Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/4/20230353

Application of RSA algorithm in e-passport with python3 implementation

Yiyang Liu

Cheshire Academy, Cheshire, CT 06410

Yiyang.liu@cheshireacademy.org

Abstract. The e-passport is a bold approach which companied with Biometric Identification and Radio Frequency Identification. By employing the RSA encryption algorithm, e-passport have become effective for protecting the security during the transmission of information. This paper introduces the application of the RSA encryption algorithm in the e-passport, the possible security risks and vulnerabilities of active authentication, and the implementation of RSA encryption and decryption. The processing of RSA encryption and decryption is simulated by using Python, and the advantages and disadvantages of using this implementation in an e-passport are discussed. Because this paper uses a relatively basic RSA algorithm logic, there are certain security issues when applying it to an actual e-passport.

Keywords: RSA, Active Authentication, Python Simulation, Security, E-passport.

1. Introduction

The E-passport is a passport with an embedded smart card with the use of Radio Frequency Identification and Biometric Identification, to contain the personal data of the passport holder. Many countries have adopted e-passports to verify the identity of the passport holder and to make travelling convenient. In 2019, more than 150 different countries, including China and the United States, are already using e-passports[1]. However, the security and integrity of e-passports is critical. The International Civil Aviation Organization has created several sets of e-passport standards, RSA is one of the recommended algorithms used for active authentication protocols[2]. According to ICAO standards, face recognition is a biometric technology available worldwide that can be used for e-passport identification. Thus, the e-passport will contain a digital photo image of the holder's face, fingerprint and iris data.

ICAO specification recommend the use of a security feature called "Active Authentication". Active authentication is derived from the RSA algorithm. It guarantees that an e-passport has a private key as proof of its authenticity. The corresponding public key is stored as part of the signature data on the passport. Its main purpose is to prevent the passport from being cloned [3].

The discussion on e-passport security should include the security of the e-passport distribution, management and validation systems. However, this paper discusses only the basic security mechanisms described in the ICAO specification: Active Authentication and the simulation of RSA implementation in Python, and discusses whether these security mechanisms achieve their original goals, whether the RSA algorithm achieves its goals, and presents the corresponding problems and remaining issues.

© 2023 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/4/20230353

2. RSA Algorithm

RSA is a simple but efficient encryption algorithm with a private key and a public key, the private key being stored in an electronic passport. The RSA algorithm is summarized in three main steps[4].

2.1. Key generation

(1) Choose two large prime numbers (p and q).

(2) Calculate $n = p \times q$ and $z = (p-1)(q-1)$

(3) Select an integer number e where $1 < e < z$ as a public key. It should satisfy GCD (e, φ(n)) = 1.

(4) Compute the private key d such that $d = e - 1 \bmod z$)

(5) You can bundle private key pair as (n, d), and public key pair as (n, e)

2.2. Encryption

In RSA, plain text and cipher text both should not over $[\log_2 n]$. The cipher text is made by $\text{CipherText} = M_e \bmod n$.

2.3. Decryption

The text is recovered employing the private key(d) by $\text{PlainText} = C_d \bmod n$

