Descriptografia: digamos que um usuário c_i que seja um contato de U, isto é, $c_i \in D(U)$, recebe a mensagem (id, X, C), então, o sistema verifica que c_i está no grupo de contatos de u usando o *id*, consegue a chave de sessão K_u a partir de X e obtém a mensagem original a partir de C, usando K_u.

Para conseguir a chave de sessão a partir de X, é calculado R_i da seguinte forma

$$R_i \equiv X \pmod{N_i}$$

e então R_i é descriptografado com a chave privada de c_i, denotada aqui por priv_i e o algoritmo de chave assimétrica E, ou seja,

$$K_u = E(\text{priv}_i, R_i)$$

e com K_U se obtém a mensagem m a partir de C utilizando o algoritmo de chave simétrica E_s:

$$m = E_s(K_u, C)$$

2.3.3 Algoritmos de criptografia

Chave simétrica : O algoritmo de chave simétrica utilizado neste trabalho foi o *Advanced Encryption Standard* (AES), que é um algoritmo criado pela *National Institute of Standards and Technology* (NIST), para ser o sucessor do algoritmo DES (*Data Encryption Standard*). Ele efetua a criptografia de blocos de 128 bits usando uma chave de 128, 192 ou 256 bits[14]. Pelo fato da chave não ser tão grande e efetuar a criptografia por substituição, ao invés de operações e cálculos numéricas, este algoritmo é mais rápido do que algoritmos de chave pública-privada.

De uma maneira simplificada, o funcionamento do AES é o seguinte: ele começa fazendo uma expansão da chave; faz um *xor* de cada octeto de uma matriz de estados por cada bloco da chave; faz várias substituições não lineares e deslocamentos de bits combinando o bloco a ser criptografado com a chave e as colunas da matriz de estados.

ElGamal : é um algoritmo de criptografia assimétrica baseado no problema do logaritmo discreto[15]. O seu funcionamento é seguinte:

1. Criptografia: Considere uma tripla (g, a, x), com $(g, a) \in G^2$ e $x \in \mathbb{Z}$, onde $(G, \cdot)$ é um grupo e $g^x = a$. Tanto o grupo quantos os valores g e a são públicos. O valor x é chamado de chave privada. Então, uma mensagem m é criptografada para o par (A_0, A_1), onde
$$A_0 = g^k$$
e
$$A_1 = m \cdot a^k$$
sendo k uma chave de sessão (é gerado um valor aleatório k para cada mensagem criptografada).
2. Descriptografia: Dado (A_0, A_1) como acima e a chave privada x: calcula-se $A_2 = A_0^x$, e então
$$A_1 \cdot (A_2)^{-1} = (mA_0^k) \cdot (A_0^x)^{-1} = (mg^{xk}) \cdot (g^{xk})^{-1} = m$$
onde A_2^{-1} é o inverso de A_2 no grupo $(G, \cdot)$.

RSA : é um algoritmo de chave assimétrica cuja segurança está associada à dificuldade prática de achar a fatoração em número primos de um número qualquer. O seu funcionamento é seguinte:[16]

1. Criptografia: Sejam p e q dois números primos, $n = p \cdot q$ e $(e, d) \in \mathbb{Z}^2$ tais que $e \cdot d \equiv 1 \pmod{\phi(n)}$, onde $\phi(n)$ é a função de Euler, definida como a quantidade de naturais menores que n e coprimos de n. Neste caso, $\phi(n) = (p-1)(q-1)$.

   O inteiro e é definido como a chave pública e o inteiro d como a chave privada. Assim, para criptografar uma mensagem m, basta calcular
$$c \equiv m^e \pmod{n}$$

2. Descriptografia: Para descriptografar uma mensagem c criptografada como acima, basta elevar c à chave privada d módulo n. Ou seja:

$$m \equiv c^d \pmod{n}$$

Isto funciona graças ao Teorema de Fermat-Euler, que afirma que dados dois inteiros coprimos a e n, então, $a^{\phi(n)} \equiv 1 \pmod{n}$. Como $e \cdot d \equiv 1 \pmod{\phi(n)}$, existe $\alpha \in \mathbb{Z}$ tal que $e \cdot d = 1 + \alpha\phi(n)$, então,

$$c^d \equiv (m^e)^d \equiv m^{e \cdot d} \equiv m^{(1+\alpha\phi(n))} \equiv m \cdot (m^{\phi(n)})^\alpha \equiv m \cdot 1^\alpha \equiv m \pmod{n}$$

Uma das vantagens do RSA é que ele é mais simples de se implementar e ele tem expansão de texto criptografado 1 para 1. Por outro lado, é necessário usar valores muito grandes para as chaves e para n, senão ele se torna frágil para ataques de força bruta.

