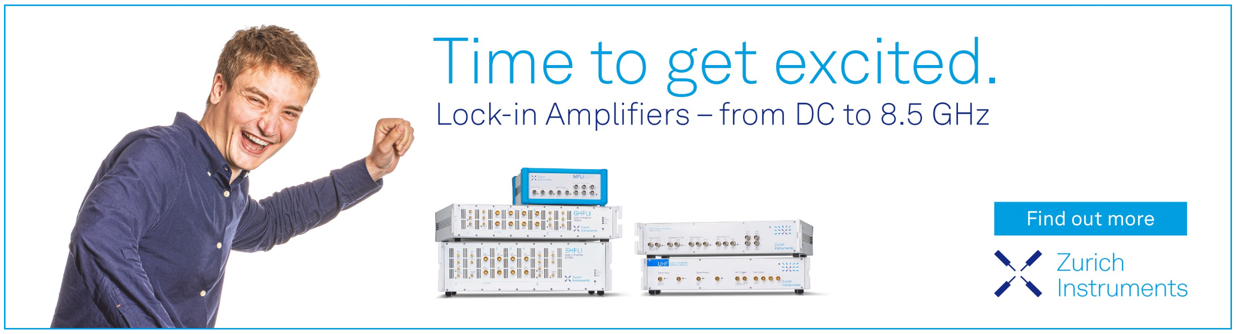
Multi prime numbers principle to expand implementation of CRT method on RSA algorithm

Cite as: AIP Conference Proceedings **2331**, 060009 (2021); https://doi.org/10.1063/5.0041856
Published Online: 02 April 2021

Nanang Triagung Edi Hermawan, Edi Winarko and Ahmad Ashari



Multi Prime Numbers Principle to Expand Implementation of CRT Method on RSA Algorithm

Nanang Triagung Edi Hermawan[1, a)], Edi Winarko[2, b)], and Ahmad Ashari[2, c)]

[1]*Indonesian Nuclear Energy Regulatory Agency, Jakarta, Indonesia*
[2]*Department of Computer Science and Electronic, Faculty of Mathematic and Natural Science, Gadjah Mada University, Yogyakarta, Indonesia*

[a)]Corresponding author: n.triagung@bapeten.go.id
[b)]e.winarko@ugm.ac.id
[c)]ashari@ugm.ac.id

Abstract. One of the uses of the Chinese Remainder Theorem (CRT) is to shorten the time in the RSA Cryptosystem decryption process. However, in applying a combination of the CRT method and the RSA Algorithm with Python programming modules, recursion errors often occurred when determining the inverse modulus to establish the private key. The error was caused by limitation iterations of the related algorithm. Some Python library needs more iteration processes to calculate the inverse modulus for larger key size. This is a pure computational problem. The standard RSA algorithm uses two prime numbers to generate their key pairs. The multi-prime RSA modification based on more than two prime numbers can be applied to solve the problem. A modified RSA was applied in Python programming that generates keys based on 2, 4, 8, 16, and 32 prime numbers. Regarding on our experiments, the combination of the CRT method and RSA algorithm based on four prime numbers can be applied without recursion error up to a key length of 6280 bits, with eight primes up to 12,560 bits, with sixteen primes up to 25,120 bits, and with thirty-two primes up to 50,240 bits. Implementation of more multi-prime numbers can expand to combine the CRT method and RSA Algorithm for longer key sizes by approximation relation y = 1567,8x + 63,333, which y is the key size and x is the number of primes numbers. The method also can increase key generation and decryption rate significantly.

INTRODUCTION

Data or information is people's daily needs. The valid information should be guaranteed by adequate security measures. It should be fulfilled security requirement aspects, such as confidentiality, integrity, and availability [1]. Cryptography is a method of securing data to address data confidentiality and integrity issues. Cryptography includes symmetric and asymmetric cryptography. The symmetric cryptography, also commonly called secret cryptography, uses the same and identic key to encrypt and decrypt a message. On the other hand, asymmetric cryptography uses two different keys: the public key, and the other is the private key. The public key is used to encrypt a message, and the private key is used to decrypt the message. Asymmetric cryptography is essential to authentication and key exchange of secret key. It's also called public-key cryptosystem [2].

