

Trabalho 2 - Gerador/Verificador de Assinatura

Gustavo Dezan

Felipe Dantas Borges

10 de julho de 2023

Universidade de Brasília
Segurança Computacional - CIC0201
gustavol.dezan@gmail.com felipedbbsb@gmail.com

1 Visão geral

O trabalho foi implementado utilizando a linguagem Python . O código consiste de 4 módulos: “AES.py”, “rsa.py”, “oaep.py” e main.py”. O objetivo era implementar um gerador e verificador de assinaturas RSA em arquivos.

1.1 AES

Neste módulo, está presente a implementação do algoritmo AES, uma cifra de bloco simétrica. A versão implementada do AES utiliza chaves criptográficas de 128 bits e realiza 10 rodadas para criptografar e descriptografar dados em blocos de 128 bits. Cada rodada é composta por quatro operações: AddRoundKey, SubBytes, ShiftRows e MixColumns.

1. AddRoundKey: Combinação dos dados com a chave de criptografia através de uma operação XOR.
2. SubBytes: Substituição não linear de cada byte do bloco de dados usando uma tabela de substituição.
3. ShiftRows: Deslocamento das linhas do bloco de dados para espalhar os bytes.
4. MixColumns: Transformação linear de cada coluna do bloco de dados para aumentar a difusão.

Essas operações são aplicadas em várias rodadas, exceto MixColumns na última rodada. Juntas, essas etapas garantem a segurança e confidencialidade dos dados criptografados pelo AES.

Além disso, nesta implementação, é utilizado o modo de operação chamado Galois Counter Mode (GCM), que é um modo utilizado em criptografia simétrica. O GCM utiliza uma representação especial de um grupo multiplicativo associado a uma curva elíptica. Esse modo de operação combina a autenticação de mensagem com a criptografia, permitindo alcançar tanto confidencialidade quanto integridade dos dados.

1.2 RSA

No algoritmo RSA, é essencial gerar chaves seguras e robustas. Isso envolve a geração de dois números primos grandes, p e q , com um mínimo de 1024 bits cada. A segurança do RSA é baseada na dificuldade de fatorar esses números primos.

Para garantir que os números gerados sejam primos, é comum utilizar o teste de primalidade de Miller-Rabin. Esse teste probabilístico permite determinar se um número é composto ou provavelmente primo.

O teste de Miller-Rabin envolve a escolha aleatória de uma base e a aplicação de um conjunto de testes ao número em questão. Se o número passar em todos os testes, ele é considerado provavelmente primo. Caso contrário, é definitivamente composto.

Ao repetir o teste com várias bases diferentes, aumentamos a confiabilidade do resultado. Quanto mais iterações forem realizadas, menor será a probabilidade de um número composto ser erroneamente identificado como primo.

Após a geração bem-sucedida dos números primos p e q , as chaves pública e privada podem ser calculadas usando esses valores.

1.3 OAEP

O OAEP, ou Preenchimento de Criptografia Assimétrica Ótima, é um padrão de preenchimento utilizado em conjunto com o RSA para aumentar a segurança das operações de criptografia.

O objetivo do OAEP é adicionar aleatoriedade e proteção contra ataques de criptoanálise aos dados antes da criptografia. Ele usa funções hash, como o SHA-3, para realizar o preenchimento.

O processo de preenchimento do OAEP envolve duas etapas principais: preenchimento e aleatorização. Durante o preenchimento, são adicionados bits extras aos dados originais para torná-los mais seguros contra ataques de força bruta e análise de padrões. Em seguida, a aleatorização adiciona um valor aleatório aos dados para garantir que diferentes mensagens criptografadas resultem em saídas diferentes.

O OAEP é amplamente utilizado para fortalecer a segurança do RSA, adicionando uma camada adicional de proteção aos dados antes da criptografia.

No processo de cifração assimétrica RSA, o OAEP é usado em conjunto com o algoritmo RSA para criptografar e descriptografar mensagens.

Para criptografar uma mensagem usando RSA e OAEP, o remetente primeiro aplica o preenchimento OAEP à mensagem original. Em seguida, ele usa a chave pública do destinatário para realizar a criptografia RSA dos dados preenchidos.

Ao receber a mensagem criptografada, o destinatário aplica a operação de descriptografia RSA usando sua chave privada correspondente. Em seguida, ele reverte o processo de preenchimento OAEP para obter a mensagem original.

Esse processo garante que a mensagem só possa ser descriptografada pelo destinatário correto, que possui a chave privada correspondente à chave pública usada na criptografia.

A combinação do RSA com o OAEP fornece uma criptografia assimétrica forte e confiável, adequada para proteger a comunicação e os dados sensíveis.

1.4 MAIN

Neste módulo, estamos simulando uma transmissão segura de dados usando os algoritmos AES e RSA. Vamos descrever a situação simulada:

No início, o remetente gera uma mensagem e as chaves para os algoritmos AES e RSA. Em seguida, a mensagem é criptografada usando o AES. Além disso, a chave usada na criptografia AES e o hash da mensagem são assinados com o RSA.