O ElGamal trabalha sobre um grupo qualquer (normalmente exige-se apenas que ele seja um grupo cíclico[17]), o que possibilita que se trabalhe com chaves bem menores do que as necessárias no RSA, dependendo das características do grupo escolhido. Por exemplo, sobre grupos de curvas elípticas, pode-se usar chaves de 80 bits, enquanto a maior parte das implementações do RSA utilizam chaves de 1024 bits. Por outro lado, o ElGamal tem uma expansão do texto criptografado de 2 para 1, pois, como descrito acima, ele devolve um par ordenado. Dependendo do grupo escolhido, a mensagem criptografada ficará menor do que a criptografada pelo RSA, mesmo com esta expansão, por conta do tamanho das chaves, mas, como a implementação de uma versão segura do ElGamal sobre um grupo de curvas elípticas demandaria mais tempo do que o disponível neste projeto, acabou-se implementando-o sobre o grupo multiplicativo $(\frac{\mathbb{Z}}{n\mathbb{Z}}^*, \cdot)$, o que acarretou em mensagens criptografadas maiores dos que as geradas pelo RSA. Por este motivo, optou-se por manter o RSA como algoritmo de chave pública utilizado.

2.3.4 Gerenciamento dos grupos de destinatários

Uma das vantagens do esquema de criptografia utilizado neste trabalho é a facilidade de gerenciar os grupos de destinatários: criar um grupo se resume a escolher usuários e vincular suas informações públicas ao usuário dono do grupo que está sendo criado. Adicionar ou remover um destinatário do grupo quer dizer simplesmente que as informações públicas de um usuário passarão ou deixaram de ser consideradas.

Explicando de uma forma mais clara e técnica, se um usuário U deseja criar um conjunto de destinatários $D(U)$, basta apenas guardar a chave pública e o valor coprimo de cada usuário inserido no conjunto $D(U)$. Isto quer dizer que criar um grupo de contatos envolve apenas operações de envio de dados públicos.

Para adicionar um novo contato ao conjunto $D(U)$, basta guardar a chave pública e o valor coprimo deste novo destinatário. Assim, na próxima sessão, estas informações serão consideradas no momento de gerar o *locker* e este usuário estará apto a descriptografar as mensagens.

Por último, para excluir um destinatário do conjunto $D(U)$, basta apenas desconsiderar as informações públicas deste usuário, pois assim, na próxima sessão, elas não serão usadas para calcular o *locker*, o que automaticamente impede este usuário de descriptografar as mensagens.

Portanto, as operações necessárias para manter as características de grupos de contatos não estáticos são simples e não acarretam em *overheads* para o processo de criptografar e descriptografar as mensagens.

2.4 Detalhes Técnicos

2.4.1 Geração dos valores coprimos

Conforme descrito na seção 2.3.2, para cada usuário u o sistema deve gerar e armazenar um número $N_u \in \mathbb{N}^*$ tal que dado qualquer outro usuário v, o $\text{mdc}(N_u, N_v)$ seja igual a um. Ou seja, cada um destes inteiros deve ser coprimo com todos os outros inteiros.

O problema é que gerar um número e verificar se ele é coprimo com todos os outros é muito lento, principalmente considerando que o sistema pode ter muitos usuários.

A solução usada neste trabalho foi manter, no servidor remoto, uma lista de números primos avaliada em [18]. Cada vez que um usuário é cadastrado, um número primo p é selecionado desta lista (de forma aleatória), é calculada a k-ésima potência de p tal que p^k tenha pelo menos 300 algarismos (o que ocorre quando k é aproximadamente igual a 45), o valor p^k é atribuído ao novo usuário e o primo p é eliminado da lista, fazendo com o próximo usuário cadastrado receba uma potência de outro primo.

