Projeto 3 - RSA

Cláudio Barros, 190097591

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0201 - Segurança Computacional

190097591@aluno.br

Abstract. *This article explores the implementation of the RSA encryption algorithm augmented with the Optimal Asymmetric Encryption Padding (OAEP) scheme. The RSA-OAEP combination enhances the security of asymmetric encryption by addressing vulnerabilities and improving message randomness.*

Resumo. *Este artigo explora a implementação do algoritmo de criptografia RSA aprimorado com o esquema de preenchimento Optimal Asymmetric Encryption Padding (OAEP). A combinação RSA-OAEP aprimora a segurança da criptografia assimétrica ao abordar vulnerabilidades e melhorar a aleatoriedade das mensagens.*

1. Introdução

O RSA (Rivest-Shamir-Adleman) é um dos algoritmos de criptografia assimétrica mais amplamente utilizados. A criptografia assimétrica envolve o uso de um par de chaves: uma chave pública e uma chave privada. A segurança do RSA se baseia na dificuldade de fatorar números inteiros grandes em seus primos componentes.

Aqui está uma explicação básica do funcionamento do sistema RSA:

Geração de Chaves:

- **Escolha de Números Primos:** Escolha dois números primos grandes, p e q.
- **Cálculo do Módulo:** Calcule $n = p \times q$. O produto n é usado como o módulo para ambas as chaves.
- **Função Totiente de Euler:** Calcule a função totiente de Euler, $\phi(n) = (p-1) \times (q-1)$, ou a função de Charmichael, $\lambda(n) = MMC(p-1, q-1)$
- **Escolha da Chave Pública:** Escolha um número e tal que $1 < e < \phi(n)$ e e seja coprimo com $\phi(n)$, ou seja, não tenha fatores em comum com $\phi(n)$ além de 1.
- **Cálculo da Chave Privada:** Calcule d, o inverso multiplicativo modular de e módulo $\phi(n)$. Isso significa que $(d \times e) \bmod \phi(n) = 1$.

Chave Pública e Privada:

- A chave pública consiste em (n, e) e é distribuída livremente.
- A chave privada consiste em (n, d) e deve ser mantida em segredo.

Criptografia:

- A mensagem que precisa ser criptografada é representada como um número M tal que $0 \leq M < n$.
- A mensagem criptografada C é calculada como $C \equiv M^e \bmod n$.

Descriptografia:

- A mensagem original M pode ser recuperada a partir da mensagem criptografada C usando a chave privada: $M \equiv C^d \bmod n$.

O RSA é seguro porque a fatoração do produto de dois números primos grandes é uma tarefa computacionalmente intensiva, especialmente à medida que o tamanho dos números primos aumenta. A segurança do RSA baseia-se na dificuldade de fatorar o produto de dois primos grandes, mesmo com os avanços na computação. Portanto, a força do RSA está na dificuldade de resolver o ”problema do logaritmo discreto”em aritmética modular.

O RSA como descrito acima pode ter sua segurança bastante reforçada ao utilizar um esquema de padding para aumentar a proteção contra determinados ataques e aumentando a aleatoriedade do sistema. Um desses esquemas é conhecido como OAEP (Optimal Asymmetric Encryption Padding) que transforma o RSA tradicional de um esquema determinístico de criptografia para um esquema probabilístico.

O OAEP é uma forma de rede de Feistel que utiliza um par de oráculos aleatórios G e H para processar a mensagem antes de submetê-la ao RSA. Quando o OAEP é combinado como uma função armadilha (também conhecida como função arapuca ou função alçapão), resulta em um modelo de oráculo aleatório que é seguro contra ataques de texto cifrado escolhido (IND-CPA). Uma descrição completa e detalhada do algoritmo pode ser encontrada neste link [Moriarty et al. 2016].