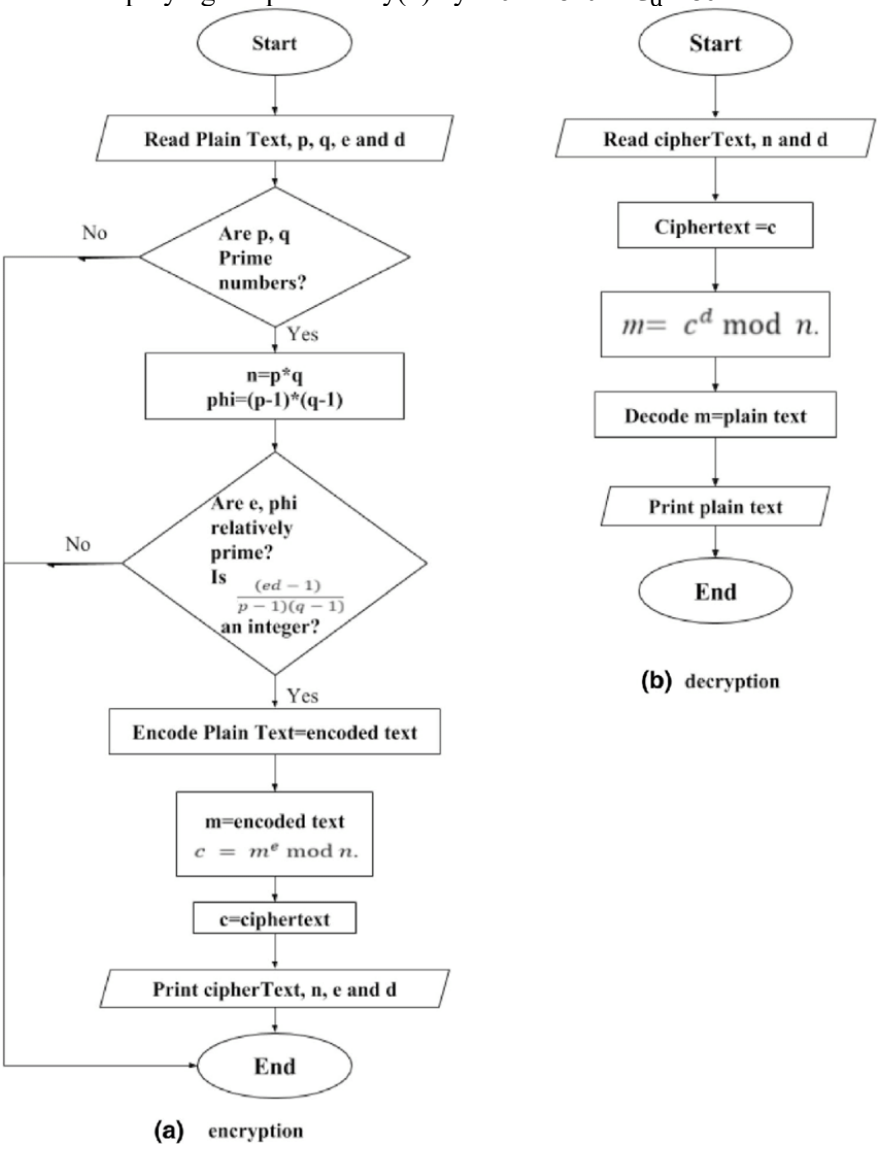


Figure1. Diagram of RSA encryption and decryption algorithms.

DOI: 10.54254/2755-2721/4/20230353

The RSA algorithm is based on the computation of fast exponentiation with modulo and Extended Euclid Algorithm. The strength of RSA is determined by the time complexity required for n to obtain p and q. Therefore, the larger the prime numbers p and q, the harder it is to factorize n.

2.4. Active authentication

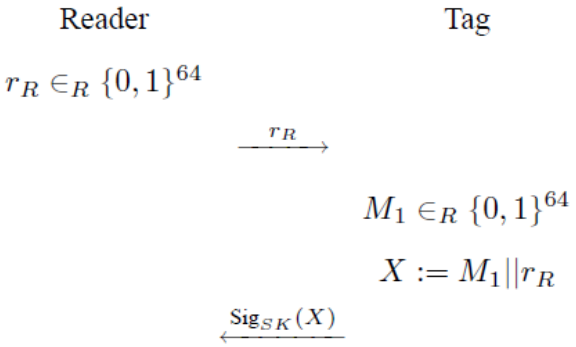


Figure2. Active authentication flow.

Reader can distinguish whether the data was not a clone of the original data by employing Active Authentication. Electronic passports that support active authentication an "active authentication key pair", the private key in key pair is stored in the chip; it will not be copied by anyways, but the chip can use it for internal authentication. The corresponding public key is stored as part of the e-passport data; thus, the checking system can read it and verify its authenticity by active authentication[5].

