Segurança Computacional
Trabalho 2 - Gerador/Verificador de Assinaturas

Mateus Freitas Cavalcanti
16/0137519
mat.fcavalcanti@gmail.com

Mariana Mendanha da Cruz
16/0136784
mariana.mendanha.mm@gmail.com

Abstract

Este trabalho explora os conceito de assinatura digital, tal como cifração simétrica e assimétrica de mensagem. Para a cifração assimétrica, é implementado o algoritmo RSA OAEP (Optimal Asymmetric Encryption Padding), tendo em vista a geração completa do par de chaves (pública e privada) e teste de primalidade (Miller-Rabin). Por outro lado, para a cifração simétrica, é implementado o algoritmo AES CTR (Counter Mode). Por fim, foi implementado um sistema cliente/servidor com o objetivo de aplicar o conceito de assinatura digital, utilizando a cifração RSA OAEP implementada durante o processo.

1. Introdução

1.1. Criptografia simétrica e assimétrica

Algoritmos de encriptação são frequentemente divididos em duas categorias, conhecidas como encriptação simétrica e assimétrica. A diferença fundamental entre esses dois métodos de encriptação se baseia no fato de que os algoritmos de encriptação simétrica fazem uso de uma única chave, enquanto a encriptação assimétrica faz uso de duas chaves diferentes, porém relacionadas. Tal distinção, embora aparentemente simples, aponta as diferenças funcionais entre as duas formas de técnicas de encriptação e as maneiras como elas são usadas.

1.1.1 RSA

O algoritmo RSA é um método de criptografia assimétrica cujo funcionamento consiste na utilização de um par de chaves para codificação e decodificação, ou seja, cada chave do par é utilizada somente em uma operação. Esse algoritmo é considerado dos mais seguros e foi o primeiro a possibilitar criptografia e assinatura digital. Para a cifração RSA, a chave é utilizada para aplicar a operação de potenciação modular, ou seja, cada caractere da mensagem em claro passa pela operação mostrada na figura 1.

$$m^e \equiv c \mod n$$

Figure 1. Algoritmo de cifração RSA.

Para a decifração, é necessário fazer a mesma operação modular com a chave complementar como mostrado na figura 2. Sendo assim, a partir da primeira operação, tem-se a mensagem cifrada:

$$c^d \equiv m \mod n$$

Figure 2. Algoritmo de decifração RSA.

1.1.2 RSA OAEP

Tendo em vista o funcionamento do algoritmo RSA, pode-se definir também o modelo RSA OAEP. O *Optimal Asymmetric Encryption Padding* (OAEP) é um esquema de preenchimento frequentemente usado junto com a criptografia RSA. Esse algoritmo utiliza um par de operações G e H para processar a mensagem em claro antes da criptografia assimétrica. O esquema de cifração e decifração deste algoritmo pode ser visualizado na figura 3.

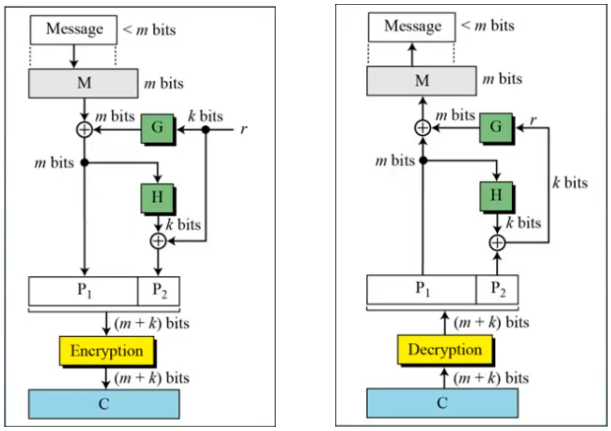

Figure 3. Algoritmo de cifração (esquerda) e decifração (direita) RSA OAEP.

A utilização dessas operações, juntamente com a cifração assimétrica, transforma o RSA OAEP em um modelo comprovadamente seguro contra diversos ataques (ataques de texto cifrado escolhido, por exemplos).