RSA is the most popular asymmetric or public-key cryptosystem. This cryptosystem comprises three phases: the key generation phase, the encryption phase, and the decryption phase [3]. The duration of RSA algorithm implementation is dominated by the key generation phase and decryption phase. Gupta and Sharma pointed out that each initialization of the RSA process requires a random selection of two very large prime numbers (p and q) [4]. This initialization process is very influenced by keys generation speed. Another study by Patidar and Bhartiya identified computational time cost on the decryption process [5]. The Chinese Remainder Theorem is commonly implemented to solve the second problem [1, 5].

Some advanced language programs are usually used to implement the RSA algorithm. The one best is Python language programming [7]. The newest version of the program is Python 3 series. Besides its simple, easy, and familiar syntax code to implement, some unperfect algorithm generates some errors. In our case, a recursion error appears when such a default modulo inverse algorithm is used to calculate the modulo inverse of the public key in the keys generation phase. It happened when the default algorithm in Python 3 programming was used. This condition makes limitations to implement the RSA cryptosystem for longer key sizes.

On the other hand, the security of the RSA cryptosystem depends on the length of key sizes. Barakat et al. recommend the key size longer than or the same 2048 bits should be used to ensure the security of a message [8]. This fact makes us should use an RSA algorithm with key sizes longer than 2048 in the future to ensure data security.

Based on the above problem described before, this research has a goal to propose a method for expanding key sizes of the RSA cryptosystem in Python 3 implementation. A modified RSA cryptosystem with multi-prime, more than two prime numbers (2, 4, 8, 16, and 32) is proposed to achieve the goal. An analysis focused only on computational parameters, especially duration time of key generation, encryption, decryption, and key sizes. The security aspect of the proposed method is not discussed in this paper.

Standard RSA Algorithm

RSA algorithm was developed in 1978 by Ronald Rivest, Adi Shamir, and Leonard Adleman [9]. The first algorithm was known to be suitable for signing and encryption, and one of the first great advances in public-key encryption [10]. RSA is a public-key cryptographic system that uses the concept of number theory. Its security depends on the complexity of prime factorization of large numbers [3], which is a well-known mathematical problem with no known effective solution. This makes RSA one of the most widely used technique for symmetric key cryptography encryption and digital signature standards [11]. Generally, the RSA algorithm comprises of three phases, which are the key generation phase, encryption phase, and the decryption phase [3].

The standard RSA algorithm is generated based on two prime numbers. The steps involved in key generation, encryption, and decryption are as follows [1],[3]:

1. choose two different prime numbers p and q, which $p \neq q$;
2. calculate the common module n such that $n = p\,q$;
3. calculate the Euler's totient function phi: $\emptyset(n) = (p - 1)\,(q - 1)$;
4. select integer e which is the encryption (public) key such that: GCD $(\emptyset(n), e) = 1;\ 1 < e < \emptyset(n)$;
5. determine d is the decryption (private) key such that $d = e - 1$ mod $\emptyset(n)$;
6. the public key is $PU = [\,e, n]$ and the private key is $PR = [\,d, n]$.

Monte Carlo and Miller-Rabin are the most implemented algorithms to generate two large random prime numbers (p, and q) (1).

Encryption is a process to encode a plain text into a decoded message in such a way that only the authorized parties can read it. The encryption algorithm is given by Equation 1 (1).

$$C = M^{e} mod\ n \qquad (1),$$

where M is a block of plain text, C is a cipher or decoded text, e is the public key, and n is common module of the cryptosystem. Decryption is a process to decode a cipher text to get original plain text. The decryption algorithm is given by Equation 2 (1).

$$M = C^{d} mod\ n \qquad (2)$$

where M is a block of plain text, C is a cipher or decoded text, d is the public key, and n is common module of the cryptosystem.