Figure 2 shows how active authentication works. Active authentication is achieved by implementing challenge-response authentication between the authentication terminal and the e-passport chip, which requires the chip to have data processing capabilities. To implement active authentication, the reader sends an 8-byte random number to the e-passport, which is encrypted with the private key (KPrAA) and returns the encrypted data to the reader, which decrypts it with the public key (KPuAA), and if the result is correct, the e-passport is proven to be legitimate. Here $\mathrm{Sig}_{sk}(X)$ is an RSA or Rabin-Williams signature with a 9796-2 padding, signed with thee-passport's secret key SK[6].

3. Implementation of RSA

The first step is creating two prime numbers with large bytes.

3.1. Generate a pseudo-prime number[7]

Algorithm 1 def probin(w): # w means byte

```
list = []
list.append('1')
for i in range(w - 2):
    c = random.choice(['0', '1'])
    list.append(c)
list.append('1')
res = int(''.join(list), 2)
return res
```

To generate a pseudo-prime number, I first generate a w-digit binary number and ensure that both the beginning and the end are 1, thus generating a large enough pseudo-prime number.

DOI: 10.54254/2755-2721/4/20230353

3.2. Determine if a pseudo-prime number is a prime number

3.2.1. Fermat Theorem

Algorithm 2 def mod_P(a, b, p):

```
if b == 0:
    return 1
res = mod_P((a * a) % p, b >> 1, p)
if b & 1 != 0:
    res = (res * a) % p
return res
```

I use the Miller-Rabin primality test, which is an implementation of Fermat's algorithm with the quadratic detection theorem, and its theoretical basis is derived from Fermat theorem. According to the Fermat theorem, if p is a prime(not 2), a is any integer, then $a^p \equiv a \bmod p$, thus $a^{p-1} \equiv 1 \bmod p$. Thus, we can randomly choose an integer a, then compute $a^{N-1} \bmod N$. If result is 1, we need further testing; if not, N is not a prime.

3.2.2. Miller-Rabin Test

Algorithm 3 def MillerRabin(a, p):

```
if mod_P(a, p - 1, p) == 1:
    u = (p - 1) >> 1
    while (u & 1) == 0:
        t = mod_P(a, u, p)
        if t == 1:
            u = u >> 1
        else:
            if t == p - 1:
                return True
            else:
                return False
    else:
        t = mod_P(a, u, p)
        if t == 1 or t == p - 1:
            return True
        else:
            return False
else:
    return False

def testMillerRabin(p, k):
    while k > 0:
        a = randint(2, p - 1)
        if not MillerRabin(a, p):
            return False
        k = k - 1
    return True
```

According to the Miller-Rabin test, we can know that is p is a prime, and $0 < x < p$, then $x \equiv 1 \bmod p$. As p already passes Fermat theorem, we can know that p is an odd number, then p-1 is an even number, so we note $p - 1 = m * 2^q$. Then $a^{p-1} \equiv 1 \bmod p$ is equivalent to $a^{m2^q} \equiv 1 \bmod p$. If $a^{m2^q} \equiv 1 \bmod p$ is valid, then according to the Miller-Rabin test, $a^{m2^{q-1}} \equiv 1 \bmod p$ is also valid.

Thus, we can further determine if a is a prime number by continuing to test if $a^{m2^{q-1}} \equiv 1 \bmod p$ is valid.

DOI: 10.54254/2755-2721/4/20230353

3.3. Generate a large prime number.

Algorithm 4 def makeprime(w):

```
while 1:
  d = probin(w)
  for i in range(50):
    u = testMillerRabin(d + 2 * i, 5)
    if u:
      b = d + 2 * i
      break
    else:
      continue
  if u:
    return b
  else:
    continue
```

The pseudo-prime number generated directly at random is probably not a prime number because of the distribution pattern of prime numbers. At this time, we just need to do more cyclic searching. The probability is that we can find a large prime number. If not, we can generate a pseudo-prime number and carry out the above nearby search.