Isto garante que os valores atribuídos serão sempre coprimos entre si, pois são sempre potências de primos diferentes, e ainda economiza processamento, já que calcular essa potência pode ser feito de forma eficiente (por exemplo, com o algoritmo de exponenciação rápida, que tem complexidade assintótica $O(\log(k))$).

2.4.2 Resolução do sistema de equações modulares

Como discutido na seção 2.3.2, no início de cada sessão é necessário resolver um sistema de equações modulares valendo-se do Teorema Chinês do Resto.

O problema que aparece aqui é a questão do desempenho do Javascript. Pelo fato de ser uma linguagem interpretada, seu desemprenho já não é tão bom. Além disto, como o código Javascript não é carregado diretamente na página para ser executado pelo navegador, pois a estrutura da extensão coloca mais camadas entre o interpretador e o código, o desempenho fica ainda pior.

Foram efetuados testes em uma máquina com um processador Intel Celeron de 1.2GHz e memória cache de 1024 KB, memória RAM de 3GB. Nestes testes, o tempo que a extensão levava para resolver um sistema de n variáveis (referente à n destinatários) foi medido 10 vezes para cada valor de n testado. Os inteiros envolvidos no teste eram inteiros de 1024 bits, portanto, não têm suporte nativo do Javascript. O tempo médio de execução é apresentado na figura 2.4, que relaciona o tempo (em segundos) com a quantidade de equações (eixo horizontal).

Como era esperado, a extensão demorou muito para resolver os sistemas. Além das razões supracitadas, um outro fator deste péssimo desempenho é o fato de não haver uma biblioteca oficial (distribuida e mantida pelos mesmos mantenedores do Javascript) de números de precisão arbitrária. O desenvolvimento de uma biblioteca destas é algo complexo e está fora do escopo do

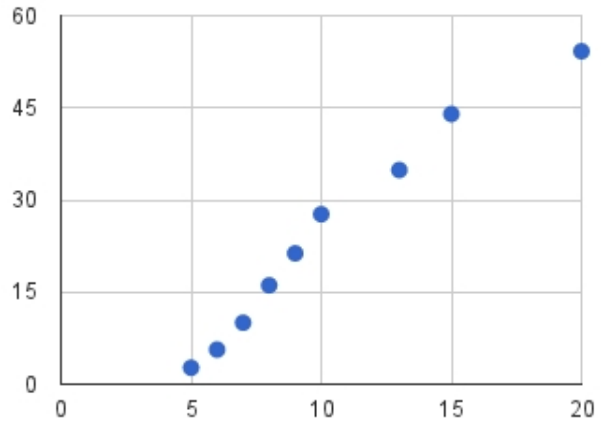

Figura 2.4: Gráfico do tempo médio de execução pelo número de contatos (Javascript)

projeto, por isto, foi utilizada a biblioteca *Leemon BigInt JS*[5], que é disponibilizada na forma de domínio público.

Observado isto, a seguinte alternativa foi formulada: como os dados utilizados neste sistema são públicos (isto é, nenhuma chave privada está presente no sistema, mas apenas a chave de sessão criptografada com as chaves públicas de cada usuário), não há problema em enviá-los a um servidor remoto, que pode calcular a solução do sistema de uma forma mais rápida e devolvê-la à extensão. Neste caso, o gargalo seria o tempo de envio e recebimento de dados pela internet, que, todavia, ainda é bem menor do que os tempos apresentados na figura 2.4.

Assim, foi implementado o mesmo algoritmo na linguagem C usando a biblioteca *GNU Multiple Precision Arithmetic Library*[19]. Além de todas as vantagens de desempenho, visto que é escrita em C e *Assembly*, ela tem diversas vantagens do ponto de vista de engenharia de software, pois é bem encapsulada, esconde os detalhes de implementação (por diversas vezes foi necessário olhar o código da biblioteca em Javascript utilizada aqui para poder executar de forma correta algumas das funções disponíveis) e tem um padrão de nomenclatura de funções e constantes bem definido.

