

Relatório: Gerador e Verificador de Assinatura RSA

Igor Bastos de Oliveira, 18/0122177

¹Dep. Ciência da Computação – Universidade de Brasília (UnB)

1. Introdução

O RSA (Rivest-Shamir-Adleman), como um dos pioneiros nas tecnologias assíncronas de criptografia de chave pública, destaca-se como uma ferramenta altamente eficiente na segurança computacional digital [2]. Ampla e frequentemente utilizada para garantir a transmissão segura de dados em diversas aplicações, como transações online e envio de e-mails, o RSA se destaca por sua notável robustez, sendo considerado um dos métodos mais seguros para a criptografia de dados.

A característica essencial do RSA fundamenta-se em sua abordagem assimétrica, onde a chave de encriptação pública é claramente distinta da chave de deciptação privada. Essa assimetria é estabelecida com base na complexidade prática associada à fatorização do produto de dois números primos grandes. Essa complexidade matemática constitui a espinha dorsal da segurança do RSA, uma vez que a fatorização eficiente de grandes números primos é uma tarefa desafiadora e intensiva computacionalmente.[1]

Além disso, o RSA representa uma das primeiras inovações a possibilitar tanto a criptografia quanto a assinatura digital. O algoritmo opera por meio da criação e divulgação de uma chave pública composta por dois números primos significativos, acompanhados de um valor auxiliar. A manutenção da confidencialidade dos números primos subjacentes é crucial para a eficácia do sistema.

Ao empregar a chave pública, qualquer pessoa pode criptografar uma mensagem, mas a decodificação bem-sucedida exige um profundo entendimento dos números primos envolvidos. Esse desafio, conhecido como o "problema RSA", é a essência da resistência do RSA contra tentativas de quebra.[3]

Neste relatório, abordaremos a criação de um programa de criptografia e descryptografia utilizando o RSA, explorando assim as potencialidades e desafios associados a essa renomada técnica de segurança.

2. RSA Implementação

Nesta seção, discutiremos a relevância e a função do código apresentado, que se concentra na implementação do algoritmo RSA (Rivest-Shamir-Adleman) para geração de chaves, criptografia e descryptografia de mensagens.

2.1. RSAKeyGenerator

A classe RSAKeyGenerator é central para a implementação do algoritmo RSA, desempenhando um papel crucial na geração das chaves pública e privada, bem como na facilitação dos processos de criptografia e descryptografia. Abaixo estão detalhes específicos sobre os principais métodos e atributos desta classe:

```
def __init__(self, key_size=1024):  
    self.key_size = key_size
```

```

self.p = self.q = self.n = self.fn = self.e = self.d = None
self.generate_keys()

```

- O método de inicialização define o tamanho padrão da chave como 1024 bits, mas permite a personalização desse tamanho.
- Atributos importantes, como p, q, n, fn, e, e d, são inicializados como None.
- Automaticamente invoca o método generateKeys() para gerar as chaves durante a criação de uma instância da classe.

```

def miller_rabin(self, n, k):
    if n in {2, 3}:
        return True

    if n % 2 == 0:
        return False

    r, s = 0, n - 1
    while s % 2 == 0:
        r += 1
        s //= 2

    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, s, n)

        if x in {1, n - 1}:
            continue

        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False

    return True

```

- Implementação do teste probabilístico de primalidade de Miller-Rabin, crucial para verificar se um número é primo.
- Utiliza parâmetros n e k, representando o número a ser testado e o número de iterações do teste, respectivamente.
- A lógica envolve manipulação eficiente de números primos e aleatoriedade para garantir a precisão do teste.

```

def extended_gcd(self, a, m):
    if math.gcd(a, m) != 1:
        return None

```

```

u1, u2, u3 = 1, 0, a
v1, v2, v3 = 0, 1, m
while v3 != 0:
    q = u3 // v3
    v1, v2, v3, u1, u2, u3 = (
        u1 - q * v1,
        u2 - q * v2,
        u3 - q * v3,
        v1,
        v2,
        v3,
    )
return u1 % m