The size of the private key (d) is very large compare to the public key component (e). It causes decryption calculation need more extra time and computational resources. This problem can be solved by implementing the Chinese Remainder Theorem (CRT) method. The CRT is used to shorten the key exponent bit size so that the decryption process is faster. The RSA-CRT private key establishment procedure is as follows:

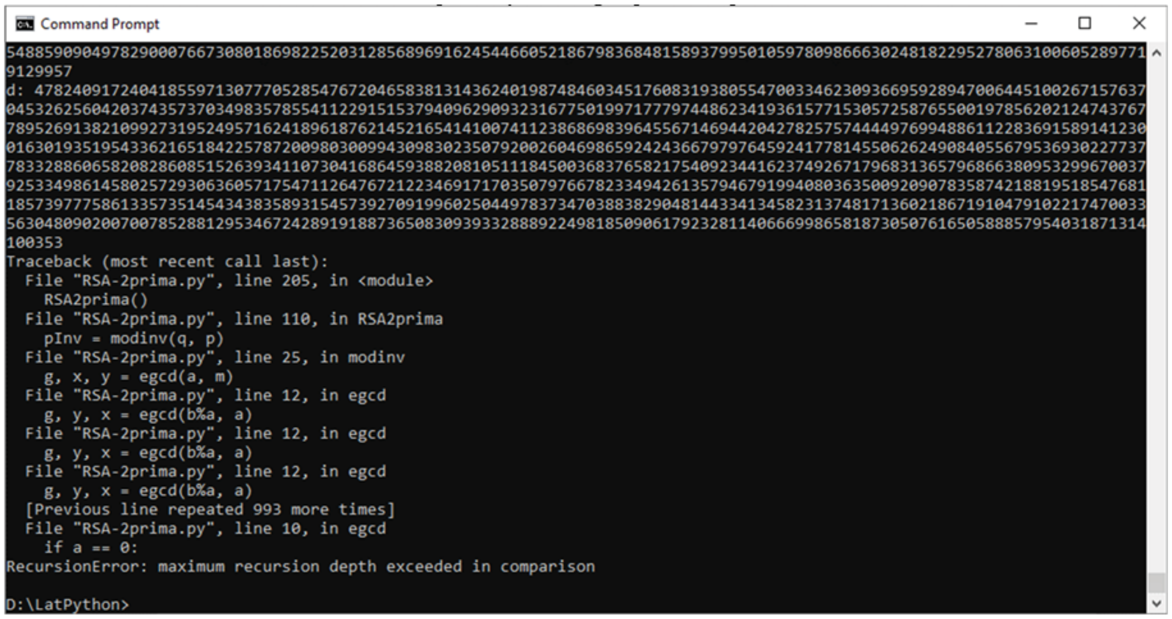
1. calculate value dp, dq, and qInv as private key components by the equation (1):
   - $dp = d\ mod\ (p - 1)$
   - $dq = d\ mod\ (q - 1)$
   - $qInv = q - 1\ mod\ p$
2. conduct calculation use above private key component by some steps below:
   - $m1 = Cdp\ mod\ p$
   - $m2 = Cdq\ mod\ q$
   - $h = qInv(m1 - m2)\ mod\ p$

$$M = m_2 + hq \tag{3}$$

By implementing the CRT method, the decryption process will be faster. The average speed by Equation 3 reaches four times faster than by Equation 2.

METHOD

As mentioned in the introduction, the recursion error is a problem in Python 3 when a default algorithm to find the inverse module is executed for determining a private key as an inverse module of the previous established public key (e). The algorithm is very related with great common division calculation. When the algorithm is executed for key sizes longer than 3200 bits, a recursion error exception will be appeared and stops the primality test processing. Figure 1 describes the recursion error message in Python 3 programming.

