

The Offline and Online Method Implementation with RSA Algorithm in Software Product Key Generator

Muhammad Daru Darmakusuma - 13518057¹

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518057@std.stei.itb.ac.id¹

Abstract—Product key or software key is a specific software-based key for a computer program to certify that the copy of the program is original. Product keys contain a series of number and/or letters that will be entered by the user during the installation of computer software. The sequence is passed to a verification function in the program. The verification function will match the sequence depending to the mathematical algorithm implemented. The algorithm that can be implemented is RSA. RSA algorithm is a asymmetric cryptography algorithm. The implementation can be done through offline or online process.

Keywords—product key, RSA algorithm, cryptography, software

I. INTRODUCTION

Product key or software key is a usually unique, alphanumeric code of any length required to install a software especially for paid software that implemented by many software developer. They help software developers ensure every copy of their software was legally purchased. Most software, like operating systems and other programs, require product keys as a general rule these days.

A product key is like a password for a program or software. This password is given upon buying the software and can only be used within the specific application that had been purchased. Product key usually can only be used once per user or sometime strictly per installation. Without a product key, the program will most likely run as a trial of the full version. Open source and free software programs barely require a product key except the developer implements it for statistical purpose.

Product keys usually only can be used once in installation of the program but some product key servers allow for the same key to be used by any number of people as long as they are not used simultaneously. In the implementation there's a limited slot of product keys, so if the software using the key is not used and shut down, another can be used and opened at the same slot.

Product key often produced and generated by a computer program called a key generator (key-gen). Key generator generates a product licensing key to activate a software application. Keygens can be distributed legally by software manufacturers for licensing software in commercial

environments to bulk licensing software in a enterprise. Keygen also can be distributed illegally for software piracy.

Cryptography is an element that being used to make a key generator and verification function for product key. The cryptography algorithm and the methods to process it can be the factors of preventing illegal distribution of key generator. The algorithm that usually used at a key generator is RSA algorithm with two type of process, offline and online.

II. THEORY

A. Cryptography

Cryptography is an important tool for information security. Cryptography is a subject that learn mathematical technic that related to information security like secrecy, data integrity, and authentication [1].

In cryptography, there are multiple terminologies, as follows:

- Message, readable information and can be receipted as visual or audio.
- Sender, someone who sends the message.
- Receiver, someone who receive the message.
- Ciphertext, message that have been encrypted so it cannot be interpreted.
- Encryption, process of encrypting message to ciphertext.
- Decryption, process of decrypting ciphertext to plaintext or original message
- Cipher, algorithm that being used to encrypt and decrypt the message.
- Key, the parameter that being used to encrypt and decrypt the message.
- Eavesdropper, someone or something that trying to catch the message while being transmitted.
- Cryptanalysis, knowledge and art of cracking ciphertext to plaintext without knowing the key.
- Cryptology, study on cryptography and cryptanalysis.

B. Asymmetric Cryptography

Asymmetric cryptography is created in 1976 to fix the disadvantage of symmetric cryptography that use the same key

for encryption and decryption [2]. The sender and the receiver have two different key to encrypt and decrypt the message. The two keys are private and public key. The public key is used to encrypt the message and the private key is used to decrypt the message.

The flow of message transmission in this cryptography is the sender have the public key to encrypt the message and then the receiver can decrypt it with their own private key [2]. The term public in the key is that the key is known to public for the use on encrypting the message. On the other hand, the private key is being kept secret to decrypt the message.

For comparison, the symmetric and the asymmetric cryptography advantages and disadvantages, as follows:

	Symmetric Cryptography	Asymmetric Cryptography
Advantages	<ul style="list-style-type: none"> Encryption and decryption process doesn't take much time Size of the key relatively short Authentication of the message is known from the ciphertext 	<ul style="list-style-type: none"> Only private key that must be kept secret when communicating Tuples of public key and private key don't need to be changed for a long period Can be used for digital signing
Disadvantages	<ul style="list-style-type: none"> Key and the message must be transmitted in a different line to keep secrecy Key should be changed every session 	<ul style="list-style-type: none"> Encryption and decryption process take a long time to be done Ciphertext size usually more bigger than plaintext Key size relatively bigger

C. RSA Algorithm

RSA algorithm is one of popular asymmetric cryptography and have many application of it. The security of RSA algorithm is located at the difficulty in factorization of integers to prime number. The properties of RSA algorithm consist of variables as follows:

- p and q prime numbers (private)
- $n = p \cdot q$ (public)
- $(n) = (p - 1)(q - 1)$ (private)
- e (encryption key) (public)

○ Terms: $\text{GCD}(e, \Phi(n)) = 1$, GCD = greatest common divisor

- d (decryption key) (private)
 - d counted from $d \cdot e^{-1} \pmod{\Phi(n)}$
- m (plaintext) (private)
- c (ciphertext) (public)

