Assinatura e Verificação de Arquivos com RSA

Vinícius T M Sugimoto

6 de Dezembro de 2020

1 Introdução

A comunicação e troca de informações sempre foi algo essencial na vida de todos, e com a internet a troca de todo tipo de dados vem aumentando significantemente. Com isso, também vem a preocupação de que suas informações não sejam alteradas ou até mesmo vistas até que cheguem aos seus destinos. Então é preciso uma maneira de garantir a segurança de suas informações enviadas pela internet, onde passam por diversos computadores e por diferentes rotas até o destino. A criptografia moderna tenta apresentar soluções para esses problemas.

A criptografia de arquivos, ou mensagens, é uma maneira de proteger seus dados de serem alterados ou vistos e permitir que quem os receba possa verificar sua integridade. Existem diversos algoritmos que implementam sistemas de cifração e decifração, como a Cifra de Viginère, o AES (American Encryption Standard) e o RSA (Rivest-Shamir-Adleman). Esse último, foco deste trabalho, apesar de poder ser utilizado para cifrar mensagens genéricas, é um algoritmo cuja perfomance deixa a desejar. Entretanto, seu esquema de chaves assimétricas o faz adequado para cifrar chaves de outros esquemas, como do AES, ou gerar assinaturas digitais.

Neste trabalho, o algoritmo RSA é ultilizado com o propósito de gerar assinaturas digitais para arquivos. O RSA é bem conhecido e fácil de ser implementado, com o auxílio de algumas ferramentas para gerar números pseudo-aleatórios e decidir primalidade, por exemplo. Mas sua versão básica, chamada de *textbook-RSA*, tem falhas preocupantes, por ser um algoritmo determinístico. Para implementar o RSA de forma mais segura, precisa-se utilizar uma função de *hash* segura e um algoritmo de *padding*. Os algoritmos SHA-3-512, de *hash* seguro, e OAEP, de *padding*, foram utilizados neste trabalho.

Nas seções seguintes, serão discutidos brevemente detalhes sobre o funcionamento do SHA-3, RSA e OAEP, o funcionamento da assinatura RSA e detalhes sobre a implementação feita para este trabalho dos algoritmos listados anteriormente. Na seção 2 será discutido o funcionamento dos algoritmos SHA-3, RSA e OAEP, na seção 3 será discutido o funcionamento da assinatura RSA e detalhes sobre a implementação.

2 Os Algoritmos

2.1 SHA-3

O SHA-3 (Secure Hash Algorithm) é um algoritmo de *hash* seguro publicado pelo NIST (National Institute of Standards and Technology) em 2015 que substitui seus antecessores, SHA-1 e SHA-2. Este algoritmo utiliza uma construção de esponja, isto é, tem uma fase para "absorver"(processamento) a entrada e uma fase para "espremer"(gerar) a saída. Esse tipo de construção permite utilizar entradas de tamanhos arbitrários para gerar saídas de tamanhos arbitrários. O algoritmo usa uma função Keccak, que realiza uma série de permutações com

operações XOR, AND e NOT, para ser eficiente tanto em hardware como em software, sobre a entrada absorvida e para espremer uma saída.

2.2 RSA e Geração de Chaves

O RSA é um sistema de cifração e decifração de chave assimétrica, isto é, utiliza duas chaves, uma pública e outra privada ou secreta, de modo que se uma chave cifra uma mensagem, somente sua outra chave correspondente possa decifrar. A segurança do RSA se baseia no problema da fatoração do produto de dois números primos grandes, e apesar de que o questionamento sobre a dificuldade de quebrar o RSA ser igual a do problema da fatoração ainda estar em aberto, para tamanhos de chaves suficientemente grandes, não existem métodos conhecidos de quebrar o RSA. Apesar disso, o RSA não é comumente utilizado para cifrar dados diretamente, mas sim chaves de sistemas criptográficos simétricos, como o AES, pois o RSA pode ser considerado devagar, se comparado a outros sistemas.

