Por Guilherme Gall, Pedro Lara, Fabio Borges

Webmail com PHP + JAVA

Neste trabalho será apresentado um novo método de autenticação para *webmail* que usa como recursos as linguagens PHP e JAVA através de tecnologias J2ME.

Neste artigo iremos propor um esquema de autenticação para *webmail* (poderá ser futuramente estendido para outros sistemas de autenticação) que usará a linguagem PHP do lado do servidor e JAVA através de tecnologias J2ME no cliente. Na verdade, o nosso grande foco consiste em que o servidor gerará um senha provisória através de alguma técnica de GNA (Geração de Números Aleatórios) e após ""encriptada"" por um servidor enviará para um usuário que dispõe de um celular ou PDA (*Personal Disital Assistants*) para "descriptar" esta senha, e assim este usuário poderá logar-se usando esta senha provisória. A vantagem do uso desta técnica consiste em possibilitar que um usuário autentique-se em uma máquina infectada, por exemplo, com um *keylogger,* uma vez que a senha gerada pelo servidor não poderá ser usado novamente para este usuário autenticar-se no *webmail*.

1. Motivações

A grande motivação deste artigo se concentra no grande números de artefatos maliciosos (*malware*) que tramitam na *web*. Entre estes um dos que mais se destaca é o *spyware*. Um *spyware* é um *software* que auxilia a captura de informações sobre uma pessoa, organização ou computador sem o seu consentimento. Um *spyware* pode enviar tais dados para outra identidade sem o seu conhecimento ou tomar controle do computador sem que o usuário saiba disso. De todos os tipos de *malwares* existentes o *spyware* é o mais comum. Segundo uma pesquisa conduzida pela Dell em setembro de 2004, estima-se que aproximadamente 90% dos PCs com *Windows* possuíam no mínimo um *spyware*. Este tipo de *software* foi responsável por metade das falhas reportadas por usuários da Microsoft em ambiente *Windows*. Outro estudo apontou uma média de 25 *spywares* por PC. No nosso trabalho atentaremos a presença de um tipo específico de *spyware:* o *keylogger.*

Um *keylogger* é um tipo de *spyware* cuja finalidade é capturar tudo que um determinado usuário digita em um teclado. Existem diversos tipos de uso de *keyloggers*, como por exemplo aqueles que monitoraram um funcionário de uma empresa. Entretanto os *keyloggers* também são usados para fins ilícitos: capturar senhas, números de cartões de crédito e qualquer outro tipo de informação puramente sigilosa. Estas informações podem ser enviadas diretamente para o *e-mail* do indivíduo que age de má fé. Tudo isso sem o consentimento da vítima que está acessando o computador "infectado" [Pereira E. et al.]. Podemos dividir os *keyloggers* em três classes diferentes: *software keylogger*, *hardware keylogger* e *kernel keylogger* [Pereira E. et al.].

Software keylogger: Talvez seja a classe de *keylogger* mais difundida na W*eb*. Estes *softwares* usam técnica de *hooking*. Os programadores deste tipo de *keylogger* utilizam funções disponibilizadas pela API (*Aplication Program Interface*) do sistema operacional para captar mensagem e teclas pressionadas antes que estas funções sejam tratadas. Esses *keylogger*s podem ser instalados remotamente, no entanto, são os mais lentos e facilmente detectáveis por programas como anti -vírus e *anti-spywares.*

Hardware keylogger: Trata-se de um equipamento físico que se localiza entre o teclado e o gabinete do computador da vítima. Apesar de possuir pouco recurso computacional, ele processa bem rápido as informações capturadas. Este dispositivo não é detectado por anti-vírus e nem *anti-spyware*, entretanto, pode ser visualmente detectado.

Kernel keylogger: Este *keylogger* atua no nível do *kernel* do sistema operacional. A confecção deste tipo de *keylogger* exige técnicas apuradas de programação e de sistemas operacionais. O sistema operacional encontra muita dificuldade também em detectá-lo. Este tipo de *keylogger* não é capaz de captar informações em nível de aplicação: operações de auto-completar, copiar e colar. Devido ao fato de trabalharem no núcleo do sistema.

2. Descrição do Método

Aqui discutiremos a técnica que iremos propor neste trabalho. O nosso grande objetivo é criar uma senha provisória para dar acesso a um determinado usuário em

um sistema uma única vez. Após autenticado com esta senha provisória, este usuário não poderá inserir esta senha a fim de autenticar-se neste sistema. Este fato nos garante que se caso tenha instalado algum tipo *keylogger* na máquina de acesso, a senha original não estará comprometida e caso um atacante intercepte a senha provisória, esta não será mais válida após a sua autenticação. Logo o usuário estaria livre de qualquer tipo de *software* comprometer a segurança aspecto da autenticação. Para gerar esta senha provisória o servidor pode usar alguma técnica de Geração de Números Aleatórios (GNA) que gere números bem distribuídos de aproximadamente 80 *bits*.

