Gerador/Verificador de assinaturas

João Victor de Souza Calassio - 180033808 William Coelho da Silva - 180029274

May 1, 2022

Universidade de Brasília - Instituto de Ciências Exatas
Departamento de Ciência da Computação - CIC
CIC0201 - Segurança Computacional - Turma B
Professor: João José Costa Gondim
Prédio CIC/EST - Campus Universitário Darcy Ribeiro
Asa Norte 70919-970 Brasília, DF

1 Introdução

Este projeto tem como objetivo implementar um gerador e verificador de assinaturas, utilizando cifras simétricas e assimétricas para transmitir mensagens de forma segura, e hashes para verificação de integridade da mensagem recebida.

Cifras simétricas são cifras que utilizam uma única chave para cifrar e decifrar textos. Dessa forma, é possível transmitir uma mensagem de forma segura, se ambas as partes (transmissor e receptor) conhecerem essa chave.

Já as cifras assimétricas utilizam um par de chaves, uma pública e uma privada. A ideia aqui é que, uma vez que um texto seja cifrado com uma chave pública, apenas quem possuir a chave privada possa decifrá-lo.

Os hashes são funções que mapeiam uma entrada de tamanho arbitrário, em uma saída de tamanho fixo, de forma que seja uniformemente distribuído. Isso quer dizer que, dada uma determinada entrada, a saída sempre será a mesma.

Em nosso trabalho, utilizamos uma cifra simétrica (AES) para cifrar determinada mensagem, uma cifra assimétrica (RSA) para cifrar a chave utilizada na cifração simétrica e poder transmiti-la de forma segura para o receptor, e uma função hash (SHA-3) para o receptor conseguir verificar a integridade da mensagem recebida.

Foi escolhido Python para implementação deste projeto por ser uma linguagem simples, tornando fácil a manipulação de textos (*strings*), que é nosso principal objeto de estudo neste trabalho.

2 Desenvolvimento

2.1 Assinatura RSA

Foram implementadas duas maneiras de implementação de encriptação e decriptação RSA, o padrão e OAEP.

2.1.1 Geração de chaves

O sistema de criptografia RSA (Rivest-Shamir-Adleman) se baseia em duas chaves: um pública e outra privada. A pública é usada para encriptar, ao passo que a privada é para decriptar e é secreta.

No caso deste projeto foram usados dois primos de 1024 bits gerando assim **n = p * q**, calculamos o totiente de Euler **phi**, escolhemos um número aleatório **e** entre um e esse número e esse dois elementos (n, e) formam a chave pública. Já para a privada, é calculado o inverso multiplicativo **d** de **e** e **phi** e daí temos a chave privada (n, d).

2.1.2 OAEP

O OAEP *Optimal Asymmetric Encryption Padding* é um esquema de preenchimento frequentemente usado em conjunto com a criptografia RSA.

Para encriptar nós primeiramente criamos uma assinatura que é um bytearray usando o plainText e a privateKey, pegamos o texto e aplicamos a função SHA-3 nele, depois transformamos o bytearray gerado em um inteiro, fazemos a potenciação deste inteiro a **e** módulo **n** e depois retornamos o bytearray deste resultado e o armazenamos. Usamos a mensagem (plainText) e a chave pública para encriptar da maneira disponível aqui.

Já para decodificar o texto cifrado, nós utilizamos a assinatura do texto, o texto encriptado, a chave pública e a chave privada para isso decriptamos o texto cifrado e o comparamos com o inverso que foi feito na assinatura, se bater, a decriptação foi concluída com sucesso.

2.2 AES-CTR

O Advanced Encryption Standard foi anunciado originalmente pelo NIST (U.S National Institute of Standards and Technology) em 2001, através da publicação FIPS 197.

Na publicação, é descrito o funcionamento do algoritmo, os princípios matemáticos por trás dele, além de exemplos descrevendo o comportamento dos dados após serem submetidos a cada uma das etapas.

O AES é uma cifra de bloco, com cada bloco sendo um array de exatamente 16 bytes. Esse bloco é transformado numa matrix 4x4 chamado *State*, onde são feitas transformações em uma ordem específica para se chegar no bloco cifrado.

Essas transformações são:

- **AddRoundKey**: adicionar a chave ao *State* utilizando um XOR
- **SubBytes**: substituição de bytes do *State* baseado em uma tabela pré definida (S-box)
- **ShiftRows**: rotacionar ciclicamente as últimas três linhas do *State*
- **MixColumns**: misturar os dados de cada uma das colunas (individualmente) para produzir novas colunas

Implementamos as transformações no *State*, bem como o algoritmo para cifrar cada bloco baseado nos pseudocódigos e exemplos descritos na publicação.

Não foi necessário implementar a decifração descrita no artigo, bem como as inversas de cada uma das transformações, por motivos específicos ao modo de operação CTR que será descrito abaixo.

2.2.1 Geração de chave

A geração de chaves simétricas para o AES não tem um algoritmo específico, pois basta gerar uma sequência aleatória de bits (no caso do AES, o padrão é 128, 192 ou 256 bits).

No entanto, é importante que essa sequência seja suficientemente imprevisível. Utilizamos a função nativa do Python *urandom*, que faz uma chamada ao sistema operacional e que tem fontes de entropia menos previsíveis, se tornando um gerador seguro o suficiente para uso criptográfico.

2.2.2 Modo de operação CTR

No modo de operação CTR (counter), há um contador que produz uma sequência que garantidamente não irá se repetir por um longo tempo. Esse contador pode ser inicializado por um vetor de inicialização (IV, ou nonce), e essa sequência é a entrada para cifração de bloco descrito pelo AES.