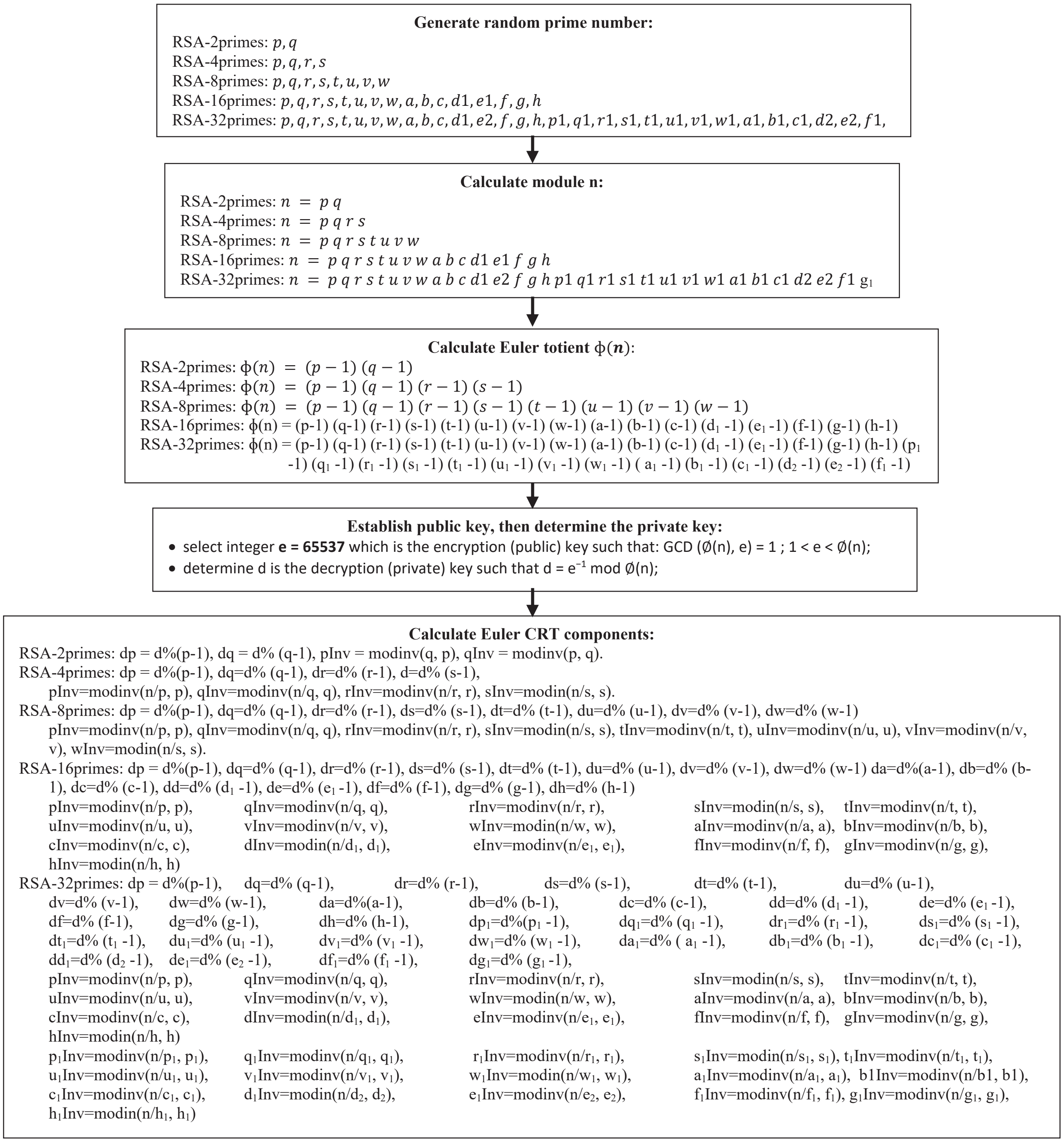


```
5488590904978290007667308018698225203128568969162454466052186798368481589379950105978098666302481822952780631006052897719129957
d: 4782409172404185597130777052854767204658381314362401987484603451760831938055470033462309366959289470064451002671576370453262560420374357370349835785541122915153794096290932316775019971777974486234193615771530572587655001978562021247437677895269138210992731952495716241896187621452165414100741123868698396455671469442042782575744449769948861122836915891412300163019351954336216518422578720098030099430983023507920026046986592424366797976459241778145506262490840556795369302277377833288606582082860851526393411073041686459388208105111845003683765821754092344162374926717968313657968663809532996700379253349861458025729306360571754711264767212234691717035079766782334942613579467919940803635009209078358742188195185476811857397775861335735145434383589315457392709199602504497837347038838290481443341345823137481713602186719104791022174700335630480902007007852881295346724289191887365083093933288892249818509061792328114066699865818730507616505888579540318713141003533
Traceback (most recent call last):
  File "RSA-2prima.py", line 205, in <module>
    RSA2prima()
  File "RSA-2prima.py", line 110, in RSA2prima
    pInv = modinv(q, p)
  File "RSA-2prima.py", line 25, in modinv
    g, x, y = egcd(a, m)
  File "RSA-2prima.py", line 12, in egcd
    g, y, x = egcd(b%a, a)
  File "RSA-2prima.py", line 12, in egcd
    g, y, x = egcd(b%a, a)
  File "RSA-2prima.py", line 12, in egcd
    g, y, x = egcd(b%a, a)
  [Previous line repeated 993 more times]
  File "RSA-2prima.py", line 10, in egcd
    if a == 0:
RecursionError: maximum recursion depth exceeded in comparison

D:\LatPython>
```