3.4. Make up diction

Algorithm 5

```
dict = {'a': '31', 'b': '32', 'c': '33', 'd': '34', 'e': '35', 'f': '36', 'g': '37',
    'h': '38', 'i': '39', 'j': '10', 'k': '11', 'l': '12', 'm': '13', 'n': '14',
    'o': '15', 'p': '16', 'q': '17', 'r': '18', 's': '19', 't': '20', 'u': '21',
    'v': '22', 'w': '23', 'x': '24', 'y': '25', 'z': '26', ' ': '50'}
```

Here we create a dictionary for reference purposes only.

3.5. Implementation of conversion between characters and numbers

Algorithm 6 def transferToNum(str):

```
m = ""
for d in str:
  m += dict[d]
return m

def transferTostr(num):
n = ""
for i in range(0, len(num), 2):
  n += {value: key for key, value in dict.items()}[num[i] + num[i + 1]]
return n
```

Switch between characters and numbers here. transferTostr converts numbers to characters and transferToNum converts characters to numbers.

3.6. Implementation of Extended Euclid Algorithm

Algorithm 7 def ext_gcd(a, b):

```
if b == 0:
  x_ori = 1
  y_ori = 0
  x = x_ori
```

DOI: 10.54254/2755-2721/4/20230353

```
    y = y_ori
    e = a
    return e, x, y
  else:
    e, x_ori, y_ori = ext_gcd(b, a % b)
    x = y_ori
    y = x_ori - a // b * y_ori
    return e, x, y
```

This algorithm relies on the fact that if x divides both a and b, there will be a pair of coefficients (c, d) such that a * c + b * d = x. The algorithm finds these coefficients by repeatedly subtracting the smaller argument from the larger one until the smaller one becomes 0. Instead of repeatedly subtracting, it is better to calculate how many times b can be subtracted from a and then subtract b * c.

3.7. Implementation of fast exponentiation with modulo

Algorithm 8 def powmode(a, b, c):

```
  bina = bin(b)[2:][::-1]
  lenbina = len(bina)
  basea = []
  lastb = a
  basea.append(lastb)
  for _ in range(lenbina - 1):
    nextb = (lastb * lastb) % c
    lastb.append(nextb)
    lastb = nextb
  result = multi(basea, bina, c)
  return result % c
```

Algorithm 9 def expo(a, bina, n):

```
  r = 1
  for x in range(len(a)):
    array = a[x]
    if not int(bina[x]):
      continue
    r *= array
    r = r % n
  return r
```

The exponentiation implementation has the time complexity of logarithms. Instead of iterating from 1 to e and then multiplying b by r (the result), it iterates from the most significant bit of e to the least significant bit of e, doing r = r * r + bit at each bit. The reason for this is that if r equals b^x and you append a bit to the end of x, then the new x will be x*2+bits

3.8. Generate keys and test[8]

Algorithm 10 def probin(w): def gen_key(pri, pub):

```
  n = pri * pub
  fy = (pri - 1) * (pub - 1)
  e = 35537
  q = e
  w = fy
  a = ext_gcd(q, w)[1]
```

DOI: 10.54254/2755-2721/4/20230353

```
    if a < 0:
        d = a + fy
    else:
        d = a
    print("Public key:" + "(" + str(n) + "," + str(e) + ")\nPrivate key:" + "(" + str(n) +
"," + str(d) + ")")
    return (n, e), (n, d)
```

It's the process of generating both private and public keys.

```
def encrypt(p, pub):
    a = pub[0]
    b = pub[1]
    c = exp_mode(p, a, b)
    return c
```

Algorithm 11 def probin(w): def decrypt(c, selfkey):

```
    a = selfkey[0]
    b = selfkey[1]

    p = exp_mode(c, a, b)
    return p
```