The flow of the encryption and decryption process can be done as follows:

Generating a public key:

1. Select two prime no's. Suppose $P = 53$ and $Q = 59$.
2. Now First part of the Public key : $n = P \cdot Q = 3127$.
3. We also need a small exponent say e, but e must be an integer, not be a factor of n.
 $1 < e < \Phi(n)$ [$\Phi(n)$ is discussed below],
Let us now consider it to be equal to 3.
4. Our Public Key is made of n and e

Generating a private key:

1. We need to calculate $\Phi(n)$:
Such that $\Phi(n) = (P-1)(Q-1)$
so, $\Phi(n) = 3016$
2. Now calculate Private Key, d :
 $d = (k \cdot \Phi(n) + 1) / e$ for some integer k
For $k = 2$, value of d is 2011.

Encrypting and decrypting process:

1. Convert letters to numbers : $H = 8$ and $I = 9$
2. Thus Encrypted Data $c = 89e \pmod{n}$.
Thus our Encrypted Data comes out to be 1394
3. Now we will decrypt 1394 :
Decrypted Data = $cd \pmod{n}$.
Thus our Encrypted Data comes out to be 89
 $8 = H$ and $9 = I$ i.e. "HI".

D. Software Activation

Software activation is a process of activating or verifying if the software that being installed or used is legally purchased. Software activation usually need a product key that need to be inserted. Software activation mainly have two methods of verification and usage of product key. The two methods of activation as follows:

1. Offline Activation

Offline activation is a software activation process without needing a connection to the internet. The verification process in this method usually done by the installer or the software function alone by decrypting the product key to the correct plaintext with different private key for each user. This method is prone to be attacked by a cracker that created a key generator with the same cipher.

2. Online Activation

Online activation is a software activation process that need a connection to the internet. The verification process in this method usually done similar as offline activation but the key difference is the plaintext is served for different user, so it the cracker can't produce a key generator that randomly generate a key. Product key sometimes is unnecessary for modern activation because most of it now use the user account to activate the product.

III. EXPERIMENT

For the experiment, the process that will be analyzed are the offline and online method of software activation with RSA algorithm. The output and parameter that will be analyzed are the security of the key authenticity. In the offline method will only use one plaintext as the verifier of the product and for the online method will use a text file that pretends as a database of plaintext slot of each users.

A. Implementation

The implementation of the key generator were developed in Python 3.7 with RSA algorithm. The program developed is helped by an utility file to calculate the key generated by the RSA algorithm. The source codes for the implementation as follows:

Utility file:

```
from random import getrandbits, randrange

def gcd(a, b):
    while b > 0:
        a, b = b, a % b
    return a

def lcm(a, b):
    return a * b // gcd(a, b)

def is_prime(n, k=128):
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 2 == 0:
        return False

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

    for _ in range(k):
        a = randrange(2, n - 1)
        x = pow(a, r, n)
        if x != 1 and x != n - 1:
            j = 1
            while j < s and x != n - 1:
                x = pow(x, 2, n)
```

```
        if x == 1:
            return False
        j += 1
        if x != n - 1:
            return False
    return True

def genPrime(length=1024):
    while True:
        p = getrandbits(length)
        p |= (1 << length - 1) | 1
        if is_prime(p, 128):
            return p

def is_square(n):
    if n < 0:
        return False
    prev = n
    x = n // 2
    while x * x != n:
        x = (x + (n // x)) // 2
        if x >= prev:
            return False
        prev = x
    return True

def sqrtmod(a, p):
    a = a % p
    res = []
    for x in range(2, p):
        if (x * x) % p == a:
            res.append(x)
    return res

def multi_inverse(e, totient):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_totient = totient

    while e > 0:
        temp1 = temp_totient // e
        temp2 = temp_totient - temp1 * e
        temp_totient = e
        e = temp2

        x = x2 - temp1 * x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y

    if temp_totient == 1:
        return d + totient

def genE(totient):
    while (True):
        e = randrange(2, totient)
        if (gcd(e, totient) == 1):
            return e
```

```

def genPrimeRange(start, end):
    while (True):
        res = randrange(start, end)
        if (is_prime(res)):
            return res

def str_to_int(m):
    _m = ""
    for c in m:
        i_c = str(ord(c))
        i_c = "0" * (3 - len(i_c)) + i_c
        _m += i_c
    return int(_m)

def int_to_str(m):
    _m = str(m)
    _m = "0" * ((3 - (len(_m) % 3)) if len(_m)
% 3 > 0 else 0) + _m
    x = ""
    for i in range(0, len(_m), 3):
        c = chr(int(_m[i:i+3]))
        x += c
    return x

def bin_to_int(m):
    _m = ""
    for b in m:
        i_b = str(int(b))
        i_b = "0" * (3 - len(i_b)) + i_b
        _m += i_b
    return int(_m)

def int_to_bin(m):
    _m = str(m)
    _m = "0" * ((3 - (len(_m) % 3)) if len(_m)
% 3 > 0 else 0) + _m
    x = bytes()
    for i in range(0, len(_m), 3):
        b = int(_m[i:i+3])
        b = b.to_bytes(1, 'big')
        x += b
    return x