FIGURE 1. The recursion error message in Python 3 programming

The appeared error is caused by a condition when the maximum limit iteration in the algorithm is exceeded. It can be known that the default programming limits the iteration not more than 993 times. If the limit is exceeded the value, iteration will be stopped, and all programming processes also will be stopped. The multi-prime RSA, combined with the CRT method, has been proposed to solve the problem. The steps involved in key generation, encryption, and decryption to implement multi-prime numbers of modified RSA are as described in Figure 2 and Figure 3. Our modified RSA generates a cryptosystem base



Generate random prime number:

RSA-2primes: p, q

RSA-4primes: p, q, r, s

RSA-8primes: p, q, r, s, t, u, v, w

RSA-16primes: $p, q, r, s, t, u, v, w, a, b, c, d1, e1, f, g, h$

RSA-32primes: $p, q, r, s, t, u, v, w, a, b, c, d1, e2, f, g, h, p1, q1, r1, s1, t1, u1, v1, w1, a1, b1, c1, d2, e2, f1,$

Calculate module n:

RSA-2primes: $n = p\,q$

RSA-4primes: $n = p\,q\,r\,s$

RSA-8primes: $n = p\,q\,r\,s\,t\,u\,v\,w$

RSA-16primes: $n = p\,q\,r\,s\,t\,u\,v\,w\,a\,b\,c\,d1\,e1\,f\,g\,h$

RSA-32primes: $n = p\,q\,r\,s\,t\,u\,v\,w\,a\,b\,c\,d1\,e2\,f\,g\,h\,p1\,q1\,r1\,s1\,t1\,u1\,v1\,w1\,a1\,b1\,c1\,d2\,e2\,f1\,g_1$

Calculate Euler totient $\phi(\boldsymbol{n})$:

RSA-2primes: $\phi(n) = (p-1)\,(q-1)$

RSA-4primes: $\phi(n) = (p-1)\,(q-1)\,(r-1)\,(s-1)$

RSA-8primes: $\phi(n) = (p-1)\,(q-1)\,(r-1)\,(s-1)\,(t-1)\,(u-1)\,(v-1)\,(w-1)$

RSA-16primes: $\phi(n) = (p-1)\,(q-1)\,(r-1)\,(s-1)\,(t-1)\,(u-1)\,(v-1)\,(w-1)\,(a-1)\,(b-1)\,(c-1)\,(d_1-1)\,(e_1-1)\,(f-1)\,(g-1)\,(h-1)$