1.1.3 AES

No caso da criptografia simétrica, é possível definir o padrão de criptografia avançada (ou ***Advanced Encryption Standard*** - *AES*) como um padrão de criptografia, sendo aprovado pela agência de segurança nacional dos EUA. Esse algoritmo é baseado na cifração em blocos, ou seja, divide a mensagem em claro em estruturas matriciais (blocos) e aplica a cifração simétrica em cada bloco, de acordo com a figura 4

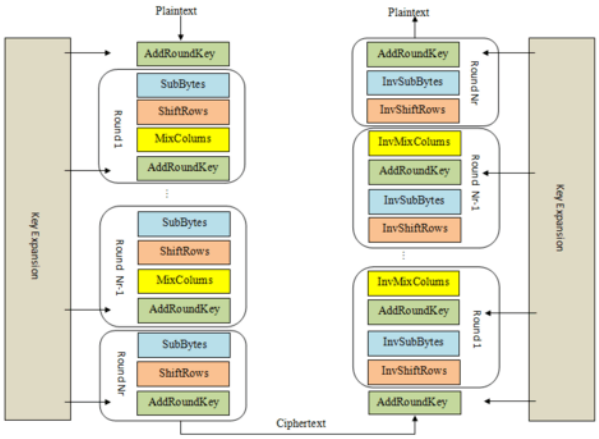

Figure 4. Algoritmo de cifração e decifração AES.

A implementação do algoritmo do AES consiste em uma sequência de operações cíclicas (AddRoundKey, SubBytes, entre outros) que podem ser devidamente verificadas na página sobre AES Encryption.

1.1.4 AES CTR (Counter Mode)

Visando a criptografia de mensagens de tamanhos arbitrários, vários modos de operação foram inventados, os quais permitem que o ciframento em blocos forneça confidencialidade. Sendo assim, novas propostas para o AES foram criadas. Dentro delas, foi desenvolvido o modelo **AES CTR**, onde cada bloco de texto claro passa pela operação de 'ou exclusivo' (XOR) com um contador criptografado (incrementado para cada bloco subsequente). O funcionamento para cifração e decifração de mensagem utilizando o AES CTR pode ser verificado nas figuras 5 e 6, respectivamente.

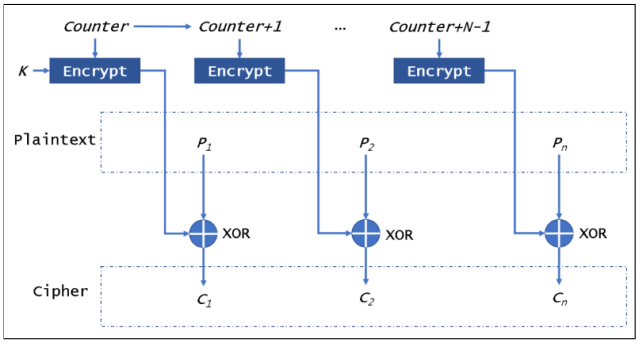

Figure 5. Algoritmo de cifração AES CTR.

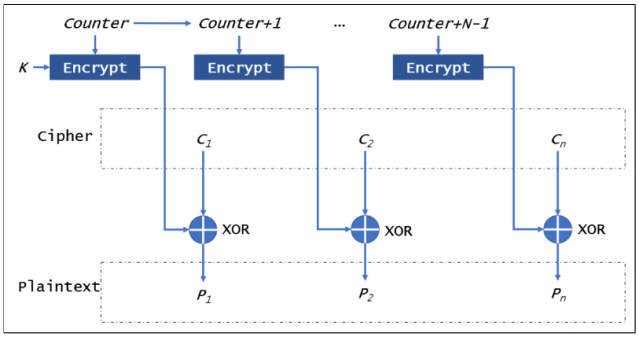

Figure 6. Algoritmo de decifração AES CTR.

Uma vez que os blocos de criptografia do AES são de 128 bits, é necessário adicionar um *nonce* a cada interação do contador para atingir o tamanho dsejado do bloco. Considerando o contador de 64 bits, o *nonce* gerado aleatoriamente deve ter 64 bits.

1.2. Assinatura digital

O conceito de assinatura digital é determinado pela necessidade de validação de documentos enviados através de rede de comunicação. O modelo de assinatura digital utilizado neste documento pode ser verificado na figura 7.

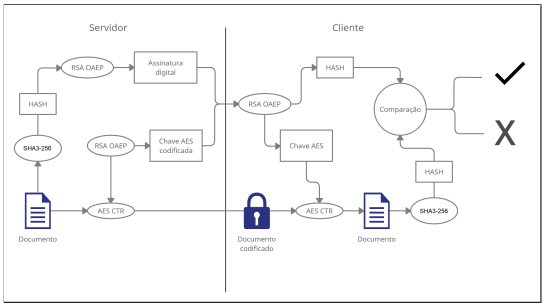

Figure 7. Modelo de assinatura digital.