Encryption m is the message being encrypted Encryption becomes c, decryption c is the ciphertext, decryption is plaintext m

Algorithm 12 if __name__ == "__main__":

```
    p = makeprime(528)
    print("p:", p)
    q = makeprime(528)
    print("q:", q)

    print("1.Generate public-private key")
    pubkey, selfkey = gen_key(p, q)

    print("2.enter plain text")
    plaintext = str(input())
    m = int(transferToNum(plaintext))

    print("3.Encrypting messages with public keys")
    c = encrypt(m, pubkey)
    print("Ciphertext:", c)

    print("4.Decryption with private key")
    d = decrypt(c, selfkey)
    print("Plain text:", transferTostr(str(d)))
```

Input plain text and encrypt it, finally output the decrypted plain text

Here is one of the results:

p:

652233718493320877818820779545371705680961052798589898595147520614167189096686868461294192790286645206983890442437887949322625381972890759572100761457802921119

q:

498364561112432783105777668462732930875764453394624013864964888921995608705039064143128664052696147382863612643768421305137907543935284078751988673868336345163

DOI: 10.54254/2755-2721/4/20230353

4. Generate public-private key

Public Key:

(32505017085965389291194565057598191591182942653060713588357421856145380354226763257974265519960108109171730529419716183538536392849170964922342176577413838232513894953377976385400638739108421084259146744366780754658706272611981299586231818322509501698863855836663879643590240451864612954512634041603836945941764619739735537)

Private Key:

(32505017085965389291194565057598191591182942653060713588357421856145380354226763257974265519960108109171730529419716183538536392849170964922342176577413838232513894953377976385400638739108421084259146744366780754658706272611981299586231818322509501698863855836663879643590240451864612954512634041603836945941764619739763734968921126384495327047674633255200878730451790261497502466582328561867420459347036801683620461295187724999014149299855508789535701742853611809210518508822735482216818653882584177205598511636988211774216252594270664091115031504631404080319803123839798783688344028253951723325009011375262745244266907820240020831697)

(2)enter plain text

i love you

5. Encrypting messages with public keys

Ciphertext:

19848197643625443631282886613514069977006206735976147072274850376950862493090745401448392183454411997743153808822152106669925770709449471222631640320783517616967068652059296598093805605206363035289033619058079860223813263512247030374104131241265256176825446813894495930810329450992488571247162278486069643414745688989 8

Plain text: i love you

6. Discussion

The RSA algorithm modeled in this thesis has two advantages, one of which is its speed advantage over other primality tests. The Miller-Rabin algorithm ensures that the algorithm can test the data to be measured at a very high speed in terms of processing large numbers. At the same time, the fast-exponential approach to computation ensures that the processing system does not slow down due to too much data. The second is stability. The algorithm requires a small processing capacity, which makes it easier for the processing system to process large numbers without excessive loss of hardware (chip) capacity and lifetime, which also ensures that the algorithm does not make errors in the process of processing data due to the lack of speed and capacity of the chip.

However, at the same time, the present algorithm also has three drawbacks. The Miller-Rabin test is a more conservative test method, and the accuracy is very much improved compared to the Fermat test. It has been shown that the error probability of the Miller-Rabin test is about $\frac{1}{4}$ after specifying any 1-time a. After rotating k times a, the error rate of testing prime numbers is $\left(\frac{1}{4}\right)^{k}$. When k is 6, the probability is less than $\frac{1}{1000}$, but this also shows that there is no 100% guarantee that the number must be a prime number. Second, the method used in the algorithm to generate the prime number is the longer string to represent a binary number, and then convert it to the corresponding decimal integer, but the extra-long binary number may lead to digital overflow when converting to decimal, so that python cannot save it as the correct value and thus lead to data errors. The third is security. It is very simple for an attacker to track or eavesdrop on the data processing part of the algorithm, and once the attacker obtains the specific number of prime numbers, the corresponding public and private keys will be cracked as a result.

