Disciplina SSC0747 - Engenharia de Segurança

Relatório da Implementação da Smart Home do Johnny

Grupo:

Henrique Bonini Nº USP 8956690
Lucas Luchiari Nº USP 8084260

Sumário:

Introdução

Este trabalho tem por motivação na primeira parte a implementação de um sistema de criptografia para transmissão de mensagens entre “peers” - dispositivos da rede - que são dispositivos sensores e/ou atuadores dentro da Smart Home do nosso amigo João a uma estação central que computa as mensagens, executa os comandos e gera estatísticas, a fim de automatizar sua casa, tornando-a “*smart house*”, conceito definido em [1] como “Uma habitação que incorpora uma rede de comunicações que conecta os principais aparelhos e serviços elétricos e permite que eles sejam controlados remotamente, monitorados ou acessados”.

Para a segunda parte, uma série de melhorias serão sugeridas a fim de que nosso amigo João possa desfrutar da sua casa com segurança, sem que qualquer pessoa possa ameaçá-lo ou sua família.

Algoritmo Base: O RSA

Introdução ao RSA:

O algoritmo de criptografia RSA começou com Rivest, Shamir e Adleman, que foram os pesquisadores que conceberam a ideia do algoritmo e portanto deram nome ao mesmo. Esse atualmente tem sido amplamente utilizado no mundo todo na criptografia de diversas aplicações como: HTTPS, PGP, SSL, entre outros.

Esse algoritmo é classificado como algoritmo de chave pública ou assimétrica que se baseia na fatoração de números primos muito grandes para sua codificação. É composto de três etapas: Geração de Chave, Encriptação e Decriptação. É um algoritmo determinístico, portanto sempre uma mensagem criptografada será completamente recuperada sem perda.

Como é um algoritmo de chave pública, ou seja, geram-se duas chaves: a pública (podendo ser distribuída a qualquer usuário) e a privada (que deve ser bem guardada e jamais distribuída). Isso permite que a quebra do algoritmo ocorra apenas por meio da descoberta da chave privada ou descobrindo a combinação de primos utilizados na criptografia e fazendo o processo de decriptação, o que levaria muito tempo dependendo do tamanho da chave utilizada, tornando quase inviável um processo como o de força bruta, por exemplo.

Motivação para se usar RSA:

Neste trabalho optamos por utilizar o RSA devido à sua facilidade de entendimento e implementação, além de possuir muitas boas referências e ser um algorítmos relativamente simples e eficiente, mesmo com chaver menores que recomenda a literatura.

Estamos cientes de que não é viável fazer uma melhoria no algoritmo, uma vez que feito isso, poderíamos facilmente ficar internacionalmente famosos e nos renderia uma boa publicação. Com isso, fizemos um algoritmo baseado em RSA que criptografa com os mesmos princípios porém com forma diferente.

Algoritmo Implementado: *JohnnySHS RSA Based Algorithm*

O algoritmo consiste em uma versão modificada do RSA, com algumas simplificações e ajustável. Foi escrito na linguagem C++ versão 11 utilizando as bibliotecas padrão do *Standard Library*. O código é estruturado para ser modular e possui algumas bibliotecas externas que foram encontradas na internet para auxílio no trabalho com os números e operações matemáticas.

Ele foi implementado na interface do Qt, por isso é possível executá-lo em Qt, mas não é um requisito ter as bibliotecas do Qt, uma vez que não as estamos utilizando. A principal ideia do Qt seria da implementação de uma interface com o usuário caso o tempo permitisse. Infelizmente isso não foi possível mas a biblioteca está suficientemente organizada para tal.

O código é executado em linha de comando e possui a análise dos comandos passados para ele, inclusive um “help” explicando a maneira certa de executar.

Principais diferenças para o RSA Comum:

- Utiliza geração própria dos primos com resolução variável da chave definida pelo usuário;
- Realiza a encriptação caractere a caractere;
- Salva a mensagem como uma string de números e não como caracteres;
- Não possui otimização ou paralelismo;
- Não possui funções de *padding*.
- Não realiza a criptografia em conjunto com um método assimétrico;

Explicação do código:

Foi implementada o algoritmo RSA em uma classe própria que contém os métodos relativos para tal, separando logicamente o código. O algoritmo é dividido nas mesma três etapas do RSA padrão: Geração de Chaves, Encriptação e Decriptação.

Geração de Chaves:

Essa parte do código é essencial para o bom andamento do algoritmo, pois garante sua força. Como no RSA padrão, o algoritmo utiliza-se de dois primos extremamente grandes para a geração da chave. Um diferencial que implementamos foi a geração de tais primos, uma vez que eles podem simplesmente ser obtidos por tabelas, sem a necessidade de geração.

A geração de primos é realizada por meio de um algoritmo probabilístico chamado Miller-Rabin que verifica se um número é realmente primo com uma dada probabilidade, sendo essa certeza definida em código pelo número de iterações definido por nós.

O código que nos baseamos pode ser encontrado no link em [2]. Sendo assim, geramos uma *string* de inteiros aleatória do tamanho que queremos, por um gerador aleatório baseado na função rand() com semente no clock do computador. A string gerada é

passada pelo teste de Miller-Rabin para verificar sua “primacidade” e caso não seja primo, fazemos uma redução do valor até que um primo mais próximo seja encontrado.

Para gerarmos números inteiros tão grandes utilizamos uma biblioteca chamada **InfInt** (baseado em [3]) do que abstrai as operações com inteiros extremamente grandes e nos permite trabalhar com strings numéricas contendo o valor do inteiro.