RSA-32primes: $\phi(n) = (p-1)\,(q-1)\,(r-1)\,(s-1)\,(t-1)\,(u-1)\,(v-1)\,(w-1)\,(a-1)\,(b-1)\,(c-1)\,(d_1-1)\,(e_1-1)\,(f-1)\,(g-1)\,(h-1)\,(p_1-1)\,(q_1-1)\,(r_1-1)\,(s_1-1)\,(t_1-1)\,(u_1-1)\,(v_1-1)\,(w_1-1)\,(a_1-1)\,(b_1-1)\,(c_1-1)\,(d_2-1)\,(e_2-1)\,(f_1-1)$

Establish public key, then determine the private key:

- select integer **e = 65537** which is the encryption (public) key such that: GCD (Ø(n), e) = 1 ; 1 < e < Ø(n);
- determine d is the decryption (private) key such that d = e^{-1} mod Ø(n);

Calculate Euler CRT components:

RSA-2primes: dp = d%(p-1), dq = d% (q-1), pInv = modinv(q, p), qInv = modinv(p, q).

RSA-4primes: dp = d%(p-1), dq=d% (q-1), dr=d% (r-1), d=d% (s-1),
pInv=modinv(n/p, p), qInv=modinv(n/q, q), rInv=modinv(n/r, r), sInv=modin(n/s, s).

RSA-8primes: dp = d%(p-1), dq=d% (q-1), dr=d% (r-1), ds=d% (s-1), dt=d% (t-1), du=d% (u-1), dv=d% (v-1), dw=d% (w-1)
pInv=modinv(n/p, p), qInv=modinv(n/q, q), rInv=modinv(n/r, r), sInv=modin(n/s, s), tInv=modinv(n/t, t), uInv=modinv(n/u, u), vInv=modinv(n/v, v), wInv=modin(n/s, s).

RSA-16primes: dp = d%(p-1), dq=d% (q-1), dr=d% (r-1), ds=d% (s-1), dt=d% (t-1), du=d% (u-1), dv=d% (v-1), dw=d% (w-1) da=d%(a-1), db=d% (b-1), dc=d% (c-1), dd=d% (d_1 -1), de=d% (e_1 -1), df=d% (f-1), dg=d% (g-1), dh=d% (h-1)
pInv=modinv(n/p, p), qInv=modinv(n/q, q), rInv=modinv(n/r, r), sInv=modin(n/s, s), tInv=modinv(n/t, t),
uInv=modinv(n/u, u), vInv=modinv(n/v, v), wInv=modin(n/w, w), aInv=modinv(n/a, a), bInv=modinv(n/b, b),
cInv=modinv(n/c, c), dInv=modin(n/d_1, d_1), eInv=modinv(n/e_1, e_1), fInv=modinv(n/f, f), gInv=modinv(n/g, g),
hInv=modin(n/h, h)