Uma implementação da geração de chaves do RSA, demonstrada em 1, consiste de gerar dois números primos grandes, comumente chamados p e q, de tamanhos próximos, que são utilizados para gerar um $n = pq$, conhecido como o módulo, depois $\phi = (p-1)(q-1)$ é calculado e um inteiro e é escolhido tal que $1 < e < \phi$ e $MDC(e, \phi) = 1$, conhecido como expoente público, e por fim calcular d tal que $1 < d < \phi$ e $ed \equiv 1 \bmod \phi$, conhecido como expoente secreto. A chave pública é o par de módulo e expoente (n, e) e a chave privada é o par (n, d). Geralmente $e \in \{3, 5, 17, 257, 65537\}$, pois esses valores são primos e têm somente dois bits com valor 1, facilitando a exponenciação modular.

Algorithm 1 Geração de Chaves RSA

function RSA_GenKeys:
  $p, q \leftarrow$ primos grandes pseudoaleatórios de tamanhos parecidos
  $n = pq$
  $\phi = (p-1)(q-1)$
  $e \leftarrow$ inteiro tal que $1 < e < \phi$ e $MDC(e, \phi) = 1$
  $d \leftarrow$ inteiro tal que $1 < d < \phi$ e $ed \equiv 1 \bmod \phi$
  return $((n, e), (n, d))$
end function

Uma chave RSA é comumente associada com um comprimento em bits, o comprimento em bits de n. Valores normalmente desejados para o comprimento de n são 1024, 2048 etc. Este trabalho apresenta uma implementação do RSA com um comprimento de chave de 1024 bits.

A cifração RSA utiliza a chave pública, por exemplo, e computa o texto cifrado c de uma mensagem m com $c = m^e \bmod n$ e a decifração com a chave secreta é $m = c^d \bmod n$. Assim, não há algum componente randômico, isto é, são algoritmos determinísticos, o que os torna vulneráveis a ataques de texto escolhido (não-CPA-seguro). O RSA como descrito acima sem um *padding* não é semanticamente seguro. Na subseção seguinte é apresentado o algoritmo de *padding* OAEP e como pode ser utilizado para resolver o problema do determinismo no RSA.

2.3 OAEP e Cifração e Decifração com RSA

O OAEP, Optimal Asymetric Encryption Padding, é um esquema de *padding* muito utilizado junto ao RSA para resolver seu problema de determinismo. O OAEP usa um par de oráculos randômicos para processar a mesagem antes de cifrá-la, resultando em um esquema CPA-seguro. O OAEP com o RSA serve para adicionar randomicidade ao esquema determinístic o tornando

em um esquema probabilístico. O OAEP também previne que qualquer informação de algum texto cifrado vaze, impedindo a decifração parcial por parte de um adversário.

Apesar do OAEP ser um esquema de *padding*, ele faz mais do que só adicionar um *padding* à mensagem de entrada. Considerando a descrição do RSA dada em 2.2 e os oráculos randômicos G e H, uma implementação do OAEP pode ser dada como no algoritmo 2.

Algorithm 2 Padding com OAEP

function OAEP_ENCODE(m: mensagem):
  $n \leftarrow$ comprimento de chave RSA em bits
  $k_0 \leftarrow$ constante fixada
  $k_1 \leftarrow n - k_0 - |m|$
  $m \leftarrow m\overbrace{00...0}^{k_1}$
  $r \leftarrow$ cadeia de k_0-bits gerados randômicamente
  $X \leftarrow m \oplus G(r)$, com G expandindo r para $n - k_0$ bits
  $Y \leftarrow r \oplus H(X)$, com H expandindo X para k_0 bits
  return $X||Y$, a concatenação de X e Y
end function
function OAEP_DECODE($X||Y$: mensagem codificada):
  $r \leftarrow Y \oplus H(X)$
  return $m00...0 \leftarrow X \oplus G(r)$
end function

Neste trabalho, a constante k_0 foi fixada em 88, representando um comprimento em bits, o que implica em uma mensagem de no máximo 117 bytes. Caso uma mensagem tenha tamanho maior que o máximo o OAEP deve abortar e apresentar uma mensagem de erro. Neste trabalho os oráculos H e G utilizados foram $H = G = SHA\text{-}3$.