A grande dificuldade se concentra em enviar esta senha de maneira segura para o usuário que deseja autenticar-se em um sistema. Para isso usaremos a técnica de criptografia de chave pública: um servidor sobre o qual o usuário deseja-se autenticar gera uma senha aleatória através de GNA; "encripta" esta senha usando a chave pública do destinatário (que será conhecida e poderá estar armazenada em uma base de dados) e a sua chave privada (do servidor), para decifrar esta senha o usuário usa a chave pública do servidor e a sua chave privada. Para "descriptar" a senha o usuário usará um telefone celular ou um PDA (*Personal Digital Assistants*), por exemplo.

A escolha de curvas elípticas está intrinsecamente ligada com o fato de usarmos equipamentos com pouco poder computacional para auxiliar o envio seguro da senha provisória para o usuário. Como visto na tabela 1 podemos ter criptografia segura usando aritmética de 160 *bits*. Se usássemos o RSA com segurança equivalente teríamos que usar inteiros de 1024 *bits*. Tal diferença é agravada quando se pretende implementar nestes tipos de dispositivos. Na verdade, usaremos estes equipamentos (celulares, PDAs) para tão somente efetuar os cálculos pertinentes a decifragem da senha.

O envio da senha "encriptada" para o usuário pode ser feito através de mensagem SMS (*Short Message Service*) visto que o PHP oferece este tipo de serviço. Caso o usuário deseje, ele pode optar por inserir manualmente a senha "encriptada" em seu celular, no entanto, isto não é o mais aconselhável, uma vez que esta senha pode ser formada de alguns longos dígitos. Este usuário também tem a opção de fazer o *download* desta senha "encriptada" através de um cabo transferidor de dados, por exemplo.

Os algoritmos de GNA e os algoritmos de criptografia podem ser construídos usando rotinas em PHP, no entanto, estas rotinas usam números grandes (160 *bits* se usarmos algum método baseado em curvas elípticas). Para utilizar números desta magnitude pode ser adicionada ao PHP a biblioteca (GNU *Multiple Precision Arithmetic)* (GMP) [GMP PHP]. Além de ser uma das mais rápidas bibliotecas para lidar com inteiros grandes, esta biblioteca dispõe de funções demasiadamente usadas em criptografia, como por exemplo exponenciação modular.

3. Escolha e Justificativa do Algoritmo

A escolha de um algoritmo baseado em curvas elípticas está intrinsecamente ligada com o fato de que estaremos usando equipamentos com pouco poder computacional, neste caso celulares. Algoritmos baseados em curvas elípticas proporcionam uma chave criptográfica consideravelmente menor que alguns clássicos da criptografia, tais como o RSA. Se estivéssemos usando o RSA precisaríamos de uma chave criptográfica de aproximadamente 1024 *bits*, no entanto se trabalharmos com curvas elípticas usaríamos uma chave de 160 *bits,* mantendo a mesma segurança. Este fato decorre que o Problema do Logaritmo Discreto (problema sobre o qual o algoritmo está baseado) sobre curvas elípticas é mais difícil de ser resolvido do que o Problema da Fatoração de Inteiros que é o problema que o RSA se baseia.

Existem vários algoritmos que trabalham sobre curvas elípticas, inclusive algoritmos que já existiam antes da descoberta desta técnica em 1985. Na verdade, os algoritmos que se baseavam no Problema do Logaritmo Discreto em grupos aditivos da forma Zp podem ser estendidos para atuar sobre curvas elípticas. Um algoritmo que se destaca é o Menezes-Vanstone que, diferentemente dos algoritmos adaptados para trabalhar sobre curvas elípticas, não precisa codificar a mensagem em pontos da curva elíptica, facilitando assim a "encriptação" da mensagem. No nosso caso o servidor gerará uma senha aleatória. Se usarmos o criptossistema Menezes-Vanstone não teremos dificuldades em "encriptar" esta senha. No entanto, usando o protocolo Diffie-Hellman, por exemplo, antes de "encriptar" a mensagem deveríamos codificá-la em pontos de uma curva elíptica, gerando assim uma dificuldade a mais para a implementação.

4. Tecnologias Envolvidas

Nesta seção resumiremos as principais ferramentas e tecnologias envolvidas na execução do método proposto.