```

RSA cipher tool:

```

import utils

class RSA:
    def genKey():
        p = 0
        q = 0
        while (p == q):
            p = utils.genPrimeRange(100, 500)
            q = utils.genPrimeRange(100, 500)
        n = p * q
        totient = (p - 1) * (q - 1)
        e = utils.genE(totient)
        d = utils.multi_inverse(e, totient)

        return e, d, n

```

```

def encrypt(plaintext, e, n,
block_length=2):
    encrypted_blocks = []
    block = ord(plaintext[0])
    for i in range(1, len(plaintext)):
        if (i % (block_length) == 0):
            encrypted_blocks.append(block)
            block = 0
            block = block * pow(10,
block_length+1) + ord(plaintext[i])
            encrypted_blocks.append(block)
        for i in range(len(encrypted_blocks)):
            encrypted_blocks[i] =
str(pow(encrypted_blocks[i], e, n))
    ciphertext = " ".join(encrypted_blocks)

    return ciphertext

def decrypt(ciphertext, d, n,
block_length=2):
    blocks = ciphertext.split(' ')
    _blocks = []
    for b in blocks:
        _blocks.append(int(b))
    plaintext = ""
    for i in range(len(_blocks)):
        _blocks[i] = pow(_blocks[i], d, n)
        temp = ""
        for j in range(block_length):
            temp = chr(_blocks[i] % pow(10,
block_length+1)) + temp
            _blocks[i] //= pow(10,
block_length+1)
        plaintext += temp

    return plaintext

```

Offline keygen:

```

from RSA import RSA

e, d, n = RSA.genKey()
f = open('RSAoff.pri', 'wt')
f.write(str(d) + "," + str(n))
f.close()
f = open('RSAoff.pub', 'wt')
f.write(str(e) + "," + str(n))
f.close()
f = open('offline.txt', 'rt')
m = f.read()
c = RSA.encrypt(m, e, n)
print("Product key: {}".format(c))

```

Offline verifier:

```

from RSA import RSA

print('====RSA====')
f = open('offline.txt', 'rt')

```

```

m = f.read()
f.close()
c = input("Insert Product Key: ")
f = open("RSAoff.pri", "rt")
key_string = f.read().split(",")
f.close()
n = int(key_string[0])
d = int(key_string[1])
p = RSA.decrypt(c, d, n)
print(p)
print("Correct product key: {}".format(c))
print("Product activated: {}".format(p == m))

```

Online keygen:

```

from RSA import RSA
import random

e, d, n = RSA.genKey()
f = open('RSAon.pri', 'wt')
f.write(str(d) + "," + str(n))
f.close()
f = open('RSAon.pub', 'wt')
f.write(str(e) + "," + str(n))
f.close()
f = open('online.txt', 'rt')
marr = f.read().splitlines()
m = random.choice(marr).split(",")[0]
c = RSA.encrypt(m, e, n)
print("Product key: {}".format(c))

```

Online verifier:

```

from RSA import RSA

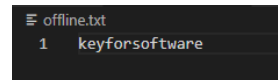
print('====RSA====')
f = open('online.txt', 'rt')
marr = f.read().splitlines()
c = input("Insert Product Key: ")
f = open("RSAoff.pri", "rt")
key_string = f.read().split(",")
f.close()
n = int(key_string[0])
d = int(key_string[1])
p = RSA.decrypt(c, d, n)
print("Correct product key: {}".format(c))
act = False
if str(p) + '\n' in marr:
    act = True
    marr.remove(str(p) + '\n')
    marr.append(str(p) + '\ny')
print("Product activated: {}".format(act))
f = open('online.txt', 'wt')
for key in marr:
    f.write(key + "\n")
f.close()

```

B. Case Studies

1) Offline Activation

To test which is the most secure for the distribution of software license, we need to make the plaintext that will be the verifier for the product key. For the offline method was used a single plaintext file to verify the product keys as follows:

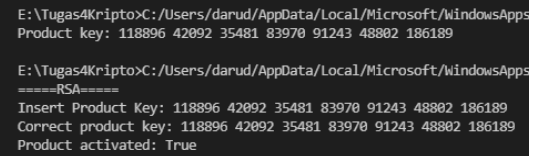


```

offline.txt
1 keyforsoftware

```

For the output of the keygen and verifier with offline method shown as follows:



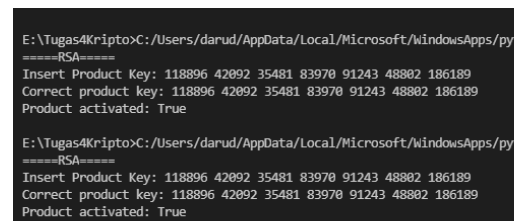
```

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/WindowsApps/
Product key: 118896 42092 35481 83970 91243 48802 186189

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/WindowsApps/
====RSA====
Insert Product Key: 118896 42092 35481 83970 91243 48802 186189
Correct product key: 118896 42092 35481 83970 91243 48802 186189
Product activated: True

```

And if we insert the same product key, it still will be accepted, as follows:



```

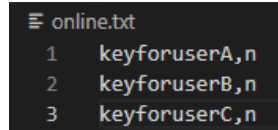
E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/WindowsApps/py
====RSA====
Insert Product Key: 118896 42092 35481 83970 91243 48802 186189
Correct product key: 118896 42092 35481 83970 91243 48802 186189
Product activated: True

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/WindowsApps/py
====RSA====
Insert Product Key: 118896 42092 35481 83970 91243 48802 186189
Correct product key: 118896 42092 35481 83970 91243 48802 186189
Product activated: True

```

2) Online Activation

For the online method was used a list of plaintext with label if the key was used or not in a text file to verify the product keys as follows:

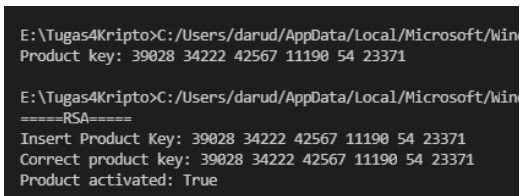


```

online.txt
1 keyforuserA,n
2 keyforuserB,n
3 keyforuserC,n

```

For the output of the keygen and verifier with online method shown as follows:



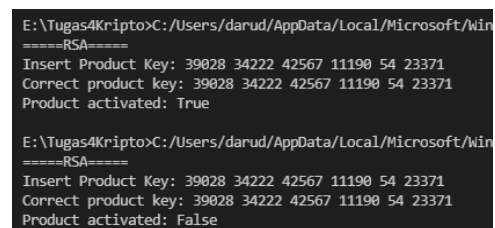
```

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/Win
Product key: 39028 34222 42567 11190 54 23371

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/Win
====RSA====
Insert Product Key: 39028 34222 42567 11190 54 23371
Correct product key: 39028 34222 42567 11190 54 23371
Product activated: True

```

And if we insert the same product key, it will be rejected, as follows:



```

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/Win
====RSA====
Insert Product Key: 39028 34222 42567 11190 54 23371
Correct product key: 39028 34222 42567 11190 54 23371
Product activated: True

E:\Tugas4Kripto>C:/Users/darud/AppData/Local/Microsoft/Win
====RSA====
Insert Product Key: 39028 34222 42567 11190 54 23371
Correct product key: 39028 34222 42567 11190 54 23371
Product activated: False

```

C. Analysis

From the experiment we can analyze that both of the method can be utilized to generate a key and verify it. But for the distribution of the keygen, there are major difference to consider the security and the integrity of the key that being used. The safest method to generate and verify the software is by online method

IV. CONCLUSION

Online activation for product key generator and verifier is the safest implementation for software distribution. The online method cannot use the same key repeatedly and safe for wide distribution for the keygen. The crackers will have a difficult time to develop a keygen that can match the plaintext.

ACKNOWLEDGMENT

First, the author would like to be thankful to God. The author like to express his gratitude and appreciate Mr. Rinaldi Munir for his teachings and lectures in Cryptography. The author also thankful for his family and friends.

REFERENCES

- [1] Munir, Rinaldi. 2019. Slide Kuliah Pengantar Kriptografi IF4020 Kriptografi. Bandung: Institut Teknologi Bandung
- [2] Munir, Rinaldi. 2019. Slide Kuliah Kriptografi Kunci-Publik IF4020 Kriptografi. Bandung: Institut Teknologi Bandung

- [3] Munir, Rinaldi. 2019. Slide Kuliah Algoritma RSA IF4020 Kriptografi. Bandung: Institut Teknologi Bandung
- [4] <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>, visited at 19 December 2021, 17.00 WIB
- [5] <https://keygen.sh/blog/how-to-generate-license-keys-in-2021/>, visited at 19 December 2021, 20.00 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2021



Muhammad Daru Darmakusuma, 13518057