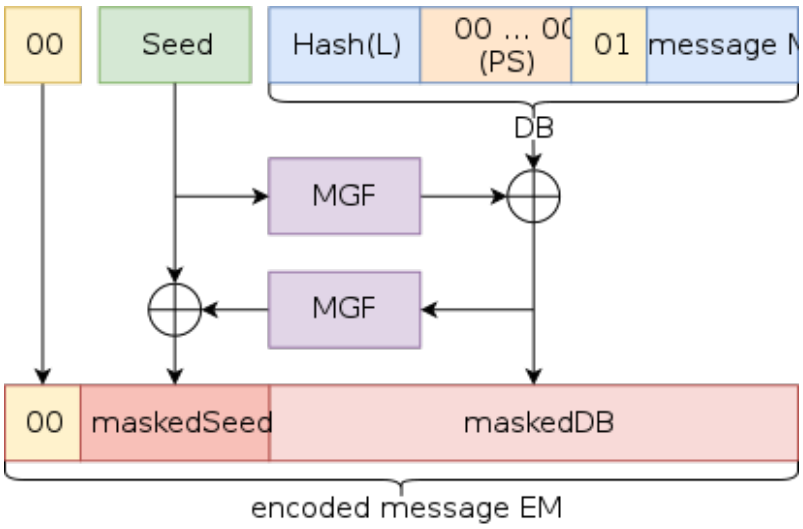

Figura 1. Esquema OAEP de acordo com a RFC 8017 [Moriarty et al. 2016]

2. Implementação

A implementação do RSA foi feita utilizando C++, juntamente com a biblioteca GNU MP para lidar com números muito grandes e aritmética modular. A função de Hash utilizada no esquema OAEP foi SHA-1 em sua implementação da OpenSSL. As seções a seguir mostram as principais funções utilizas em cada etapa descrita anteriormente. O repositório do GitHub com o código completo pode ser acessado aqui.

2.1. Geração de Chaves

```
void generateKeys(mpz_class& privateKey, mpz_class&
 ↪  publicKey, mpz_class& exponent)
{
    std::chrono::time_point<std::chrono::system_clock>
 ↪  now = std::chrono::system_clock::now();
    std::time_t seedValue =
 ↪  std::chrono::system_clock::to_time_t(now);

    gmp_randclass randclass(gmp_randinit_default);
    randclass.seed(seedValue);

    mpz_class p(0);
    mpz_class q(0);
    mpz_class tmp(0);

    mpz_class range(0);
    mpz_ui_pow_ui(range.get_mpz_t(), 2, 2048);

    //get p with at least 1024 bits
    tmp = randclass.get_z_range(range);
    while(mpz_sizeinbase(tmp.get_mpz_t(), 2) < 1024)
    {
        tmp = randclass.get_z_range(range);
    }
    mpz_nextprime(p.get_mpz_t(), tmp.get_mpz_t());

    //get q with at least 1024 bits
    tmp = randclass.get_z_range(range);
    while(mpz_sizeinbase(tmp.get_mpz_t(), 2) < 1024)
    {
        tmp = randclass.get_z_range(range);
    }
    mpz_nextprime(q.get_mpz_t(), tmp.get_mpz_t());

    mpz_class n = p * q;

    //get Charmichael's totient function
    mpz_class lambdaP = p - 1;
    mpz_class lambdaQ = q - 1;
    mpz_class lambdaN = lcm(lambdaP, lambdaQ);

    mpz_class e(65537u);
    mpz_class d(0);
```

```
    mpz_invert(d.get_mpz_t(), e.get_mpz_t(),
 ↪  lambdaN.get_mpz_t());

    publicKey = n;
    exponent = e;
    privateKey = d;
}
```

2.2. RSA Pleno

```
int RSA_Enc(mpz_class& n, mpz_class& e, mpz_class& m,
 ↪  mpz_class& c)
{
    int cmp1 = cmp(m, (n-1));
    int cmp2 = cmp(m, 0);
    if(cmp1 > 0 || cmp2 < 0) //if m < 0 or m > n -1
    {
        std::cout << "message representative out of
 ↪  range\n";
        return 0;
    }

    mpz_powm_sec(c.get_mpz_t(), m.get_mpz_t(),
 ↪  e.get_mpz_t(), n.get_mpz_t());
    return 1;
}

int RSA_Dec(mpz_class& n, mpz_class& d, mpz_class& c,
 ↪  mpz_class& m)
{
    int cmp1 = cmp(c, (n-1));
    int cmp2 = cmp(c, 0);
    if(cmp1 > 0 || cmp2 < 0) //if c < 0 or c > n -1
    {
        std::cout << "ciphertext representative out of
 ↪  range\n";
        return 0;
    }

    mpz_powm_sec(m.get_mpz_t(), c.get_mpz_t(),
 ↪  d.get_mpz_t(), n.get_mpz_t());
    return 1;
}
```