O mesmo procedimento do teste do Javascript foi utilizado para testar a função implementada em C, isto é, foi utilizada a mesma máquina, inteiros do mesmo tamanho, dez testes para cada sistema e o tempo médio de execução de cada um destes dez testes registrado. Os resultados estão apresentados na figura 2.5.

Pode-se observar que o tempo de execução é extremamente mais baixo: um sistema com 100 equações levou, em média, 63 centésimos de segundo enquanto um sistema com 20 equações levou, em média, 54 segundos para ser resolvido usando Javascript.

Então, mesmo com o tempo de envio e recebimento das informações pela internet, resolver o sistema no servidor, usando uma implementação em linguagem C, leva menos tempo do que calcular em Javascript na máquina do cliente. Além disto, o navegador do cliente fica bloqueado enquanto ele efetua os calculos, o que atrapalha a navegação e dá a impressão que o navegador "travou", por isto, enviar os dados ao servidor e deixar o navegador livre enquanto os dados

[5]http://www.leemon.com

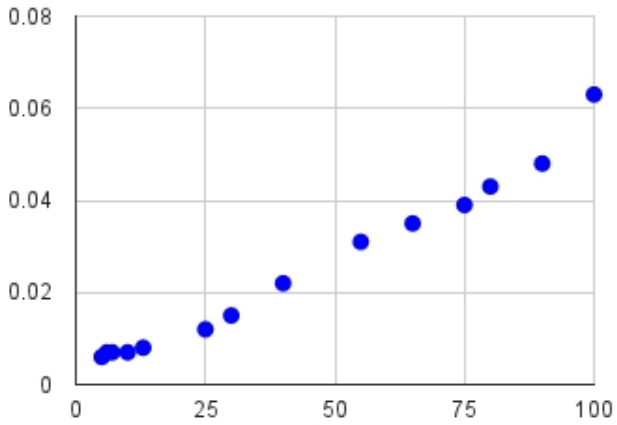

Figura 2.5: Gráfico do tempo médio de execução pelo número de contatos (C)

não são recebidos de volta vai de encontro com o objetivo de interferir o mínimo possível na experiência do usuário com a internet.

3

Resultados

Como dito anteriormente, foi produzida uma extensão para o navegador Mozilla Firefox que criptografa e descriptografa campos de texto de páginas HTML e faz o gerenciamento das chaves.

Quando o navegador é aberto, uma pequena janela no campo inferior esquerdo é mostrada para que o usuário efetue a autenticação no sistema, conforme mostra a figura 3.1. Os dados do usuário são lidos e então verifica-se, no banco de dados local, se ele está cadastrado. Caso não esteja, lhe é oferecida a opção de efetuar o cadastro e depois pede-se que digite os e-mails dos destinatários. Caso já esteja cadastrado, uma chave de sessão é gerada, ela é criptografada com as chaves públicas dos contatos e o sistema de equações modulares é resolvido.

Quando o navegador carrega uma página, a extensão procura por algum texto que tenha sido criptografado e o descriptografa ou não, conforme descrito na seção 2.1. Do ponto de vista do usuário, a página está sendo carregada normalmente.

Sempre que um usuário deseja criptografar um texto, ele precisa pressionar as teclas de atalho. Este é um dos pontos em que a experiência do usuário com o navegador muda, pois, introduz uma etapa a mais na edição das mensagens, já que normalmente seria necessário apenas escrever o texto e depois enviá-lo clicando em algum botão ou simplemente apertando a tecla "ENTER", mas, infelizmente, é muito difícil de ser evitado, pois se o usuário não precisasse pressionar as teclas de atalho, o próprio sistema teria que decidir quais campos criptografar e quais não criptografar, ou seja, seria preciso fazer algum tipo de análise semântica das páginas. Por exemplo, em uma página de e-mail, não se pode criptografar o campo "destinatário", mas não há problema algum em criptografar o corpo da mensagem...

