

UnB - Universidade de Brasília

CIC0201 - Segurança Computacional – 2022/2

Professor: João Gondim

Aluno: Igor Silva de Oliveira Cardoso (160125073)

Trabalho de Implementação 2 - Gerador/Verificador de Assinaturas

Estrutura do projeto e Recursos

O projeto foi implementado utilizando a linguagem de programação Python, e está estruturado em classes. Há testes de execução dentro de cada classe, entretanto, o fluxo de execução inicia a partir do arquivo `run.py` ou o `run_failure_case.py`, sendo que a execução do `run_failure_case.py` é um caso com em que a mensagem é comprometida, não sendo possível validar a assinatura. Os dois fluxos podem ser executados com os comandos `python3 src/run.py` ou `python3 src/run_failure_case.py` a partir da raiz do projeto.

Teste de primalidade por Miller-Rabin (Classe **MillerRabin)**

Os testes de primalidade buscam determinar se um número é primo ou composto. Sendo que, a sua utilização é necessária por conta da criptografia RSA, necessita de números primos muito grandes para garantir sua segurança. O Teste de Primalidade de Miller-Rabin usa como base o Teste de Primalidade de Fermat. Para implementação do método `__primality` da classe **MillerRabin** foi utilizado o algoritmo que encontra-se nas referências. Em que por meio do método `primeGenerate` são gerados um número primo a partir de um tamanho (em bits) previamente definido pelo atributo da classe **MillerRabin**, e esse número é verificado pelo método `__primality`.

Descrição da classe:

- **keyLen**: atributo de classe que define o tamanho do primo gerado em bits.
- **primeGenerate(self, **args)**: método responsável por gerar um número primo de tamanho **keyLen** usando o método `getrandbits` da classe **random**, e também garantir que o número é primo através do método `__primality`.
- **setBit(self, value, bit)**: método privado responsável por fazer os “sets” nos bits do número gerado para garantir que ele é ímpar.
- **__primality(self, n: int, t=2000)**: método privado responsável por realizar o teste de primalidade no número gerado. Recebe um inteiro “**n**” e um valor “**t**”

que indica a quantidade de testes, sendo que há um valor default para “t” de 2000.

Cifragem/Decifragem usando o RSA (Classe **RSA**)

O RSA é basicamente o resultado de dois cálculos matemáticos. Um para cifrar e outro para decifrar. O RSA usa duas chaves criptográficas, uma chave pública e uma privada. No caso da criptografia assimétrica tradicional, a chave pública é usada para criptografar a mensagem e a chave privada é usada para descriptografar a mensagem. A classe **RSA** é responsável por realizar todas as implementações das funcionalidades do algoritmo RSA usando OAEP, hashing SHA3 e o Base64.

A classe **RSA** herda a classe **MillerRabin**, e por meio dela são gerados os primos, e em seguida são gerados as duas chaves pelo método **generateKeys**, após a geração das chaves é realizado o preenchimento a clara da mensagem usando o método **oaep**, realizando a assinatura da mensagem por meio da equação:

$$M_{Cifrado} = oaep(M_{Claro})^{kU[0]} \bmod kU[1]$$

Já a verificação e a cifragem da mensagem é realizada por meio das chaves privadas por meio da seguinte equação:

$$M_{claro} = reverseOaep(M_{Cifrado}^{kR[0]} \bmod kR[1])$$

Descrição da classe:

- **kU**: atributo da classe que é uma tupla com a chave pública
- **kR**: atributo da classe que é uma tupla com a chave privada
- **DEFAULT_SIZE_PADDING**: atributo constante da classe que representa um tamanho padrão do preenchimento
- **oaep_send** e **oaep_receive**: atributos da classe que representam o hash de envio e recebimento da mensagem, que é utilizado para verificar assinatura e autenticidade da mensagem.
- **__coprime**: método privado responsável por verificar se dois números são coprimos.
- **generateKeys**: método responsável por gerar as chaves: privada e pública.
- **__XOR**: método privado responsável por realizar a operação XOR (OR exclusivo) para cada posição da string de bits.