2.3. OAEP

```
int MGF1(vec_u8& output, vec_u8 &seed, uint64_t maskLen)
{
    uint64_t maxLen = ((uint64_t)HASH_LEN) << 32; //2^32 *
 ↪  hLen
    if(maskLen > maxLen)
        return 0;

    vec_u8 tmp;
    for(uint32_t i = 0; i < maskLen; i++)
    {
        vec_u8  i_bytes = u32_to_bytes(i);
        vec_u8 s = concat_bytes(seed, i_bytes);
        vec_u8 hashOut(HASH_LEN, 0);
        SHA1(&s[0], s.size(), &hashOut[0]);

        tmp = concat_bytes(tmp, hashOut);
    }

    output.resize(0);
    output.insert(output.end(), tmp.begin(), tmp.begin() +
 ↪  maskLen);
    return 1;
}

int OAEP_Enc(vec_u8& EM, vec_u8& M, vec_u8& P,  uint64_t
 ↪  len)
{
    if(P.size() > HASH_MAX_INPUT)
    {
        std::cout << "[OAEP_Enc] parameter string too
 ↪  long\n";
        return 0;
    }
    if(M.size() > (len - 2*HASH_LEN - 1))
    {
        std::cout << "[OAEP_Enc] message too long\n";
        return 0;
    }

    int64_t psLen = len - M.size() - 2*HASH_LEN - 1;
    vec_u8 PS((psLen > 0 ? psLen : 0), 0);

    vec_u8 pHash(HASH_LEN, 0);
    SHA1(&P[0], P.size(), &pHash[0]);
```

```
    vec_u8 DB = concat_bytes(pHash, PS);
    DB.push_back(0x01);
    DB = concat_bytes(DB, M);

    gmp_randclass randclass(gmp_randinit_default);

    mpz_class seed = randclass.get_z_bits(HASH_LEN*8);

    vec_u8 seed_bytes(20, 0);
    mpz_export(&seed_bytes[0], 0, 1,
↪   sizeof(seed_bytes[0]), 0, 0, seed.get_mpz_t());

    vec_u8 dbMask(0);

    if(!MGF1(dbMask, seed_bytes, len - HASH_LEN))
        return 0;

    mpz_class maskedDB_mpz(0);
    mpz_class DB_mpz = bytes_to_mpz(DB);
    mpz_class dbMask_mpz = bytes_to_mpz(dbMask);

    mpz_xor(maskedDB_mpz.get_mpz_t(), DB_mpz.get_mpz_t(),
↪   dbMask_mpz.get_mpz_t());

    vec_u8 maskedDB = mpz_to_bytes(maskedDB_mpz);

    vec_u8 seedMask(0);
    if(!MGF1(seedMask, maskedDB, HASH_LEN))
        return 0;

    mpz_class maskedSeed_mpz(0);
    mpz_class seedMask_mpz = bytes_to_mpz(seedMask);
    mpz_xor(maskedSeed_mpz.get_mpz_t(), seed.get_mpz_t(),
↪   seedMask_mpz.get_mpz_t());

    vec_u8 maskedSeed = mpz_to_bytes(maskedSeed_mpz);

    EM = concat_bytes(maskedSeed, maskedDB);

    return 1;
}

int OAEP_Dec(vec_u8& M, vec_u8& EM, vec_u8& P)
{
    if(P.size() > HASH_MAX_INPUT)
```

```
    {
        std::cout << "[OAEP_Dec] decoding error: parameter
↪   length do not match hash length in bytes\n";
        return 0;
    }
    if(EM.size() < (2*HASH_LEN + 1))
    {
        std::cout << "[OAEP_Dec] EM length too short\n";
        return 0;
    }

    vec_u8 maskedSeed = vec_u8(EM.begin(), EM.begin() +
↪   HASH_LEN);
    vec_u8 maskedDB = vec_u8(EM.begin()+HASH_LEN,
↪   EM.end());
    vec_u8 seedMask(0);
    MGF1(seedMask, maskedDB, HASH_LEN);

    mpz_class seed_mpz(0);
    mpz_xor(seed_mpz.get_mpz_t(),
            bytes_to_mpz(maskedSeed).get_mpz_t(),
            bytes_to_mpz(seedMask).get_mpz_t());
    vec_u8 seed = mpz_to_bytes(seed_mpz);

    vec_u8 dbMask(0);
    MGF1(dbMask, seed, (EM.size() - HASH_LEN));

    mpz_class DB_mpz(0);
    mpz_xor(DB_mpz.get_mpz_t(),
            bytes_to_mpz(maskedDB).get_mpz_t(),
            bytes_to_mpz(dbMask).get_mpz_t());

    vec_u8 DB = mpz_to_bytes(DB_mpz);

    vec_u8 pHash(HASH_LEN, 0);
    SHA1(&P[0], P.size(), &pHash[0]);

    vec_u8 pHashDB = vec_u8(DB.begin(),
↪   DB.begin()+HASH_LEN);

    if(comp_bytes(pHash, pHashDB) == false)
    {
        std::cout << "[OAEP_Dec] decoding error: hash
↪   values do not match\n";
        return 0;
    }
```

```
    vec_u8 PS(0);
    bool separatorFound = false;
    size_t MBegin = 0;
    for(size_t i = HASH_LEN; i < DB.size(); i++)
    {
        if(DB[i] == 0)
            continue;
        if(DB[i] == 1)
        {
            if(i != DB.size() - 1)
            {
                separatorFound = true;
                MBegin = i + 1;
            }
        }
    }

    if(separatorFound)
    {
        M = vec_u8(DB.begin() + MBegin, DB.end());
        return 1;
    }
    else
    {
        std::cout << "[OAEP_Dec] decoding error: 0x01
 ↪  separator not found\n";
        return 0;
    }
}
```