```

- Implementa o algoritmo de Euclides Estendido para calcular o inverso multiplicativo modular de a módulo m.
- Garante que a e m sejam primos entre si (coprimos), um requisito fundamental para a determinação da chave privada.

```

    def generate_keys(self):
        self.generate_prime_pair()
        self.calculate_n_and_fn()
        self.calculate_public_key()
        self.calculate_private_key()

def generate_prime_pair(self):
    while True:
        try:
            self.p = self.generate_primes()
            self.q = self.generate_primes()
            break
        except RecursionError:
            pass

def calculate_n_and_fn(self):
    self.n = self.p * self.q
    self.fn = (self.p - 1) * (self.q - 1)

def calculate_public_key(self):
    self.e = 0
    while math.gcd(self.fn, self.e) != 1:
        self.e = random.randrange(2, self.fn)

def calculate_private_key(self):
    self.d = self.extended_gcd(self.e, self.fn)

```

- Coordena a geração completa das chaves, invocando métodos específicos para gerar o par de primos, calcular n e fn, bem como determinar as chaves pública e privada.

- Gera um par de primos utilizando o método `generateprimes`, evitando recursões infinitas e garantindo a obtenção de primos distintos.
- Calcula os valores de n e fn com base nos primos gerados, essenciais para a construção das chaves.
- Determina a chave pública (e) garantindo que seja coprima em relação a fn .
- Utiliza o algoritmo de Euclides Estendido (`extendedgcd`) para calcular a chave privada (d).

```

        def rsa_encrypt(self, message, public_key):
            e, n = public_key
            cipher_text = [pow(ord(char), e, n) for char in message]
            return cipher_text

    def rsa_decrypt(self, cipher_text, private_key):
        d, n = private_key
        decrypted_text = ''.join([chr(pow(char, d, n)) for char in cipher_text])
        return decrypted_text

```

O RSA encrypt é responsável por criptografar uma mensagem utilizando a chave pública do algoritmo RSA. A lógica por trás dessa criptografia é baseada no uso da exponenciação modular para garantir eficiência computacional. pontualmente, ele faz:

- O método descompacta os valores da chave pública (e e n). Itera sobre cada caractere da mensagem, aplicando a função `ord(char)` para obter o valor numérico correspondente ao caractere na tabela ASCII.
- Em seguida, utiliza a função `pow` para realizar a exponenciação modular, onde a base é o valor numérico do caractere, o expoente é e (componente da chave pública), e o módulo é n (outro componente da chave pública).
- O resultado dessa operação é uma lista de valores criptografados, que representa a mensagem criptografada..
- Determina a chave pública (e) garantindo que seja coprima em relação a fn .
- Utiliza o algoritmo de Euclides Estendido (`extendedgcd`) para calcular a chave privada (d).

Ja o RSA decrypt tem como função descriptografar uma mensagem criptografada utilizando a chave privada do algoritmo RSA. Da mesma forma que a etapa de criptografia, utiliza a exponenciação modular para garantir eficiência durante o processo.

- O método descompacta os valores da chave privada (d e n).
- Itera sobre cada valor na lista `ciphertext`. (outro componente da chave pública).
- Aplica a função `pow` para realizar a exponenciação modular inversa, onde a base é o valor criptografado, o expoente é d (componente da chave privada), e o módulo é n (outro componente da chave privada).
- O resultado dessa operação é uma lista de valores numéricos, que são convertidos de volta para caracteres usando a função `chr`. A concatenação desses caracteres resulta na mensagem original descriptografada.

Essas são algumas classes importantes para a geração de chave publica e privada, alem delas existem classes auxiliares para geração de numeros aleatorios, todas são fundamentais para a criptografia e descriptografia RSA

3. Assinatura Digital com RSA e Esquema OAEP

esta seção, exploraremos a implementação de assinaturas digitais utilizando o algoritmo RSA em conjunto com o esquema OAEP (Optimal Asymmetric Encryption Padding). Essa abordagem visa garantir a autenticidade e integridade das mensagens, fornecendo uma camada adicional de segurança. Abaixo estão os principais pontos relacionados à classe `RSASignatureGenerator`:

3.1. Módulos de importação

Para o desenvolvimento da assinatura digital, foi necessário a utilização de alguns módulos necessários para auxiliar na construção do algoritmo, entre eles temos:

- `hashlib`: utilizado para implementar funções hash seguras, sendo fundamental para calcular o hash SHA3-224 da mensagem. O SHA3-224 é escolhido devido à sua resistência comprovada a colisões e propriedades criptográficas sólidas
- `Os`: acessar funcionalidades relacionadas ao sistema operacional. Em particular, a função `os.urandom()` é usada para gerar bytes aleatórios, essenciais para o esquema OAEP, onde aleatoriedade é crucial para a
- `base64`: É utilizado para codificar e decodificar dados em formato base64. Aqui, ele desempenha um papel importante na representação compacta e segura dos hashes e das assinaturas, facilitando a integração com sistemas que esperam dados em formato de texto.
- `Math`: operações matemáticas específicas.
- `RSAPKeyGenerator`: Realça a necessidade de um sistema RSA completamente funcional para operações de criptografia e descriptografia

3.2. Métodos da Classe

```
def sha3_224(self, m):
    sha3 = hashlib.sha3_224()
    sha3.update(m)
    return sha3.digest()

def mgf1(self, seed, mlen):
    t = b''
    hlen = 28

    for c in range(ceil(mlen / hlen)):
        c_bytes = c.to_bytes(4, byteorder='big')
        t += self.sha3_224(seed + c_bytes)

    return t[:mlen]
```

- `sha3224(self, m)`: Este método recebe uma mensagem `m` e calcula o hash SHA3-224 usando o módulo `hashlib`. O resultado é um resumo criptográfico de 224 bits da mensagem. Ele é essencial para a obtenção do hash da mensagem, que é uma etapa crítica no processo de assinatura digital.
- `mgf1(self, seed, mlen)`: Implementa a função de geração de máscara (MGF1) Parte integrante do processo de codificação OAEP, fornecendo máscaras seguras para o padding.

```

    def codifica_oaep(self, m, k, label=b''):
        mlen = len(m)
        lhash = self.sha3_224(label)
        hlen = len(lhash)

        ps_len = k - mlen - 2 * hlen - 2
        ps = b'\x00' * ps_len

        db = lhash + ps + b'\x01' + m

        seed = os.urandom(hlen)
        db_mask = self.mgf1(seed, k - hlen - 1)
        masked_db = self.xor_bloco(db, db_mask)

        seed_mask = self.mgf1(masked_db, hlen)
        masked_seed = self.xor_bloco(seed, seed_mask)

        return b'\x00' + masked_seed + masked_db

```

- `codificaoaep(self, m, k, label=b'')`: Realiza a codificação OAEP na mensagem `m` com padding e máscaras pseudoaleatórias para melhorar a segurança durante a cifragem. Prepara a mensagem para cifragem, incorporando o esquema OAEP.

```

    def xor_bloco(self, a, b):
        return bytes(x ^ y for x, y in zip(a, b))
        return b'\x00' + masked_seed + masked_db

```

- `xorBloco(self, a, b)(self, m, k, label=b'')`: Realiza a operação de XOR elemento a elemento entre dois blocos de bytes `a` e `b`. Utilizado internamente para combinar blocos de dados de maneira segura, sendo uma operação fundamental na implementação de esquemas criptográficos.

```

    def cifra(self, mensagem, chave_publica):
        e, n = chave_publica
        return pow(mensagem, e, n)

```

- `cifra`: Realiza a cifragem de uma mensagem numérica usando a chave pública do RSA.

```

    def assina_mensagem(self, mensagem):
        mensagem_bytes = mensagem.encode('utf-8')
        resultado_sha = self.sha3_224(mensagem_bytes)

        mensagem_hash = base64.encodebytes(resultado_sha).strip()
        k = (self.public_key[1].bit_length() + 7) // 8

        cifrado_oaep = self.codifica_oaep(mensagem_hash, k, self.label)
        c = self.cifra_raw(cifrado_oaep, self.private_key)

        return base64.encodebytes(c).strip()

```

- ### 3.3. Testes

```

C:\Users\lgilg\PycharmProjects\lisa\lisaProject\lisa\Scripts\python.exe C:\Users\lgilg\PycharmProjects\lisa\lisaProject\lisa.py
#
# ----- Run: Principal ----- #
1. Correção de chaves RSA
2. Assinatura RSA
3. Sdip
4. Escrito em bytes C1, 1, de 31:

```

```

1 #! /usr/bin/perl
2 #
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 #
31 #
32 #
33 #
34 #
35 #
36 #
37 #
38 #
39 #
40 #
41 #
42 #
43 #
44 #
45 #
46 #
47 #
48 #
49 #
50 #
51 #
52 #
53 #
54 #
55 #
56 #
57 #
58 #
59 #
60 #
61 #
62 #
63 #
64 #
65 #
66 #
67 #
68 #
69 #
70 #
71 #
72 #
73 #
74 #
75 #
76 #
77 #
78 #
79 #
80 #
81 #
82 #
83 #
84 #
85 #
86 #
87 #
88 #
89 #
90 #
91 #
92 #
93 #
94 #
95 #
96 #
97 #
98 #
99 #
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
8
```

[illegible]

Ao apertar a opção 2, o programa vai pedir ao usuario uma mensagem para o processo de criptografia e assinatura digital

```

1 #
2 #
3 #
4 #
5 #
6 #
7 #
8 #
9 #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 #
31 #
32 #
33 #
34 #
35 #
36 #
37 #
38 #
39 #
40 #
41 #
42 #
43 #
44 #
45 #
46 #
47 #
48 #
49 #
50 #
51 #
52 #
53 #
54 #
55 #
56 #
57 #
58 #
59 #
60 #
61 #
62 #
63 #
64 #
65 #
66 #
67 #
68 #
69 #
70 #
71 #
72 #
73 #
74 #
75 #
76 #
77 #
78 #
79 #
80 #
81 #
82 #
83 #
84 #
85 #
86 #
87 #
88 #
89 #
90 #
91 #
92 #
93 #
94 #
95 #
96 #
97 #
98 #
99 #
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
979 #
980 #
981 #
982 #
983 #
984 #
985 #
986 #
987 #
988 #
989 #
990 #
991 #
992 #
993 #
994 #
995 #
996 #
997 #
998 #
999 #
1000 #

```

Figure 4. Opção de assinatura digital

O programa vai ficar em loop ate o usuario apertar o 3 e finalizar.

4. Conclusão

O presente trabalho abordou detalhadamente a implementação de assinaturas digitais utilizando o algoritmo RSA em conjunto com o esquema OAEP (Optimal Asymmetric Encryption Padding). Ao longo da exploração das classes RSAKeyGenerator e RSASignatureGenerator, foi possível compreender os processos de geração de chaves, criptografia e descryptografia no contexto do RSA, bem como a aplicação do esquema OAEP para garantir a segurança e integridade das assinaturas digitais.

A classe RSAKeyGenerator destacou-se como uma ferramenta robusta para a criação de chaves pública e privada, fundamentais para a implementação segura do algoritmo RSA. A escolha criteriosa de métodos, como o teste de primalidade de Miller-Rabin e o algoritmo de Euclides Estendido, demonstrou um compromisso claro com a segurança matemática subjacente ao RSA.

A classe RSASignatureGenerator ampliou esse cenário ao incorporar o esquema OAEP, proporcionando uma camada adicional de segurança na assinatura digital de mensagens. A utilização de funções hash seguras, geração de máscaras pseudoaleatórias e operações XOR contribuíram para a robustez do esquema OAEP, resguardando contra potenciais vulnerabilidades.

Em conclusão, a implementação apresentada destaca-se não apenas pela aplicação efetiva do algoritmo RSA e do esquema OAEP. Este trabalho proporciona uma base sólida para compreensão e implementação de sistemas de segurança digital, contribuindo significativamente para o entendimento do papel do RSA na segurança da informação.

References

- [1] BLAKLEY, G. R., AND BOROSH, I. Rivest-shamir-adleman public key cryptosystems do not always conceal messages. *Computers & mathematics with applications* 5, 3 (1979), 169–178.
- [2] KUMAR, M. Advanced rsa cryptographic algorithm for improving data security. In *Cyber Security: Proceedings of CSI 2015* (2018), Springer, pp. 11–15.
- [3] SIHOTANG, H. T., EFENDI, S., ZAMZAMI, E. M., AND MAWENGKANG, H. Design and implementation of rivest shamir adleman's (rsa) cryptography algorithm in text file data security. In *Journal of Physics: Conference Series* (2020), vol. 1641, IOP Publishing, p. 012042.