- **__strToByte**: método privado responsável por realizar a conversão de binário (string) para bytes.
- **__bitStrToBytes**: método privado responsável por realizar a conversão de bit em forma de string para byte.
- **toBase64**: método responsável por realizar o base64 de string.
- **oaep**: método responsável por realizar o preenchimento da mensagem. Nesse método é obtido os valores do dict que são as duas partes do hash do envio da mensagem.
- **reverseOaep**: método responsável por realizar o processo inverso do preenchimento da mensagem. Nesse método é obtido os valores do dict que são as duas partes do hash de recebimento da mensagem.
- **__encoderFunction**: método privado auxiliar que é responsável por efetuar o cálculo da cifração dos caracteres da mensagem.
- **__decoderFunction**: método privado auxiliar que é responsável por efetuar o cálculo da decifração dos caracteres da mensagem.
- **encoder**: método responsável por realizar a chamada do do método que realiza o preenchimento e cifração da mensagem. Nesse método foi implementado uma flag para indicar o uso do preenchimento oaep.
- **decoder**: método responsável por realizar a chamada do método que realiza a decifração da mensagem e desfazer o preenchimento.

```

→ trab2-igor-cardoso git:(master) x python3 rsa.py
Message:  Igosr Cardoso 123
Chaves:
    kU:  (3811, 34393)
    kR:  (5731, 34393)

Encoder message: 4bWC4bWC5LeJ5LeJ54uFAeKKjuG1guS3ieS3ieCimeeLheeLhQDdio7g
opnni4Xkt4kBS4uF45eZ4oq054uF4bWC5LeJ4KKZ54uF5ama5LeJ4oq045eZ4KKZ4KKZ4
bWC4bWC45eZ4bWC45eZAOKKjuKKjuKKjuW2hOCimQDhtYLj15kA5amaAOWpmuW2hAHio7ni4
Xj15kA4KKZ4bWC4oq054uFAOCimeS3ieG1ggHhtYLio4A5baE4oq054uF4oq054uF4oq045e
Z4oq04bWC4oq04KKZ45eZ5LeJ45eZ45eZAOKKjuKKjgDgopnni4UBAeOXmeW2hOWpmuS3ieCi
meOXmeCimeG1guKKjuW2hAEA54uFAQDhtYLGopnkt4kA4oq045eZ54uFAeW2h0OXmeeLhQDni
4UB5LeJ5LeJAeWpmuCimeOXmeCimeOXmeWpmgDj15kB4KKZ54uFAeWpmgDni4XltoIio7kt4
niio7kt4nni4UBAOG1gueLheOXmeKKjuCimeCimeW2hOCimQHj15nlqZrj15kB45eZ5ama45e
Z45eZ5ama4oq05LeJ4KKZ5baE5LeJAeCimeS3iQHhtYLGopnj15nkt4kAAOCimQDdio7iio4B
AeWpmueLhQDj15nlqZrj15kB45eZ4KKZAeWpmuW2hAAB4bWC4bWC5LeJ4bWC45eZ45eZ4
5eZ4KKZAeOXmeW2hAAB5LeJ5amaAQDkt4kB4oq05ama54uFAAHj15kA54uF54uF4bWC

Dencoder message: Igosr Cardoso 123
oaep_send -> {'hashPart1': 'OTMwMjkzMTc1NzgyMjYzMjE4OTM0NjI1MzAyNjY2NTEwM
TQ1MzM2NjMxNTYwNDc1MjYxMzQxMzQ4ODU1MDI3ODgzNzUwNDc3ODk1NDg5MDIxMTg3MDAwND
A3OTc0MDk0OTIyOTQxNDM3Mjg5MjUxMDQxNzI1MzU1MDM5MDk1Nzc3OTk1NzU2ODk5NTQwMjg
wNTkyMzI4OTU2NQ==', 'hashNonce': 'MTY1NzgzMjg2MjMyMjIwNjIxNjE5NDk4NDY4MTI
0MjAwNDQxNzE0MTE2NTI4NTIzODE0OTk4MzIzMTg3ODE4NTc2ODA1NDAwNTQ4MzA='}
oaep_receive -> {'hashPart1': 'OTMwMjkzMTc1NzgyMjYzMjE4OTM0NjI1MzAyNjY2NT
EwMTQ1MzM2NjMxNTYwNDc1MjYxMzQxMzQ4ODU1MDI3ODgzNzUwNDc3ODk1NDg5MDIxMTg3MDA
wNDA3OTc0MDk0OTIyOTQxNDM3Mjg5MjUxMDQxNzI1MzU1MDM5MDk1Nzc3OTk1NzU2ODk5NTQw
MjgwNTkyMzI4OTU2NQ==', 'hashNonce': 'MTY1NzgzMjg2MjMyMjIwNjIxNjE5NDk4NDY4
MTI0MjAwNDQxNzE0MTE2NTI4NTIzODE0OTk4MzIzMTg3ODE4NTc2ODA1NDAwNTQ4MzA='}
>>> Assinatura pode ser confirmada!

```

Imagem 1 - Teste de execução da classe RSA

Arquivos de fluxo de execução

A execução do programa pode começar pelo arquivo run.py em que será solicitado o tamanho da chave e a mensagem a ser enviada. Entretanto, o programa pode ser executado pelo arquivo run_failure_case.py que simula um caso que a mensagem é corrompida, não sendo possível autenticar a confiabilidade da mensagem.

Considerações

Durante o desenvolvimento do trabalho foram encontrados vários desafios, entretanto foi possível implementar todas as etapas propostas, o algoritmo de cifragem usando o preenchimento está bastante ineficiente para mensagens e chaves longas.

Referências Bibliográficas

- 1 DE LIMA, Daniel Chaves. Trabalho de Conclusão de Curso: **Algoritmo para o teste de Primalidade de Miller-Rabin**. Universidade Federal do Pará Campus de Castanhal: Faculdade de Matemática, 2019.
- 2 ANDERSON, Ross J.. **Security Engineering: A Guide to Building Dependable Distributed Systems** Second Edition. Indianapolis, Indiana: Wiley Publishing, Inc., 2008.
- 3 BARBOSA, Luis Alberto De Moraes; BRAGHETTO, Luis Fernando B; BRISQUI, Marcelo Lotierso; DA SILVA, Sirlei Cristina. **RSA Criptografia Assimétrica e Assinatura Digital**. Campinas: UNICAMP – Universidade Estadual de Campinas, 2023.
- 4 SHOUP, Victor. **OAEP Reconsidered**. Switzerland: IBM Zurich Research Lab, 2001.
- 5 KATZ, Jonathan; LINDELL, Yehuda. **Introduction to Modern Cryptography**. Boca Raton London New York Washington, D.C: CRC PRESS, 2007.