É possível verificar, pelo modelo adotado, a validação de documentos enviados pelo servidor, uma vez que, tendo o hash codificado da mensagem original e a própria men-

sagem codificada, é possível comparar a autenticidade do documento recebido pelo remetente.

2. Materiais e Métodos

A implementação de todo o algoritmo de geração e verificação de assinatura digital usando RSA OAEP e AES CTR foi feita utilizando a linguagem **python**, por meio da ide **Visual Studio Code**. O sistema implementado segue um modelo cliente-servidor, feito com base no protocolo de comunicação HTTP, onde o servidor hospeda um serviço REST API e recebe requisições do cliente (requisição de documentos assinados).

2.1. Parte I - Geração de chaves e cifra simétrica

De maneira inicial, é necessário gerar as chaves para efetuar as cifras simétricas e assimétricas do processo de assinatura.

2.1.1 Chaves assimétricas (Chaves pública e privada)

A geração de chaves assimétricas depende diretamente da definição de números primos **p** e **q** iniciais. Considerando p e q de **1024 bits**, é possível utilizar o teste de primalidade de Miller-Rabin com o objetivo de se determinar números primos de 308 dígitos (1024 bits). Tendo p e q, basta seguir o algoritmo para se encontrar **N**, **E** e **D**. Sendo assim, as chaves pública e privada podem ser dadas por

- Chave privada: (D, N)
- Chave pública: (E, N)

2.1.2 Chave simétrica

Para o caso da chave simétrica, considerando o uso do algoritmo AES CTR, basta gerar uma **string aleatória de 16 caracteres alfanuméricos**. Para os caracteres alfabéticos, são considerados somente letras maiúsculas. Um exemplo de chave simétrica gerada é dada por **CL5X7XM2BMRC5KUR**.

2.2. Parte II - Assinatura

Para a assinatura de documentos, é necessário separar o algoritmo implementado nas seguintes partes:

2.2.1 Cálculo de hash e codificação RSA OAEP

Inicialmente, é necessário calcular o hash da mensagem utilizando o algoritmo **SHA-3**. Logo em seguida, é feita a codificação do hash utilizando o modelo **RSA OAEP** proposto (figura 8).

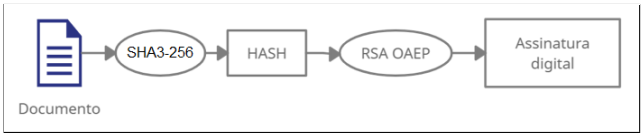

Figure 8. Cálculo de hash e codificação RSA OAEP.

2.2.2 Codificação AES da mensagem

Para efetuar a verificação do arquivo pelo cliente, é necessário enviar a mensagem encriptada. Sendo assim, é feita a codificação da mensagem original utilizando o algoritmo **AES CTR** (figura 9).

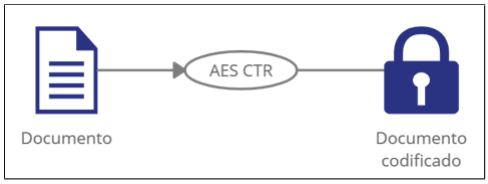

Figure 9. Codificação AES da mensagem.

2.2.3 Codificação RSA OAEP de chave simétrica

Por fim, para obter a mensagem decodificada pelo lado do cliente, é necessário enviar a chave simétrica e o *Nonce* utilizadas no algoritmo AES CTR. Sendo assim, é feita a codificação desses parâmetros utilizando o algoritmo **RSA OAEP** (figura 10).

Figure 10. Codificação RSA OAEP de chave simétrica.

Utilizando todas as informações obtidas nas sessões 2.2.1, 2.2.2 e 2.2.3, é possível montar o *payload* de envio de assinatura. Nesta estrutura estão presentes todas as informações que o lado do cliente precisa para validar a mensagem recebida (figura 11).

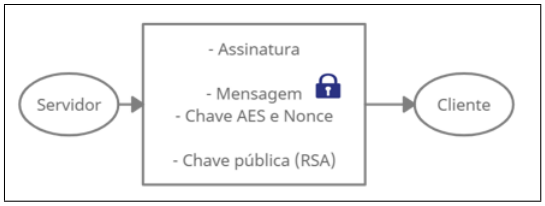

Figure 11. Envio de dados de assinatura.