7. Conclusion

The Python3-based RSA implementation is feasible but cannot be applied to the actual e-passport authentication. The security and correctness pitfalls of the algorithm lead to the inability to ensure the

DOI: 10.54254/2755-2721/4/20230353

correctness and security of the e-passport authentication process, which also jeopardizes the security of the passport holder. The authors believe that it is not possible to enhance the algorithm based on the dry algorithm, and that the Miller-Rabin algorithm is inherently problematic, which cannot be solved by adding more algorithms. Therefore, the algorithm needs to be replaced with an algorithm that discriminates larger prime numbers with higher correctness. At the same time, additional algorithms can solve the security problem, but they will inevitably make the procedure too redundant and lead to a reduction in the speed of the processing system.

This also illustrates, in disguise, the superiority of the algorithm in the real-life e-passport system. The e-passport uses a new processing algorithm that allows it to identify and authenticate e-passport information more quickly than the algorithm designed in this paper. Also, the e-passport uses advanced anti-eavesdropping technology in security to ensure the security of the e-passport in the authentication process. Therefore, this paper argues that the use of the e-passport that takes biometric technology is difficult to be copied and the risk of information leakage is extremely low[9]. Meanwhile, the algorithm in this paper should focus on how to improve the efficiency of the code so that the code still has fast processing speed in the face of large amounts of data and work on solving the possible eavesdropping phenomenon in the code to improve the security of the code.

Reference

[1] "Security Mechanisms in Electronic ID Documents." Federal Office for Information Security, J anuary 20, 2021. https://www.bsi.bund.de/EN/Topics/ElectrIDDocuments/SecurityMechanis ms/securBAC/bac_node.html.

[2] "Doc Series // ." Doc Series. Accessed July 7, 2022. https://www.icao.int/publications/pages/p ublication.aspx?docnum=9303.

[3] "Basics of Epassport Cryptography // ." Basics of ePassport Cryptography. Accessed July 7, 2 022. https://www.icao.int/Security/FAL/PKD/BVRT/Pages/Basics.aspx.

[4] Aljuaid, Nouf & Gutub, Adnan. (2019). Combining RSA and audio steganography on personal computers for enhancing security. SN Applied Sciences. 1. 10.1007/s42452-019-0875-8.

[5] Shehata, Khaled, Hanady Hussien, and Sara Yehia. "FPGA Implementation of RSA Encryption Algorithm for E-Passport Application." The International Conference on Electrical Engineeri ng 9, no. 9th (2014): 1–5. https://doi.org/10.21608/iceeng.2014.30363.

[6] Liu, Yifei, Timo Kasper, Kerstin Lemke-Rust, and Christof Paar. "E-Passport: Cracking Basic Access Control Keys." On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, O DBASE, GADA, and IS, n.d., 1531–47. https://doi.org/10.1007/978-3-540-76843-2_30.

[7] "Python implements the rsa algorithm." Python implements the RSA algorithm. Accessed July 3 1, 2022. https://blog.csdn.net/qq_36944952/article/details/103973849?spm=1001.2101.3001. 6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFrom Baidu~default-1-103973849-blog-46356869.pc_relevant_multi_platform_featuressortv2dupr eplace&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommen dFromBaidu~default-1-103973849-blog-46356869.pc_relevant_multi_platform_featuressort v2dupreplace.

[8] Sharma, Shruti, and Harshali Zodpe. "Implementation of Cryptography Algorithm for E-Passpo rt Security." 2016 International Conference on Inventive Computation Technologies (ICICT), 2016. https://doi.org/10.1109/inventive.2016.7830150.

[9] Lekkas, Dimitrios & Gritzalis, Dimitris. (2007). E-Passports as a Means Towards the First Worl d-Wide Public Key Infrastructure. 4582. 34-48. 10.1007/978-3-540-73408-6_3.