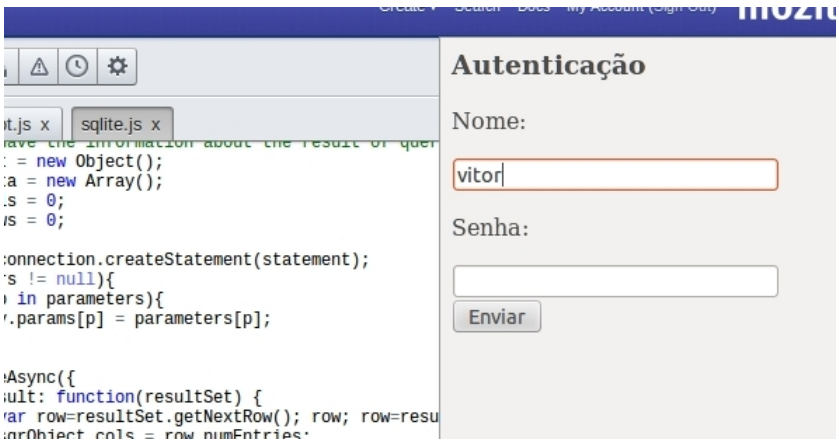

Figura 3.1: Janela mostrada para autenticar usuários

Todavia, dados os objetivos propostos, os resultados foram satisfatórios. Do ponto de vista do usuário, boa parte do processo acontece de forma "transparente": com exceção cadastro e da necessidade de pressionar as teclas de atalho (dois pontos muito difíceis de serem eliminados), os outros pontos que poderiam alterar a experiência do usuário são o tempo necessário para inicialização do navegador, devido ao *overhead* de gerar as chaves de sessão e gerar o *locker* e o tempo necessário para criptografar e descriptografar as mensagens. Porém, como a criptografia das mensagens é feita usando o AES, ela ocorre muito rápido: o tempo necessário para o usuário parar de pressionar as teclas de atalho e começar a interagir com o navegador novamente é maior do que o tempo que a mensagem leva para ser criptografada. O processo para descriptografar é um pouco mais crítico, pois é necessário varrer a página, porém, isto também não apresenta um *overhead* que possa ser notado, já que, em geral, o tempo que o navegador leva para reendenizar a página, executar os scripts anexados a ela e mostrá-la já é grande.

Uma forma de aprimorar o sistema e deixá-lo mais usual é gerenciar a DOM *Tree* das páginas de tal forma que toda vez que um novo nó for inserido nela, a extensão verifique se aquele nó tem carrega um texto cripografado, pois, até a entrega desta monografia, o a extensão percorre a página durante o seu carregamento e descriptografa o que for necessário, mas não leva em consideração os nós que são adicionados depois, o que é algo típico em páginas de HTML dinâmico.

Ademais, vale notar que apesar de uma extensão para o navegador Mozilla Firefox ter sido implementada neste trabalho, ele não está vinculado a nenhum navegador específico e pode ser implementado também em outros navegadores.

4

Conclusão

Este trabalho apresentou um método geral de criptografia de *broadcast* no lado cliente, criptografando os campos das páginas HTML antes que estes sejam processados pelos sites e protegendo os dados tanto de terceiros que possam acessá-los enquanto viajam pela rede quanto dos próprios servidores.

Apesar da ideia de criptografar os campos parecer simples, é uma tarefa complicada executá-la: o Javascript não é uma boa linguagem para fazer cálculos com números grandes, é necessário se preocupar com diversos problemas além de garantir a segurança, como desempenho, tamanho da mensagem depois de criptografada, *overheads* durante o carregamento das páginas... Isto tudo dificulta o trabalho e cria uma grande distância entre a idealização teórica do sistema e a sua implementação.

Contudo, verificou-se que é possível construir tal sistema e fazer com que ele seja funcional. Tanto a instalação quanto a utilização são simples, o que aumenta a chance do sistema ser aceito pelos usuários.

No futuro, pretende-se disponibilizar este sistema sob alguma licensa de software livre, colocando-o como projeto oficial do Centro de Competência de Software Livre da USP (CCSL).

5

Parte Subjetiva

5.1 A Universidade de São Paulo

Sempre estudei em escola pública, concluindo meu ensino médio na escola Professora Maria de Lourdes Almeida Sinisgalli, uma escola que nem sequer aparece na lista oficial de notas do ENEM, pois dentre os seus alunos, a cada ano, apenas poucas dezenas fazem o exame.

Portanto a USP (e na verdade qualquer universidade, tanto pública quanto privada) sempre foi algo fora da minha realidade. Tanto eu quanto muitos vizinhos e colegas de escola nem ao menos sabíamos que a USP existia.