2.3. Parte III - Verificação

No lado do cliente, é necessário fazer a verficação do documento assinado e conferir a autenticidade do mesmo.

Para isso, é necessário utilizar a chave pública recebida no payload e efetuar o processo inverso da seção 2.2, ou seja:

1. Decodificar a assinatura para obter o hash da mensagem.
2. Decodificar a chave AES e Nonce.
3. Decodificar a mensagem utilizando o algoritmo AES CTR.
4. Calcular o hash da mensagem decodificada e comparar com o hash obtido com o passo 1.

Sendo assim, uma vez que os hashs comparados são iguais, a mensagem é válida. Caso contrário, a mensagem recebida não é aceita. A representação do algoritmo de verificação de assinatura pode ser visualizada na figura 7, no lado do cliente.

3. Resultados

O sistema implementado possui 3 modos de operação:

3.1. Geração de chaves assimétricas

O modo de geração de chaves deve ser executado de maneira inicial, ou seja, antes de abrir o servidor ou executar o cliente, é necessário gerar o par de chaves de acordo com a linha de comando da figura 12. Gerar o arquivo de chaves antes da execução dos outros modos é essencial, uma vez que as chaves pública e privada são retiradas do arquivo 'KeyPair.kp'.

3.2. Assinatura de mensagem

Para este modo de operação, o cliente deve explicitar a flag '- -raw' na linha de comando para que o servidor envie uma mensagem de texto simples assinada. Sendo assim, o servidor deve fornecer uma mensagem de texto (string) para assinatura por meio de entrada de dados no terminal. O funcionamento deste modo pode ser exemplificado nas figuras 13 e 14, que mostram o lado do servidor e cliente, respectivamente.

3.3. Assinatura de arquivo

No modo de assinatura de arquivos, o cliente deve explicitar a flag '- -file' e, logo após especificar o arquivo desejado, envia uma requisição de assinatura do arquivo ao servidor. No outro lado, o servidor procura pelo arquivo no diretório './server_docs' e efetuar a assinatura do seu conteúdo. A partir disso, o funcionamento é análogo ao do modo anterior e pode ser visto nas figuras 15 e 16.

O funcionamento da assinatura para arquivos não foi otimizado, isto é, pra documentos grandes (pdf, por exemplo) o servidor faz a codificação do documento utilizando o AES CTR de maneira sequencial. Visando a melhoria do sistema, pode-se implementar o AES CTR de maneira paralela, uma vez que esse algoritmo permite a codificação simultânea de diversos blocos.

```
E:\Users\droto\Desktop\Trab2_SC>python DocSigner.py genkeys
 [!] Gerando arquivo 'KeyPair.kp' ... Done.
```

Figure 12. Geração de arquivo com chaves assimétricas.

```
E:\Users\droto\Desktop\Trab2_SC>python DocSigner.py server
http server is starting...
http server is running...

[?] Enter string message to sign: This is a secret message ;) shiii

 > Plain: This is a secret message ;) shiii
 [*] Gerando hash da mensagem ...
  |> Plain hash: 6e99591032e3229be07f8c496213cd78fd822f4b2cd39a0ed404c437c6515e2e
 [*] Importando chave privada do arquivo arquivo 'KeyPair.kp' ... Done.
  |> Codificando Hash usando RSA OAEP ...
 [*] Gerando chave AES e nonce ...
  |-[$] Key: BX31PY5G9OAKFXZ0
  |-[$] Random generated Nonce: 0010110111011001101110010100001000100010011000111011101001010101
     |> Codificando chave AES e nonce usando RSA ...
 [*] Codificando mensagem usando AES CTR ...

 [$] Assinatura completa, enviando para o cliente

127.0.0.1 - - [01/May/2022 14:08:18] "POST / HTTP/1.1" 200 -
```

