TRABALHO DE IMPLEMENTAÇÃO 2: GERADOR/VERIFICADOR DE ASSINATURAS RSA

Eduardo Marques – 211021004
Yan Tavares – 202014323

UnB – CIC0201 – Segurança Computacional
2025/1 – Profª Priscila Solis

1 Introdução

A assinatura digital RSA-2048 é um pilar da segurança moderna, garantindo autenticidade e integridade de mensagens. Este trabalho apresenta uma implementação completa em C: geração de chaves, assinatura com padding OAEP e hash SHA3-256, e verificação rigorosa. Utiliza-se a biblioteca GMP para aritmética de múltipla precisão e implementam-se do zero as primitivas criptográficas.

1.1 Repositório

O código com a implementação completa pode ser encontrado em `https://github.com/yantavares/segcomp`.

2 Fundamentação Teórica

2.1 Algoritmo RSA

O RSA explora a dificuldade de fatorar $n = pq$. Define-se:

$$\varphi(n) = (p-1)(q-1), \quad \gcd(e, \varphi(n)) = 1, \quad d \equiv e^{-1} \pmod{\varphi(n)}.$$

A chave pública é (n, e); a privada, (n, d). A segurança baseia-se na dificuldade da fatoração e na resistência da aritmética modular.

O valor do expoente público e foi fixado em 65537, um primo amplamente adotado por oferecer boa eficiência computacional e segurança adequada. Valores muito pequenos, como $e = 3$, são vulneráveis a ataques específicos e devem ser evitados. A implementação garante que $\gcd(e, \varphi(n)) = 1$, e é o valor padrão implementado em bibliotecas padrão coo *OpenSSL*.

2.2 Assinatura Digital

Invertendo o uso tradicional da criptografia:

$$S = \text{Pad}(H(M))^d \bmod n, \quad \text{Pad}(H(M)) \stackrel{?}{=} S^e \bmod n.$$

O hash da mensagem $H(M)$ é expandido via OAEP antes da exponenciação. A verificação recupera o valor hasheado e o compara com o calculado localmente.

2.3 Padding RSA–OAEP

OAEP (RFC 3447) protege contra ataques de texto escolhido. O processo é:

1. Gera uma *seed* aleatória de tamanho h_{Len}.
2. Constrói $DB = \text{lHash} \| PS \| 0x01 \| M$.
3. Calcula dbMask = MGF1(seed, $|DB|$), seedMask = MGF1(dbMask, h_{Len}).
4. Obtém maskedDB = $DB \oplus$ dbMask, maskedSeed = seed $\oplus$ seedMask.
5. Saída final: $0x00 \|$maskedSeed$\|$maskedDB.

Listing 1: MGF1 (SHA3-256) - Pseudocódigo

```
function MGF1(seed, maskLen):
  T = ""
  for counter in 0 to ceil(maskLen/hLen)-1:
    C = I2OSP(counter, 4)
    T += SHA3-256(seed || C)
  return T[0:maskLen]
```

2.4 Hash SHA3-256

Baseado em Keccak-f[1600] com estado $5 \times 5 \times 64$ bits e 24 rodadas. Etapas principais:

- **Absorção:** entrada em blocos de 1088 bits.
- **Padding:** esquema $10 \ldots 1$.
- **Permutação:** 24 rodadas com as funções $\Theta, \rho, \pi, \chi, \iota$.
- **Extração:** coleta de 256 bits do estado final.

Listing 2: Keccak-f[1600

```
for round in 0..23:
  Theta(state)
  Rho(state)
  Pi(state)
  Chi(state)
  Iota(state, RC[round])
```

3 Implementação

3.1 Estrutura Geral

O arquivo `main.c` oferece um menu com as opções:

- Geração de chaves RSA-2048.
- Assinatura (SHA3-256 + OAEP).
- Verificação de assinaturas.
- Extração do conteúdo de arquivos assinados.

3.2 Geração de Primos

A função `generate_prime` utiliza o teste de Miller–Rabin com 40 iterações (erro $\approx 2^{-80}$):

Listing 3: Teste de Miller–Rabin

```
int miller_rabin(mpz_t n, int k):
  write n-1 = d * 2^r
  repeat k vezes:
    a = random(2, n-2)
    x = powmod(a, d, n)
    if x == 1 or x == n-1: continue
    repeat r-1 vezes:
      x = powmod(x, 2, n)
      if x == n-1: break
    if x != n-1: return composite
  return probably prime
```

3.3 Formato de Arquivo Assinado

```
-----BEGIN SIGNED MESSAGE-----
<Base64(Mensagem)>
-----BEGIN SIGNATURE-----
<Base64(Assinatura)>
-----END SIGNATURE-----
```

4 Testes e Validação

Tamanho do arquivo	Geração de chave (s)	Assinatura (s)
1 KB	2.1	0.05
1 MB	3.8	0.20
10 MB	9.5	1.8

Tabela 1: Desempenho médio (Ubuntu 20.04, quad-core 2.5 GHz)

Os testes em ambientes Linux e macOS confirmam:

- Assinatura válida é verificada com sucesso.
- Qualquer modificação no conteúdo invalida a assinatura.
- Padding incorreto é detectado.

5 Compilação e Uso

Requisitos: GCC 7+, GMP 6+.

Compilação:

```
gcc -o rsa_signature main.c -lgmp -lm
```

Execução:

```
./rsa_signature
```

6 Considerações de Segurança

6.1 Força Criptográfica

Chaves RSA-2048 oferecem aproximadamente 112 bits de segurança, conforme recomendações do NIST até 2030.

6.2 Aleatoriedade

Utiliza-se `/dev/urandom` com fallback para `srand(time(NULL))`; para produção, recomenda-se uso de HSM ou gerador CSPRNG.

7 Conclusão

Foi implementado um sistema completo de geração e verificação de assinaturas RSA, abordando conceitos essenciais de hashing, padding e aritmética modular. Como trabalhos futuros, propõem-se extensões com curvas elípticas, integração com PKCS#11 e marcação temporal.

Referências

[1] Rivest, R., Shamir, A., Adleman, L. (1978). *Commun. ACM*, 21(2):120–126.

[2] Jonsson, J., Kaliski, B. (2003). RFC 3447: RSA Cryptography Specifications v2.1.

[3] NIST (2015). FIPS 202: SHA-3 Standard.

[4] Menezes, A., Van Oorschot, P., Vanstone, S. (1996). *Handbook of Applied Cryptography*.

[5] NIST (2020). SP 800-57 Part 1 Rev. 5.

[6] GNU MP (2020). The GMP Library. `https://gmplib.org/`