Mas, terminado o ensino médio, comecei a trabalhar e acabei conhecendo a ETESP (Escola Técnica Estadual de São Paulo), onde consegui começar a fazer um curso técnico em informática. Foi lá que soube o que era a USP, pois praticamente todos os meus colegas de classe estavam terminando o ensino médio e estudando para o vestibular. Assim, comecei a fazer um cursinho pré-vestibular durante os finais de semana e continuei trabalhando e fazendo o curso técnico durante a semana.

Depois de um ano estudando, trabalhando e praticamente não dormindo, conseguir passar na FUVEST. Tinha muitas expectativas, algumas se mostraram verdadeiras, mas outras não (algo natural, pois a idealizei demais).

Após começar a estudar aqui, uma das coisas que mais me desmotivava era a falta de humanidade nas relações e avaliações dos alunos. Mesmo depois de entrar na universidade eu precisava trabalhar e ganhar dinheiro para me manter e isto implica em uma série de atividades extra-acadêmicas, mas, para a meritocracia, isto não importa. Se alguém mora muito longe, chega cansado na universidade por conta do transporte público, precisa trabalhar, participa de projetos sociais, faz atividades fora da universidade, mas tem média 7, então ele com certeza é pior do que um uma pessoa que consegue pagar uma república do lado da USP, que vai de carro à universidade, que tem todas as condições socioeconômicas ao seu favor e tem média 8. Acho realmente muito raso e triste o método de julgamento e classificação dos alunos baseado apenas nas notas que cada aluno tem. Isto acaba mudando o foco dos estudos, que deveria ser *aprender*, para "passar", "tirar nota", etc.

Talvez por conta disto eu já tenha visto tantas pessoas colando durante provas, decorando provas dos anos anteriores na esperança de que os professores iriam colocar as mesmas questões (o que muitas vezes acontecia), porém, são poucos os que eu conheço que estudam com prazer, que querem realmente aprender ao invés de passar e se formar.

Apesar dessas críticas, a USP me proporcionou muitas experiências que não teria vivido em outras universidades: a iniciação científica, que graças ao apoio do meu orientador e à algumas noites mal dormidas, rendeu um artigo publicado em uma revista internacional; o estágio no tanque numérico da POLI, onde tive contato com supercomputadores de ponta e apliquei os conceitos de programação paralela em problemas reais; o estágio no laboratório de matemática aplicada, lugar onde consegui aprender muito sobre Linux, redes de computadores e onde comecei a dispertar meu interesse por criptografia e segurança de dados; o intercâmbio, que me proporcionou uma experiência na qual obtive um grande crescimento pessoal e profissional, melhorando muito meu nível de francês, fazendo quatro matérias em outra universidade, em outra lingua, e participando de um projeto relacionado com criptografia.

Acredito que esta universidade mudou completamente a minha vida e gostaria muito que ela investisse em mais projetos de extensão para se fazer mais presente na sociedade e, quem sabe, conseguir inserir muitos jovens de difícil condição socioeconômica em uma realidade totalmente diferente, assim como aconteceu comigo.

5.2 O Instituto de Matemática e Estatística

Na minha opinião, o ponto forte do curso de computação da USP é o seu formalismo, rigor e toda a base matemática passada aos alunos. E acho que o fato dele ser ministrado dentro do IME influencia muito nisto, pois o IME é um ótimo instituto e tem uma tradição matemática muito forte.

Por outro lado, durante toda a minha graduação, só vi dois professores conversarem na sala de aula, com os alunos, sobre iniciação científica. Ao meu ver, os alunos deveriam ser muito mais incentivados a participar dessa atividade, pois, além de toda a bagagem científica, ela traz muita maturidade.

5.3 O BCC

O curso de computação na USP, apesar da grade um pouco antiga, é muito bom: ele oferece uma boa base teórica e mescla com matérias práticas como as de laboratório de programação.

É muito difícil dizer quais as matérias que eu julgo mais importante no curso, mas vou tentar listar algumas:

- Estrutura de dados: uma matéria muito importante, onde se aprende de verdade onde usar cada estrutura, os pontos fracos e fortes de cada uma delas, começasse a aprender sobre complexidade assintótica, reforçasse os conceitos de recursão...