Figure 13. Assinatura de mensagem texto (Servidor).

```
E:\Users\droto\Desktop\Trab2_SC>python DocSigner.py client --raw

 [...] Aguardando documento ...

------------------------------------------------------------
PAYLOAD
Encoded Doc length: 96 bytes
Encoded AES key (BASE64): ['KbwJh8j+K67UnF9EFqTlFfUb605fZcQT2LctyX5YBReR/aaMmoMKvBlIyIxAw6BnZqzQVaooruv2UaVVyQ24tbqi+p0
TBtLcrFVQ9YPz1H20r7GBj/c3y1XZYxSqKWQuK93Z9A0+YR54FbB/eh2sWMngGkU0GQFHozDxaVTj8Kri8jzb8KGqSQuW61ovEASDQ2lo7W8R7uFzL6Ohaz
upcmSARokwE9lDJI6BrmZC/3muwjhTvdjz7ahclmRhg6LDGBKVmzd/7WlH3Jugp9xofhK38fRqSHJpI+RpzTTAOxzdqF91wL0apPmQRLhD3JpwOYv4E7vA
WJZbi3EVN0qhA==', 'IteAITOTBrdLUKtg88MGwU9oKSNGJwcE+fOcwn/HPiPIY71cZGijBY2ll+Y0pQSEAG5HPRLM6AwKpoVsYwSXG67VD7F2QzEw6bbk
DE5mhyA43w6NdZEFXYBoZM8IG4J65rn+HBKqgVnXcJsQK2laDU48DwNLiW84eg/y9urX45qzkqEPY/IZ3mO1sN2iFIAv97GQ04U42EYkOGbFj8NKI4dvY8n
DCp8pdkWJ1sZhe6OBULVA1NP7zAhlx7VR+4wVHoyFTh6Fnu70sLPUqT9dN4l1PsBnMcfrFedbHFWTejBrZtf/dcTDOWe5xyTWEGo57KXZ9H8XDc78NNC2v0
9X/g==']
Digital signature (BASE64): auB5LVnuUv4Vj6TiRfmaG6Lk/2E+LvHDhUMgfmQDXjSoW4SneGkzD44m+A/hfpt/gQNFFmcE3eBAfVZ7ysYfs8sVC9x
S5x/mIs4wkJ0o7ys0dv4gfmAzGcxuVRnpiGFVI3IctFrdFWsNXpLeeIOit6rFMhQJto07CAMcKFWORJKuq81PN7/rGCCI2lodXUBFfP8JJUaH5DmUppWGxg
T3VZ0bRhjgU7fCJMyu2vOFh5t6rPKvP2ud6i2lUpXCDVal+xXWiQO8tPGrh9jRbAGjPK2X+rUvjK0ygJoxtC3/6SA5VDH0TF5MQL8X9c7kN8NPbuj9VG4m8
qDxKwNc6iZYdw==
------------------------------------------------------------

 [!] Verificando assinatura ...
 [*] Importando chave pública do arquivo arquivo 'KeyPair.kp' ... Done.
 [*] Decodificando hash recebido e chave AES ...
  |-[$] Hash (recebido): 6e99591032e3229be07f8c496213cd78fd822f4b2cd39a0ed404c437c6515e2e
  |-[$] Key: BX31PY5G9OAKFXZ0
  |-[$] Nonce: 0010110111011001101110010100001000100010011000111011101001010101
 [*] Decodificando documento recebido usando AES CTR ...
  |> [████████████] 48/48
 [*] Gerando hash da mensagem ...
  |-[$] Hash (calculado): 6e99591032e3229be07f8c496213cd78fd822f4b2cd39a0ed404c437c6515e2e
 [*] Comparando hashes ...

>>> Esperado (hash): 6e99591032e3229be07f8c496213cd78fd822f4b2cd39a0ed404c437c6515e2e
>>> Recebido (hash): 6e99591032e3229be07f8c496213cd78fd822f4b2cd39a0ed404c437c6515e2e

 [$] Mensagem é válida :)

 [$] Mensagem: This is a secret message ;) shiii
```

Figure 14. Assinatura de mensagem texto (Cliente).

```
E:\Users\droto\Desktop\Trab2_SC>python DocSigner.py server
http server is starting...
http server is running...

 > Filename: secret.jpg
 [*] Gerando hash do arquivo ...
  |> File hash: 7ecf105ebf8d517b3968397a093c565e28469559bee88efc11f5f9af0fafdf27
 [*] Importando chave privada do arquivo arquivo 'KeyPair.kp' ... Done.
  |> Codificando Hash usando RSA OAEP ...
 [*] Gerando chave AES e nonce ...
  |-[$] Key: 7YVIHKFORHQSPPCM
  |-[$] Random generated Nonce: 0000101010011111110000110000110111011110000111010101011001101011
     |> Codificando chave AES e nonce usando RSA ...
 [*] Codificando arquivo usando AES CTR ...
  |> [████████████] 19928/19928
 [$] Assinatura completa, enviando para o cliente

127.0.0.1 - - [01/May/2022 14:05:17] "POST / HTTP/1.1" 200 -
```