O remetente envia os seguintes dados para o destinatário: a chave pública, a mensagem cifrada com AES, a chave da criptografia AES assinada com RSA e o hash da mensagem assinado com RSA.

O destinatário recebe os dados enviados e usa o RSA para recuperar a chave de criptografia AES. Em seguida, ele decifra a mensagem usando o AES com a chave recuperada.

Para verificar a integridade da mensagem, o destinatário usa o RSA para recuperar o hash original da mensagem. Ele então calcula um novo hash para a mensagem decifrada e verifica se o hash calculado coincide com o hash recuperado.

Essa simulação demonstra como ocorre uma transmissão segura de dados, onde a mensagem é criptografada com o AES para garantir a confidencialidade, e a integridade é verificada usando a assinatura digital RSA. A combinação desses algoritmos proporciona um ambiente seguro para a comunicação entre o remetente e o destinatário.

A seguir o resultado dos executáveis:

```

-----
Key:                1e997998d28ce6bcbe9af14191212b
-----

-----
MENSAGEM:          Hello, World!e eu gosto mto de sla  e pah#$$~&*()g
-----

-----
MENSAGEM:          61818dbb6cbb176c16880896560fe5cc43d81734f4606e108b44cc6560e845854497160eb216b47c0410266ab3083
-----

-----
MENSAGEM:          Hello, World!e eu gosto mto de sla  e pah#$$~&*()g
-----

```

Figura 1: executável da AES

```

Insira sua mensagem: Hello, World!
Padding adicionado: b'\x00\xcc\x18\xbd\xa9\xd7\xe9\xee\x08yI>\x08\xbe\xdeX\x97I\x86\xa2d*0zf\xa8\x8e\xb6/\x1fz\x85\xae\xa7\xff\xc6\xf8\xbf\x1e\xdfq\xc1GV\xa0a\xdb6b\xf5\x80\xffM\xe4;I\xfa\x82\xd8\nK\x80\xf8CJ\x01Hello, World!'
Public key: (253819281778227677081374155331985260493515683727430187591942700947663892044373715238846741833602251217809996120535369736648992804594054329
3685245895929148647355251340620685459746549239047520782946154671014934730137586382716344876738752325292546669337837483686274223418948724951411143960142
3001378146203634177653160624524504310399694090108180049236684469662098886925982517833327070435835861878263666902408252828379401725463074977315889838447
0340183486428198432139003454082345467343217271316310822469460086648566368796363981989836641876142140587434534424375945358997299992577410761141694416779
85765271327077719192444643, 139338172423140256655277102961118396666655225025512292567634257846888327025407895079242202778805079534232012189410466013743
96634889292064973178693033026293084179812615357996022730110664839983443101500501307945284646250472034017533055831400608076173514382303723043733109436796
55872659263313672638220025987134237)
Private key: (25381928177822767708137415533198526049351568372743018759194270094766389204437371523884674183360225121780999612053536973664899280459405432
9368524589592914864735525134062068545974654923904752078294615467101493473013758638271634487673875232529254666933783748368627422341894872495141114396014
2300137814620363417765316062452450431039969409010818004923668446966209888692598251783332707043583586187826366690240825282837940172546307497731588983844
70340183486428198432139003454082345467343217271316310822469460086648566368796363981989836641876142140587434534424375945358997299992577410761141694416779
85765271327077719192444643, 2313083191206408500685056690989777339364008655275518120042832371266426082165724662747846113305338543662314971880456657079870
5030821149711563067089740835836451869764113105019237738238330840934066757180061706509835077318520973130825321626842073556034026561361253026926608635964
6340670926682039702160677090127520666900636028361103108541135118932008038005092125839550847771797776244126100685540803145006856201696175245998338646
8602459071473409834657000089565943562014692781832302845887694191580043523198440246827808337949581357214346320167873183506618036662317187779870809124079
930869437879144006670687900016915676115605)
Decifrando: b'\x00\xcc\x18\xbd\xa9\xd7\xe9\xee\x08yI>\x08\xbe\xdeX\x97I\x86\xa2d*0zf\xa8\x8e\xb6/\x1fz\x85\xae\xa7\xff\xc6\xf8\xbf\x1e\xdfq\xc1GV\xa0a
\xdb6b\xf5\x80\xffM\xe4;I\xfa\x82\xd8\nK\x80\xf8CJ\x01Hello, World!'
Mensagem original: Hello, World!
Message hash: 1af17a664e3fa8e419b8ba05c2a173169df76162a5a286e0c405b460d478f7ef
Assinatura: b'\x00\x9e\x95\x96\x15\xcc0\x11D\xd9y\x88;\xcFJA\xaaU\xf4\x95\x06\x9c\x81kLe\x1f'\x01n\x03|\xcc\xa7\xff\xc6\xf8\xbf\x1e\xdfq\xc1GV\xa0a\xdb
6b\xf5\x80\xffM\xe4;I\xfa\x82\xd8\nK\x80\xf8CJ\x011af17a664e3fa8e419b8ba05c2a173169df76162a5a286e0c405b460d478f7ef"

```

Figura 2: executável da main