RSA-32primes: dp = d%(p-1), dq=d% (q-1), dr=d% (r-1), ds=d% (s-1), dt=d% (t-1), du=d% (u-1),
dv=d% (v-1), dw=d% (w-1), da=d%(a-1), db=d% (b-1), dc=d% (c-1), dd=d% (d_1 -1), de=d% (e_1 -1),
df=d% (f-1), dg=d% (g-1), dh=d% (h-1), dp_1=d%(p_1 -1), dq_1=d% (q_1 -1), dr_1=d% (r_1 -1), ds_1=d% (s_1 -1),
dt_1=d% (t_1 -1), du_1=d% (u_1 -1), dv_1=d% (v_1 -1), dw_1=d% (w_1 -1), da_1=d% (a_1 -1), db_1=d% (b_1 -1), dc_1=d% (c_1 -1),
dd_1=d% (d_2 -1), de_1=d% (e_2 -1), df_1=d% (f_1 -1), dg_1=d% (g_1 -1),
pInv=modinv(n/p, p), qInv=modinv(n/q, q), rInv=modinv(n/r, r), sInv=modin(n/s, s), tInv=modinv(n/t, t),
uInv=modinv(n/u, u), vInv=modinv(n/v, v), wInv=modin(n/w, w), aInv=modinv(n/a, a), bInv=modinv(n/b, b),
cInv=modinv(n/c, c), dInv=modin(n/d_1, d_1), eInv=modinv(n/e_1, e_1), fInv=modinv(n/f, f), gInv=modinv(n/g, g),
hInv=modin(n/h, h)
p_1Inv=modinv(n/p_1, p_1), q_1Inv=modinv(n/q_1, q_1), r_1Inv=modinv(n/r_1, r_1), s_1Inv=modin(n/s_1, s_1), t_1Inv=modinv(n/t_1, t_1),
u_1Inv=modinv(n/u_1, u_1), v_1Inv=modinv(n/v_1, v_1), w_1Inv=modin(n/w_1, w_1), a_1Inv=modinv(n/a_1, a_1), b1Inv=modinv(n/b1, b1),
c_1Inv=modinv(n/c_1, c_1), d_1Inv=modin(n/d_2, d_2), e_1Inv=modinv(n/e_2, e_2), f_1Inv=modinv(n/f_1, f_1), g_1Inv=modinv(n/g_1, g_1),
h_1Inv=modin(n/h_1, h_1)

FIGURE 2. The proposed method: Key generation algorithm of modified RSA with multi prime numbers

Encryption process:
$C = M^e \bmod n$

Decryption process:

RSA-2primes: $m_1 = (C^{dp} \bmod p)*q*pInv$; $m_2= (C^{dq} \bmod q)*p*qInv$; $M= (m_1+m_2)\%n$

RSA-4primes: $m_1 = (C^{dp} \bmod p)*qrs*pInv$; $m_2= (C^{dq} \bmod q)*prs*qInv$;
$m_3 = (C^{dr} \bmod r)*pqs*rInv$; $m_4= (C^{ds} \bmod s)*pqr*sInv$;
$M= (m_1+m_2+m_3+m_4)\%n$

RSA-8primes: $m_1 = (C^{dp} \bmod p)*qrstuvw*pInv$; $m_2= (C^{dq} \bmod q)*prstuvw*qInv$; $m_3 = (C^{dr} \bmod r)*pqstuvw*rInv$;
$m_4= (C^{ds} \bmod s)*pqrtuvw*sInv$; $m_5 = (C^{dt} \bmod t)*pqrsuvw*tInv$; $m_6= (C^{du} \bmod u)*pqrstvw*uInv$;
$m_7 = (C^{dv} \bmod v)*pqrstuw*vInv$; $m_8= (C^{dw} \bmod w)*pqrstuv*wInv$;
$M= (m_1+m_2+m_3+m_4+m_5+m_6+m_7+m_8)\%n$

RSA-16primes: $m_1 = (C^{dp} \bmod p)*qrstuvwabcd_1e_1fgh*pInv$; $m_2 = (C^{dq} \bmod q)*prstuvwabcd_1e_1fgh*qInv$;
$m_3 = (C^{dr} \bmod r)*pqstuvwabcd_1e_1fgh*rInv$; $m_4 = (C^{ds} \bmod s)*pqrtuvwabcd_1e_1fgh*sInv$;
$m_5 = (C^{dt} \bmod t)*pqrsuvwabcd_1e_1fgh*tInv$; $m_6 = (C^{du} \bmod u)*pqrstvwabcd_1e_1fgh*uInv$;
$m_7 = (C^{dv} \bmod v)*pqrstuwabcd_1e_1fgh*vInv$; $m_8 = (C^{dw} \bmod w)*pqrstuvabcd_1e_1fgh*wInv$;
$m_9 = (C^{da} \bmod a)*pqrstuvwbcd_1e_1fgh*aInv$; $m_{10} = (C^{db} \bmod b)*pqrstuvwacd_1e_1fgh*bInv$;
$m_{11} = (C^{dc} \bmod c)*pqrstuvwabd_1e_1fgh*cInv$; $m_{12} = (C^{dd} \bmod d1)*pqrstuvwabce_1fgh*dInv$;
$m_{13} = (C^{de} \bmod e1)*pqrstuvwabcd_1fgh*eInv$; $m_{14} = (C^{df} \bmod f)*pqrstuvwabcd_1e_1gh*fInv$;
$m_{15} = (C^{dg} \bmod g)*pqrstuvwabcd_1e_1fh*gInv$; $m_{16} = (C^{dh} \bmod h)*pqrstuvwabcd_1e_1fg*hInv$;
$M= (m_1+m_2+m_3+m_4+m_5+m_6+m_7+m_8+ m_9+m_{10}+m_{11}+m_{12}+m_{13}+m_{14}+m_{15}+m_{16})\%n$