Ciframos o valor atual do contador com a chave, e o resultado dessa cifração é utilizado em um XOR com o bloco de 16 bytes da mensagem. O resultado dessa operação é o bloco cifrado.

Essa operação é repetida, até todos os blocos terem sido cifrados:

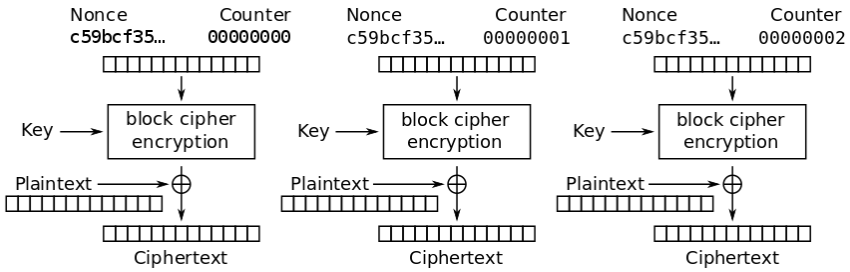

Figure 1: Cifração no modo CTR

Já para a decifração, o mesmo processo é feito, invertendo os papeis do texto cifrado e da mensagem original. Sendo assim, fazemos a **cifração** do contador com a chave (por esse motivo, no modo CTR não utilizamos a decifração de bloco do AES), e o resultado dessa operação é utilizado no XOR com o texto cifrado. Como o XOR é inversível, o resultado disso será o texto que foi utilizado originalmente na cifração:

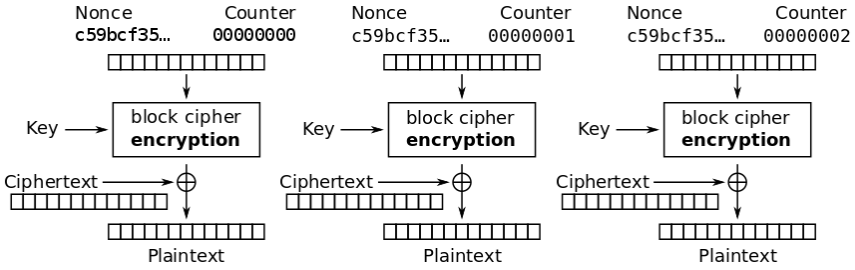

Figure 2: Decifração no modo CTR

2.3 Programa

Para simular a transmissão e recepção de mensagens, o programa principal foi separado em duas classes um transmissor, e um receptor. O objetivo disso é tornar os ambientes de transmissão e recepção isolados, não compartilhando nada além do que seria compartilhado numa transmissão real: textos cifrados e chaves públicas.

Cada uma das partes (transmissor e receptor) tem um par de chaves RSA próprio, que é gerado no inicio da execução do programa, e as chaves públicas são conhecidas por ambas as partes.

O transmissor recebe a mensagem que será transmitida, e a chave pública do receptor.

2.3.1 Transmissor

A transmissão é feita seguindo as seguintes etapas, sequencialmente:

1. É gerada uma chave de sessão de 128 bits

2. A mensagem é cifrada com o AES-CTR, utilizando a chave gerada
3. A chave gerada é cifrada com o RSA OAEP, utilizando a chave pública do receptor
4. É gerada a assinatura SHA-3 da mensagem
5. A assinatura da mensagem é cifrada com o RSA, utilizando a chave privada do transmissor
6. A assinatura cifrada, chave cifrada e mensagem cifrada são codificados em BASE64 para transmissão e retornados

O receptor recebe a assinatura cifrada, chave cifrada, mensagem cifrada e a chave pública do transmissor.

2.3.2 Receptor

O recebimento é feito seguindo as seguintes etapas, sequencialmente:

1. A assinatura cifrada, chave cifrada e mensagem cifrada são decodificados do formato BASE64
2. A chave de sessão é decifrada utilizando a chave privada do receptor
3. A mensagem é decifrada utilizando a chave de sessão
4. É calculado o hash SHA-3 da mensagem decifrada
5. A assinatura recebida é decifrada com a chave pública do transmissor

Ao fim, a mensagem decodificada é exibida, e as assinaturas são comparadas. Se forem iguais, então sabemos que a mensagem foi transmitida sem erros e sem interferências.

3 Conclusão

Com este projeto foi possível visualizar a complexidade de se gerar cifras realmente seguras, ou seja, computacionalmente difíceis/inviáveis de serem quebradas e até mesmo geradas, já que é necessário alguns segundos toda vez que precisamos gerar nosso par de chaves públicas e privadas.

Também foi possível entender as vantagens e desvantagens entre os dois esquemas criptográficos implementados, e entender as aplicações práticas de cada algoritmo de cifração. Abordar as diferentes formas de se trabalhar e implementar os poderosos métodos RSA e AES foi enriquecedor.

Nos dias atuais, é estritamente necessário entender como funcionam esses algoritmos e seus casos de uso, para garantir a integridade e segurança de qualquer informação que seja armazenada ou transmitida e que possa impactar a vida de diferentes pessoas, empresas, governos, ou qualquer entidade que dependa de meios computacionais para esses fins.

4 Referencias bibliográficas

ADVANCED ENCRYPTION STANDARD (AES) - Federal Information Standards Publication 197
Advanced Encryption Standard - Wikipedia
The difference in five modes in the AES encryption standard
Block cipher mode of operation - Wikipedia
Online AES Encrypt and Decrypt
RSA (cryptosystem)
Optimal asymmetric encryption padding