Com o OAEP removendo o determinismo da cifração e decifração do RSA, os algoritmos usados para cifrar e decifrar mensagens podem ser descritos como em 3 e 4, respectivamente.

Algorithm 3 Cifração com RSA

function RSA_ENCRYPT(m: mensagem, k: chave RSA):
  $n, e \leftarrow$ módulo de k, expoente de k
  $m \leftarrow OAEP\_Encode(m)$
  return $m^e \bmod n$
end function

Algorithm 4 Decifração com RSA

function RSA_DECRYPT(c: mensagem cifrada, k: chave RSA):
  $n, d \leftarrow$ módulo de k, expoente de k
  $c \leftarrow c^d \bmod n$
  return $OAEP\_Decode(c)$
end function

3 Assinatura RSA e Implementação

Nesta seção será apresentado o uso da ferramenta e como a sua documentação detalhada pode ser encontrada. O trabalho foi desenvolvido na linguagem C e testado em um computador com Ubuntu Linux 20.10. O código fonte da ferramenta está hospedado no repositório em https://github.com/vinicius-toshiyuki/rsa. O executável da ferramenta pode ser gerado a partir do código fonte com auxílio da ferramente `make` como demonstrado no trecho de código 1.

O primeiro passo ao utilizar a ferramenta, chamada a partir de agora `rsa.out`, é gerar um par de chaves, que têm extensões `.pk` e `.sk`, para as chaves pública e secreta, respectivamente. A geração de chaves pode ser feita a partir de um terminal como demonstrado no trecho de código 2.

O usuário pode assinar arquivos tanto com a chave privada quanto com a chave pública, mas só pode verificar a integridade (corretamente) um arquivo com a chave-irmã da chave utilizada para assinar. O arquivo contendo a assinatura digital tem a extensão `.sign`. A assinatura de um arquivo pode ser feita a partir de um terminal como demostrado no trecho de código 3.

Para verificar a integridade de um arquivo, o usuário deve utilizar a chave-irmã da chave utilizada para gerar a assinatura. O programa imprimirá `Valid` ou `Invalid` caso a assinatura seja válida ou inválida, respectivamente. A verificação da validade da assinatura para um arquivo pode ser feita a partir de um terminal como demonstrado no trecho de código 4.

O repositório do código fonte contém a documentação do projeto e informações extras, como dependências do projeto, intruções detalhadas sobre a compilação, informações sobre os comandos suportados e referências utilizadas durante a implementação e parâmetros utilizados na configuração das funções. Instruções sobre como gerar os arquivos de documentação utilizando as ferramentas `make` e `Doxygen` estão demonstradas no trecho de código 5.

```
# Gera o arquivo "rsa.out" no diretório como o executável
make
```

Trecho de código 1: Geração do executável

```
# Gera um par de chaves com o prefixo "key"
./rsa.out -c genkeys -f key
```

Trecho de código 2: Geração de chaves RSA

```
# Gera uma assinatura .sign com prefixo "file" a partir de
# um arquivo "file" usando a chave pública em "key.pk"
./rsa.out -c sign -f file -s file -k key.pk
```

Trecho de código 3: Assinatura de arquivo

```
# Valida a assinatura em "file.sign" para um arquivo "file"
# usando a chave privada em "key.sk"
./rsa.out -c verify -f file -s file.sign -k key.sk
```

Trecho de código 4: Validação da assinatura de um arquivo

```
# Gera a documentação do projeto na pasta docs
make docs
```

Trecho de código 5: Geração dos arquivos de documentação