- Álgebras 1 e 2: por darem uma boa base matemática, ensinando de forma sólida o que são provas matemáticas; por introduzirem teorias tão belas como a teoria de grupos, corpos e anéis e tão úteis na matéria de criptografia.

- Engenharia de Software: apesar de achar que a matéria não é bem lecionada, visto que é uma matéria muito complexa e os professores tentam exigir que grupos de 3 ou 4 alunos contruam um sistema usando todos os conceitos aprendidos no curso (levantamento de

requisitos, documentação, *design*, implementação, testes, validação, etc...) em apenas seis meses, é uma matéria de extrema importância no curso, pois com ela é possível aprender como grandes projetos funcionam, aprender boas práticas de projetos e ter mais segurança para implementar sistemas maiores que os EPs.

- Álgebra linear: uma matéria extremamente útil, já que é usada com muita frequência no restante do curso e em praticamente qualquer problema que queiramos modelar e resolver. Além disto, é uma matéria fundamental para que os alunos tenham mais contato com a linguagem matemática.

Mas acho que o curso do BCC peca em não ter uma matéria obrigatória de redes de computadores e não ter matérias voltadas a sistemas web.

Para finalizar, gostaria de dizer que apesar de todas as dificuldades encontradas ao longo do caminho, tenho muito orgulho e prazer de ter feito este curso. Aprendi a superar muitos desafios, me virar sozinho, pesquisar melhor e isto tudo é de grande valia. Sinto que tenho plena capacidade de seguir adiante tanto na área acadêmica quanto no mercado.

Referências Bibliográficas

[1] E. Rescorla, "Http over tls," *RFC 2818*, 2000.

[2] T. Guardian, "Nsa paid millions to cover prism compliance costs for tech companies.." `http://www.theguardian.com/world/2013/aug/23/nsa-prism-costs-tech-companies-paid`, 2013.

[3] TIME, "The anonymous internet: Privacy tools grow in popularity following nsa revelations.." `http://business.time.com/2013/06/20/the-anonymous-internet-privacy-tools-grow-in-popularity-following-nsa-revelations/`, 2013.

[4] L. Monde, "Prism – comment passer entre les mailles de la surveillance d'internet ?." `http://bigbrowser.blog.lemonde.fr/2013/06/11/prism-comment-passer-entre-les-mailles-de-la-surveillance-dinternet/`, 2013.

[5] P. Break, "Prism break's home page." `https://prism-break.org/`, 2013.

[6] W. W. W. Consortium, "W3c website." `www.w3.org`, 2013.

[7] D. F. P. Ferguson, *JavaScript: The Definitive Guide (5th ed.)*. O'Reilly & Associates, 2006.

[8] M. Firefox, "Add-on sdk developer guide." (`https://addons.mozilla.org/en-US/developers/docs/sdk/latest/`), 2013.

[9] PHP, "Php official website." `www.php.net`, 2013.

[10] P. Dall'Oglio, *PHP - Programando com Orientação a Objetos*. Novatec Editora, 2007.

[11] SQLite, "Page about sqlite." `http://www.sqlite.org/`, 2013.

[12] J. Großschädl, "The chinese remainder theorem and its application in a high-speed rsa crypto chip," *Annual computer Security Applications Conference*, vol. 16, pp. 4–5, 2000.

[13] G.-H. C. W.-T. Chen, "Secure broadcasting using the secure lock," *IEEE Transactions of Software Engineering*, vol. 5, no. 8, pp. 929–934, 1989.

[14] U. S. N. I. of Standards and T. (NIST)., "Announcing the advanced encryption standard (aes)," novembro 2001.

[15] H. V. L. Pereira, "Algorithmes de cryptographie et le problème du logarithme discret," *Wikimedia Commons*, janeiro 2013.

[16] R. A. S. L. A. Rivest, "A method for obtaining digital signatures and public-key cryptosystems," 1978.

[17] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," 1985.

[18] BigPrimes, "A large list of prime numbers." `http://bigprimes.net/`, 2013.

[19] GNU, "The gnu multiple precision arithmetic library manual." `http://gmplib.org/manual/`, 2013.