Universidade Federal do Ceará – UFC

Aluno: Natã Santana de Morais

Número: 383808

Curso: Ciência da Computação

Disciplina: Criptografia

Implementação do algoritmo criptográfico RSA e do protocolo DSA

Crateús – CE

03/12/2017

Introdução

Na criptografia, a encriptação é o processo de transformar uma mensagem original em uma mensagem ilegível, para terceiros. Também existe o conceito de decriptação, que nada mais é o inverso da encriptação. Nessa mesma área de estudo, existem os algoritmos criptográficos simétricos e assimétricos.

Entre esses algoritmos de criptografia de dados, temos o RSA. O mesmo revolucionou esse ramo da criptografia, sendo um dos mais seguros, e o primeiro algoritmo a possibilitar assinatura digital.

O RSA tem como fundamento a utilização de um par de chaves: uma chave pública, que pode ser conhecida por todos, e uma chave privada, que deve ser mantida em sigilo. Ou seja, toda mensagem que foi encriptada usando uma chave pública, só pode ser decriptada usando a respectiva chave privada. Este algoritmo baseia-se na complexidade computacional da fatoração e sua força está quantificada no número de bits.

Existe também, um protocolo de assinatura digital, denominado DSA. Este é um algoritmo assimétrico, porém o mesmo somente assina.

Objetivos

O objetivo da implementação do RSA e do protocolo DSA é conhecer, compreender e saber construir um algoritmo desse porte. Aprender sobre o funcionamento do RSA e DSA é de extrema importância quando o assunto é criptografia assimétrica. Além disso, é importante buscar atingir uma compreensão sobre a manipulação dos conceitos teóricos de encriptação e decriptação utilizando o RSA e o DSA com a parte prática.

Além disso, é de extrema importância alcançar a compreensão total desses métodos criptográficos, pois atua fortemente na internet, como troca de mensagens de e-mails e compras on-line, por exemplo.

Desenvolvimento

A implementação dos algoritmos foi dividida em duas fases. A primeira se destinou na construção do algoritmo RSA e a fase posterior a implementação do DSA. Ambas as implementações foram desenvolvidas em linguagem JAVA.

Na primeira fase, no caso a implementação do RSA, criamos um pacote que abrigaria a classe Key, na qual guarda as informações da chave pública (e), chave privada (d) e o 'mod n'.

Logo após, foi criada uma classe RSA, na qual implementaríamos o algoritmo. Construímos um método de geração de chaves (generateKey) que recebe como parâmetro a quantidade de bits de segurança que é desejada. Dentro dessa função é gerada os primos 'p' e 'q' com a ajuda da classe BigInteger. Ambos são a metade do tamanho dos bits de segurança que queremos alcançar. Pois, quando multiplicados, se aproximam ou são iguais a quantidade de bits de segurança que desejamos. Após isso, multiplicamos p e q e guardamos em uma variável.

Seguindo o algoritmo RSA, calculamos o 'phi' na qual é dado por (p -1) * (q - 1) e depois escolhemos um primo aleatório 'e' e fixamos. Por último, calculamos o inverso de 'e mod phi' e geramos a chave 'd' privada.

Depois disso, foi criado o método de encriptação (encrypt) que recebe como parâmetro BigInteger 'e' e 'n' e o vetor de caracteres com a mensagem. O processo consiste em usar o método estático 'squareMultiply' na qual a mensagem é elevada a 'e' MOD 'n'. O método de decriptação (decrypt) é da mesma forma, porém a mensagem encriptada é elevada a 'd' MOD 'n'.

Ainda não falado, o método estático 'squareMultiply' consiste em receber BigInteger 'h', 'm' e 'x'. Com métodos auxiliares no JAVA, transformamos o 'h' em uma String em seu formato binário. Logo após, implementamos uma interação para cada posição do vetor de caracteres e aplicamos os conceitos do 'squareMultiply'.

Finalizada a primeira fase, a segunda e derradeira fase se inicia.

Primeiramente, para implementarmos o DSA, criamos uma classe 'KeyPublic' na qual estão contidas as informações que poderão ser públicas e uma outra classe 'Key' que estende 'KeyPublic' que acrescenta a classe informações privadas.