A partir de dois primos gerados aleatoriamente, p e q, seguimos como no algoritmo obtido em [4], a seguir exemplificado:

1. Faz-se o produto **n = p*q** [Obs.1]
2. Computar **n = p*q** e **φ = (p-1)(q-1)**.
3. Escolher um inteiro e onde **1 < e < φ**, tal que **mdc(e, φ) = 1.**
4. Computar o exponente **d**, tal que **1 < d < φ**, e que **e*d ≡ 1 (mod φ)**.
5. A chave pública é o conjunto **(n, e)** e a privada o conjunto **(d, n)**.

Obs.1: Sendo p e q do mesmo tamanho ou variando em alguns dígitos em tamanho. (e.g. 1024 bits.)

Encriptação:

O processo de encriptação é feito utilizando-se da chave pública gerada na etapa anterior, por meio da seguinte equação:

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

Sendo que **c** é o criptograma gerado.

Em nossa implementação foi-se criptografado caractere a caractere da mensagem e o código resultante da encriptação é salvo em uma string, bem como um inteiro que indica o tamanho em caracteres do código gerado, para que se possa recuperar depois a informação. Esse *stream* de códigos é então salva em um arquivo em modo texto que é chamado de criptograma.

Decriptação:

A informação do criptograma pode ser obtida através do uso da chave privada com a seguinte equação:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Onde **m** é a mensagem em texto pleno obtida.

Em nossa implementação o processo reverso da etapa de encriptação é utilizado, onde tokens contendo o tamanho dos tokens criptográficos são utilizados para recuperar a informação e escrever em um arquivo de texto.

Estrutura do Código:
O código foi estruturado em uma pasta ‘source’, contendo a ‘main.cpp’, o ‘Makefile’ e mais duas pastas, ‘include’ e ‘src’, contendo, respectivamente, os arquivos ‘.hh’ e ‘.h’ e os arquivos ‘.cpp’.
No mesmo nível da pasta ‘source’, há as pastas ‘docs’ e ‘infint’, respectivamente com as documentações geradas e utilizadas e com o código fonte da biblioteca externa utilizada.

Esquema de arquivos gerados:
Os arquivos gerados pelo software são chamados ‘keys.pub’ e ‘keys.pri’, contendo, respectivamente, as chaves pública e privada.
Além disso, durante a encriptação, é gerado um arquivo de mesmo nome do de texto plano utilizado como entrada, porém com a extensão ‘.crypt’ contendo a cifra e sua separação e, durante a decriptação, é gerado um arquivo de mesmo nome com a extensão ‘.txt’, contendo o texto plano obtido a partir da decriptação da cifra.

Github
O repositório de código utilizado pode ser encontrado em https://github.com/lluchiari/Johnny_s_Smart_Home_Security__Security_Engineering.

Como executar
O software deve ser executado a partir de um terminal. O mesmo pode receber vários argumentos diferentes, como -h, -g, -e e -d.
O argumento -h ou --help mostra uma mensagem breve explicando os possíveis argumentos.
O argumento -g ou --generate-key deve ser escrito seguido de um número entre 16 e 4096, indicando o tamanho em bits da chave resultante gerada
O argumento -e ou --encrypt deve ser executado seguido do nome do arquivo contendo a chave pública e do arquivo contendo a mensagem a ser encriptada.
O argumento -d ou --decrypt deve ser executado seguido do nome do arquivo contendo a chave privada e do arquivo contendo a cifra gerada pela execução com o argumento anterior.

Aumento da segurança da *smart home* do Johnny
Para que Johnny aumente a segurança de sua *smart home*, próximos passos essenciais são a proteção de sua rede WiFi com uma senha criptografada, atualmente a criptografia comumente utilizada é a AES-256. Além disso, ele deve adicionar um firewall à sua rede, abrindo apenas as portas que realmente são necessárias.
Após essas implementações, Johnny pode criar uma VPN interna, onde a comunicação de todos os dispositivos é feita através de IPSec, ou seja, protocolo IP seguro, o qual transmite os pacotes de forma encriptada. Essas medidas de segurança, por si só, já aumentam significativamente a dificuldade de um atacante obter acesso não autorizado à *smart home*.
Em seguida, para se proteger de eventuais quedas de eletricidade e eventos semelhantes, Johnny deve programar seu servidor para efetuar backups criptografados frequentemente em um disco externo. Para ter um armazenamento realmente seguro e organizado das informações e logs, Johnny deve, também, implementar um banco de dados ao seu servidor, encapsulando o tratamento de informações.

Seguindo a linha de proteção contra eventos naturais e, para aumentar a disponibilidade das informações, incluindo disponibilização remota das mesmas, Johnny poderia pensar na transferência de seu sistema no servidor caseiro para um servidor na nuvem. Porém, para isso, deve ser considerado o custo do serviço face ao do servidor caseiro, ainda mais considerando que devem ser transmitidas as imagens das câmeras o mais próximo do tempo real possível. No entanto, há o risco de Johnny ter sua casa atacada por *black hat hackers*. Como esses hackers conseguiriam as imagens da câmera ou, no mínimo, conseguiriam colocar uma imagem falsa sendo transmitida (caso Johnny não tenha efetuado perfeitamente as configurações citadas anteriormente), a melhor opção para ele seria a implementação de seu sistema em um servidor na nuvem, que normalmente possui mais segurança por parte da empresa proveniente do serviço.

Referências:

[1] Jiang L., Liu D., Yang B., Smart Home Research, Third International Conference on Machine Learning and Cybernetics, Shanghai, 2629 August 2004.

[2] http://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/

[3] https://github.com/sercantutar/infint