Figure 15. Assinatura de documento (Servidor).

```
E:\Users\droto\Desktop\Trab2_SC>python DocSigner.py client --file

 [?] Arquivo para assinatura: secret.jpg

 [...] Aguardando documento ...

------------------------------------------------------------
PAYLOAD
Encoded Doc length: 39872 bytes
Encoded AES key (BASE64): ['fXeRdY3wAjr4rMaQQcpGYnqvGhMcU8Ul2DJAvVtGprgyQhQhP2m3GHqrWuUIJEqTvSPeItz7m014YOmRz2+kF8a3T6
curzgsZgrCQegrOM2165IwfnTGiaQIyP6G7a0L49QI6XUcShQPuzqIUj6/VTzuZQI8rvx9xHjHxeKdqhY02XQ7q/UjcZ9FbLTY2qmRYekqFDzN7KGTW8Ro7
kOqqLO/dyUIQy5PDTk4XOty+8O/NM4pe8cauwN0AWwOEvo7AdfLHiptu8Xnt6hJhR4vVJypxnc9EsvCSF8lvX4UwQs55x6ehKPNWDOC083ZVoRJeYWTIfA
QZT0JPoLkjcpA==', 'PV03eai07vD9ZFr1jr6DYjMeY54agAofeILMQNcd0aIE3FIRJ/+fs7TxiDJw+JLksqevUtKo+JgOSx84/6iOXImasP9Qo8pgkxMW
shX80wNdyGdXsMRsQWxOCbjVyalWF1GdtypMguf4zXZe2ihmciumh0LtkkFGobk8jUmF8Zn6Z67wlaexPuMePrXYpLs1wosWl8D4g8qjq8NnhegVbQYEzhG
DLKeD8dGpV6zdiebk8FWknWD4sCM6tv0dI+xVVnZYzlsdS2T4X99KhHYewukwJ1PHKCmoazOcNRJSqoOLo9bqNVO6KQuvvLe3TlTOONmtMDeV5UuuXucc+0
lfKA==']
Digital signature (BASE64): IQWW4qvOS1wczkzecs7Yotqg+XoT1R6jYV9YSmO/vWOzhYxwROXSMYON/WbD+dNLY9jdkVDLqXJN4S2TUhMrHoN1cCP
7brb5RUN0X/Ztn0lUPvsOd2eW8Mjxyp6AghubosS6nX8Psl10tjdB6zpQhbJWKQpQyOO/byLr2BHZFdP1c+EvykPKyKjxFmvEIq2tSlS3yGrugyMkTf+efj
bHCiv6FjSrmG7FrnjhU+UtPMLyBoObRHKMAHS8oYJ05pqREON+392J84JoMFG7SWlCSY0C2aAMSAUYBsAuR3dwD4zVV8FMGeyLNR/9us/Udw5YYeoP1ZLxb3
k5bCPS82E/8ww==
------------------------------------------------------------

 [!] Verificando assinatura ...
 [*] Importando chave pública do arquivo arquivo 'KeyPair.kp' ... Done.
 [*] Decodificando hash recebido e chave AES ...
  |-[$] Hash (recebido): 7ecf105ebf8d517b3968397a093c565e28469559bee88efc11f5f9af0fafdf27
  |-[$] Key: 7YVIHKFORHQSPPCM
  |-[$] Nonce: 0000101010011111110000110000110111011110000111010101011001101011
 [*] Decodificando documento recebido usando AES CTR ...
  |> [████████████] 19936/19936
 [*] Gerando hash da mensagem ...
  |-[$] Hash (calculado): 7ecf105ebf8d517b3968397a093c565e28469559bee88efc11f5f9af0fafdf27
 [*] Comparando hashes ...

>>> Esperado (hash): 7ecf105ebf8d517b3968397a093c565e28469559bee88efc11f5f9af0fafdf27
>>> Recebido (hash): 7ecf105ebf8d517b3968397a093c565e28469559bee88efc11f5f9af0fafdf27

 [$] Documento é válido :)
```

Figure 16. Assinatura de documento (Cliente).

4. Referências

1. Slides de Aula
2. YOUTUBE: OAEP in RSA
3. Encriptação Simétrica vs. Assimétrica
4. Preenchimento de criptografia assimétrica ideal
5. Counter Mode - Applied Cryptography
6. Introdução ao AES - Advanced Encryption Standard
7. Criptografia Assimétrica e Assinatura Digital