4.1 Biblioteca GMP

Como parte do processo de implementação do método proposto, é necessária a implementação do algoritmo criptográfico escolhido. É fato que existem poucas implementações para algoritmos baseados em curvas elípticas para PHP e que a documentação existente é precária – em muitas vezes não existe.

Com estes fatores sentimos a necessidade de implementar todas as operações pertinentes ao algoritmo escolhido (neste caso, o Algoritmo Menezes-Vanstone). Como precisamos trabalhar com inteiros relativamente grandes, uma ótima opção seria usar a biblioteca GMP [GMP PHP]. E esta oferece funções que são de suma importância para a implementações de criptografia. Como por exemplo a função `gmp_nextprime(int $a)` que retorna o próximo primo maior que `$a`. O trecho de código abaixo mostra o funcionamento da função

`gmp_invert(resouce $a, recouce $b)`. Esta função calcula o inverso de `$a` módulo `$b` e também terá grande utilidade na implementação do algoritmo.

```
<?php
$c1="740775700530478596626314617315308773084909391351";   $primo="11692013098647223345629478661730264157247524989731";
$invMod=gmp_invert($c1,$primo);
echo gmp_strval($invMod);
?>
```

4.2. Java e biblioteca Bouncy Castle

A tecnologia Java é uma plataforma criada pela empresa Sun Microsystems na qual o desenvolvedor escreve programas através de uma linguagem de programação que gere código executável pela máquina virtual Java. O grande diferencial desse tipo de desenvolvimento é que, como os programas são escritos para uma máquina virtual, qualquer dispositivo que tenha uma máquina virtual Java instalada pode rodar os programas, independente de hardware ou sistema operacional.

A segmentação da tecnologia Java de interesse deste trabalho é a J2ME (Java 2 Micro Edition), que consiste basicamente em uma coleção de APIs (Application Programming Interface) voltadas ao desenvolvimento de aplicações para dispositivos com processamento limitado, como celulares e PDAs, e uma máquina virtual Java para rodar nesses dispositivos.

Basicamente, a arquitetura do J2ME divide-se em três camadas: máquina virtual, configurações e perfis. A máquina virtual é a camada de mais baixo nível e roda diretamente sobre o sistema operacional do dispositivo. Ela é quem executa os programas gerados para a plataforma Java e, portanto, é o que define as limitações dos programas que serão executados nos dispositivos. A camada acima da máquina virtual é a configuração e consiste basicamente em uma especificação que define o ambiente de software para uma faixa de dispositivos definida por um conjunto de características tais como tipo e quantidade de memória disponível, processador e sua freqüência de operação e o tipo de conexão de rede disponível para o dispositivo.

Em termos práticos, uma configuração é formada por um conjunto de bibliotecas que estarão disponíveis para o desenvolvedor criar programas em uma faixa de dispositivos com aplicações distintas aplicações, mas com diversas características em comum. Como exemplo de faixa horizontal de dispositivos, pode-se citar os dispositivos móveis com conexão a rede sem fio. Nessa faixa estariam incluídos aparelhos como celulares, PDAs e pagers. A justificativa para a criação de configurações é que apesar dos diversos aparelhos serem distintos em forma e funcionalidades, muitos deles tem características de processadores e memória muito similares e, por isso, são colocados sobre uma mesma especificação que determina o nível de funcionalidades e serviços que tem que ser oferecidos pela máquina virtual.

A camada de mais alto nível na tecnologia J2ME é o perfil. Ele define um conjunto de bibliotecas específicas para o desenvolvimento de programas para uma faixa vertical de dispositivos e foca mais na aplicação e segmento de mercado do aparelho do que em suas características de processador e memória. Um exemplo de faixa vertical de dispositivos são os telefones celulares. Os perfis complementam uma configuração J2ME formando um serviço completo para que as aplicações possam ser executadas. Para clarificar a diferença entre perfis e configurações, é interessante lembrar que perfis são mais específicos que configurações. Assim, se pode ter uma configuração para dispositivos com baixo poder de processamento e um perfil para os celulares. A especificação de um perfil sempre é feita para uma determinada configuração, mas uma configuração pode suportar vários perfis diferentes. Como se objetivou que o sistema de autenticação proposto por esse projeto fosse utilizável em diversos contextos e por diversos usuários, usou-se o perfil MIDP sobre a configuração CLDC para a implementação do software que rodará no dispositivo móvel por essa ser a combinação de configuração/perfil presente em quase 100% dos celulares.