2.4. RSA-OAEP

```
int RSA_OAEP_Enc(mpz_class& n, mpz_class& e, vec_u8& M,
 ↪  vec_u8& P, vec_u8& C)
{
    vec_u8 EM(0);

    size_t modlen = mpz_size_in_bytes(n);

    if(!OAEP_Enc(EM, M, P, modlen-1))
        return 0;

    mpz_class M_mpz = bytes_to_mpz(EM);
    mpz_class C_mpz(0);
```

```
    if(!RSA_Enc(n, e, M_mpz, C_mpz))
        return 0;

    C.clear();
    C = mpz_to_bytes(C_mpz);
    return 1;
}

int RSA_OAEP_Dec(mpz_class& n, mpz_class& d, vec_u8& C,
 ↪  vec_u8& P, vec_u8& M)
{
    size_t modlen = mpz_size_in_bytes(n);

    if(C.size() != modlen)
    {
        std::cout << "[RSA_OAEP_Dec] ciphertext length in
 ↪  bytes does not match modulus length in bytes\n";
        return 0;
    }

    mpz_class C_mpz = bytes_to_mpz(C);
    mpz_class EM_mpz(0);
    if(!RSA_Dec(n, d, C_mpz, EM_mpz))
    {
        std::cout << "[RSA_OAEP_Dec] decryption error\n";
        return 0;
    }

    vec_u8 EM = mpz_to_bytes(EM_mpz);
    if(!OAEP_Dec(M, EM, P))
    {
        std::cout << "[RSA_OAEP_Dec] decryption error\n";
        return 0;
    }

    return 1;
}
```

Referências

[Moriarty et al. 2016] Moriarty, K., Kaliski, B., Jonsson, J., and Rusch, A. (2016). PKCS #1: RSA Cryptography Specifications Version 2.2. Internet-Draft draft-moriarty-pkcs1-03, Internet Engineering Task Force. Work in Progress.