Após isso, criamos uma classe 'GenerateKey' para geração de chaves. A mesma possui um método principal (generateKey) que retorna as informações necessárias. Dentro deste método realizaremos quatro chamadas de outros métodos. O primeiro retorna primos p e q. Para isso, foi utilizado um algoritmo fornecido pelo livro, na qual ele se baseia em gerar um 'q' primeiro e descobrir posteriormente o 'p'. Porém, houve um problema. Se colocarmos o p como sendo de 1024bits o algoritmo demora um pouco para encontrar um padrão que case os dois. Portanto, para fins de teste, foi usado o p como sendo de 200

bits. Tanto faz, se colocarmos 1024 ou 200 bits, o algoritmo funciona. Porém, o tempo de geração de chaves entre eles é bem diferente.

Prosseguindo a segunda fase, foi implementado um método para alpha. Através de pesquisas extras, foi constatado que o alpha ‘a’ pode ser calculado escolhendo-se um aleatório no intervalo de ‘p’, elevando-o a ‘p-1’/’q’, e por fim realizando MOD ‘p’. Realizamos o mesmo procedimento no algoritmo utilizando o ‘squareMultiply’ para as operações.

Após isso, foi implementado um método para a obtenção da chave privada ‘d’ aleatória, utilizando o SecureRandom e um valor fixado de 128 bits. Para finalizar a geração de chaves foi implementado um método para o cálculo de Beta, que consiste basicamente em elevar o alpha ‘a’ em ‘d’ e finalizando com MOD ‘p’.

Logo após, foi criado uma nova classe ‘DSA’ com os métodos de verificação (verificationSignature) e geração de autenticação (generateSignature). O método de verificação foi implementado utilizando as funções que o BigInteger proporciona. O método recebe como parâmetro uma ‘KeyPublic’ e a mensagem. O mesmo calcula e realiza operações de acordo com a regra de verificação do DSA. A mensagem, para ser computada, foi transformada em seu HASH usando o hashCode do Java. Ao final de todas as operações é comparado se o resultado ‘v’ é igual ao ‘r’.

Para finalizar, implementamos o método de geração de autenticação. O mesmo recebe como parâmetro, uma ‘Key’ e a mensagem. Neste método, primeiramente é calculado um ‘Ke’ privado, de forma aleatória e no intervalo de ‘q’. Opcionalmente, o ‘Ke’ será um provável primo. Depois disso, são gerados o valore de ‘r’, que é calculado elevando alpha ‘a’ em ‘kE’ e depois realizando o MOD ‘p’ utilizando o ‘squareMultiply’, e o ‘s’, multiplicando o inverso de ‘Ke’ com o resultado da soma do HASH da mensagem com a multiplicação de ‘r’ e ‘d’ e logo depois realizando um MOD ‘q’.

Dificuldades encontradas

A implementação do RSA e do protocolo DSA tiveram suas facilidades, mas também suas dificuldades.

A primeira dificuldade encontrada foi na manipulação de inteiros muito grandes. O armazenamento de inteiros é finito, temos uma faixa de valores a se respeitar. O problema é que no RSA, as chaves precisam ser números primos bem grandes. Mas, no Java possuímos uma classe BigInteger, que facilita a manipulação de inteiros muito grandes. O que no caso, resolveu o problema.

Outra dificuldade encontrada foi saber o tamanho ideal do “p” e “q”. Pois, a multiplicação (p * q) deveria resultar em uma quantidade de bits aproximada ou igual a segurança solicitada (exemplo: 128 bits).

Uma outra dificuldade encontrada estava relacionada a mensagem. Como temos um número finito de inteiros, não poderíamos representar uma mensagem como um inteiro. Então, como aplicar a criptografia?! A solução foi utilizar a criptografia para cada byte ou char da mensagem.

Na implementação do DSA, houveram também dificuldades.

Uma dessas dificuldades foi encontrar uma maneira de encontrar um ‘q’ que divide ‘p-1’, sendo ‘p’ um primo.

Outra dificuldade, por sinal bastante duvidosa, foi a obtenção de um ‘alpha’ na qual ORD (alpha) = ‘q’. Após pesquisas, foi encontrada uma fórmula para a obtenção do ‘alpha’ e a questão foi solucionada.

Por último, queria destacar o problema em relação a demora da obtenção de p e q quando usamos o algoritmo do livro, por exemplo, quando quero um bit de segurança de 1024 bits.

Conclusão

A implementação do RSA e do protocolo de assinatura digital DSA foi de grande impacto e de crescente aprendizado. Entender como funciona, internamente, o conceito de chave pública e privada foi de extrema importância. Apesar das dificuldades encontradas ou erros que foram cometidos, podemos afirmar que os resultados foram satisfatórios.