Para implementar o algoritmo de criptografia com curvas elípticas é necessária a capacidade de fazer contas com números de grandeza maior do que a grandeza dos números representáveis com os tipos primitivos do Java. Como entre outras de suas limitações, a biblioteca de classes do perfil MIDP não traz a classe BigInteger que permite a manipulação de números com a grandeza necessária. Fez-se imprescindível a utilização de uma biblioteca de classes que trouxesse consigo uma maneira de manipular números grandes o suficiente para a implementação do sistema. A biblioteca escolhida foi a Bouncy Castle, que é uma biblioteca de classes usada para criptografia que traz uma implementação compatível com J2ME da classe BigInteger. As diversas outras classes da biblioteca Bouncy Castle que poderiam ter sido usadas na implementação deste sistema de autenticação não foram usadas por motivos esclarecidos no item Detalhes de Implementação.

5. Detalhes de implementação

Tanto na implementação do software do dispositivo móvel quanto na do servidor foram criadas duas classes para abstrair os conceitos fundamentais envolvidos na criptografia com curvas elípticas: Curva e Ponto.

A classe Curva recebe através de parâmetros do seu construtor os coeficientes *a* e *b* além de um número primo *p* usado para definição do corpo **Zp** (a curva é definida por $y^2=x^3+ax^2+b$). A principal dentre entre as suas funções é a função multiplicacao( ) que retorna um objeto do tipo Ponto resultante da multiplicação por escalar de Ponto P por um inteiro *k*. Esta função ocupa algo em torno de 92% do processamento na criptografia.

A classe Ponto recebe como parâmetros em seu construtor as coordenadas *x* e *y* de ponto e um valor *booleano* que indica se o Ponto é no infinito ou não.

Mais detalhes sobre criptografia baseada em curvas

elípticas poderão ser vistas em [Lara, P C S Oliveira, F B]. Suas outras funções são *getters* e *setters* para as coordenadas do Ponto e o valor *booleano* citado anteriormente.

No caso da implementação do software que rodará no dispositivo móvel, não foram utilizadas as classes de criptografia com curvas elípticas da biblioteca Bouncy Castle porque a documentação sobre elas é insuficiente. Assim, a única classe dessa biblioteca usada foi a versão para J2ME da classe java.math.BigInteger. Como não existe essa classe na configuração CLDC e no perfil MIDP é necessário usar um *obfuscator*, não apenas para os nomes das classes, mas também para o nome dos pacotes para usar essa classe, pois não é possível criar classes dentro do pacote Java por questões de segurança.

As chaves públicas dos usuários são pares de pontos armazenados em um banco de dados no servidor e as chaves privadas ficam apenas nos dispositivos dos usuários. A chave pública do servidor, composta por um par de pontos também é de conhecimento de cada dispositivo móvel.

6. Considerações Finais

No presente momento o método proposto neste trabalho está em fase de implementação e testes. Como trabalhos futuros, pretendemos abrir todos os códigos sob uma licença livre e estender este conceito para outros sistemas que necessitem de autenticação de usuários, além de melhorar o desempenho do programa Java que rodará nos dispositivos móveis.

Referências e links sugeridos

[GMP PHP] – http://www.php.net/gmp

Lara, Pedro Carlos da Silva; Oliveira, Fábio Borges. ***Curvas Elípticas: Aplicação em Criptografia Assimétrica.*** *In: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais 2007. Rio de Janeiro.* (WTICG), 2007.

Muchow, Jonh W. ***Core J2ME: tecnologia & MIDP***. São Paulo: Pearson Makron Books, ISBN: 85-346-1522-5 2004.

Pereira, Evandro; Fagundes, Leonardo Lemes; Neukamp, Paulo; Ludwig, Glauco; Konrath, Marlom. ***Forense Computacional: fundamentos, tecnologias e desafios atuais***. VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.

Pedro Lara - pcslara@lncc.br
Graduando em Tecnologia da Informação pelo Instituto Superior de Tecnologia e em Matemática pela Universidade Federal Fluminense através do CEDERJ, atualmente atua na área de segurança da informação e criptografia assimétrica. Sua área de interesse é Criptografia Baseada em Curvas Elípticas.

Guilherme Gall - gmgall@lncc.br
Graduando em Tecnologia da Informação e da Comunicação pelo Instituto Superior de Tecnologia em Ciências da Computação de Petrópolis. Atualmente trabalha no Serviço de Redes do Laboratório Nacional de Computação Científica.

Fábio Borges - borges@lncc.br
Possui Bacharelado em Matemática pela Universidade Estadual de Londrina (UEL) e Mestrado em Modelagem Computacional pelo Laboratório Nacional de Computação Científica (LNCC). Atualmente é Tecnologista do LNCC onde é responsável pelo Setor de Treinamento e Apoio (STA). Tem experiência na área de Segurança da Informação. Atuando principalmente nos seguintes temas: Criptografia, Esteganografia, Web.