RSA-32primes: $m_1 = (C^{dp} \bmod p)*qrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*pInv$;
$m_2 = (C^{dq} \bmod q)*prstuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*qInv$;
$m_3 = (C^{dr} \bmod r)*pqstuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*rInv$;
$m_4 = (C^{ds} \bmod s)*pqrtuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*sInv$;
$m_5 = (C^{dt} \bmod t)*pqrsuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*tInv$;
$m_6 = (C^{du} \bmod u)*pqrstvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*uInv$;
$m_7 = (C^{dv} \bmod v)*pqrstuwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*vInv$;
$m_8 = (C^{dw} \bmod w)*pqrstuvabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*wInv$;
$m_9 = (C^{da} \bmod a)*pqrstuvwbcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*aInv$;
$m_{10} = (C^{db} \bmod b)*pqrstuvwacd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*bInv$;
$m_{11} = (C^{dc} \bmod c)*pqrstuvwabd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*cInv$;
$m_{12} = (C^{dd} \bmod d_1)*pqrstuvwabce_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*dInv$;
$m_{13} = (C^{de} \bmod e_1)*pqrstuvwabcd_1fgh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*eInv$;
$m_{14} = (C^{df} \bmod f)*pqrstuvwabcd_1e_1gh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*fInv$;
$m_{15} = (C^{dg} \bmod g)*pqrstuvwabcd_1e_1fh\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*gInv$;
$m_{16} = (C^{dh} \bmod h)*pqrstuvwabcd_1e_1fg\ p_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*hInv$;
$m_1 = (C^{dp1} \bmod p_1)*pqrstuvwabcd_1e_1fghq_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*p_1Inv$;
$m_2 = (C^{dq1} \bmod q_1)*pqrstuvwabcd_1e_1fghp_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*q_1Inv$;
$m_3 = (C^{dr1} \bmod r_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*r_1Inv$;
$m_4 = (C^{ds1} \bmod s_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1r_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*s_1Inv$;
$m_5 = (C^{dt1} \bmod t_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1r_1s_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*t_1Inv$;
$m_6 = (C^{du1} \bmod u_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1v_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*u_1Inv$;
$m_7 = (C^{dv1} \bmod v_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1w_1a_1b_1c_1d_2e_2f_1g_1h_1*v_1Inv$;
$m_8 = (C^{dw1} \bmod w_1)*pqrstuvwabcd_1e_1fgh\ p_1q_1r_1s_1t_1u_1v_1a_1b_1c_1d_2e_2f_1g_1h_1*w_1Inv$;
$m_9 = (C^{da1} \bmod a_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1b_1c_1d_2e_2f_1g_1h_1*a_1Inv$;
$m_{10} = (C^{db1} \bmod b_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1c_1d_2e_2f_1g_1h_1*b_1Inv$;
$m_{11} = (C^{dc1} \bmod c_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1d_2e_2f_1g_1h_1*c_1Inv$;
$m_{12} = (C^{dd1} \bmod d_2)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1e_2f_1g_1h_1*d_1Inv$;
$m_{13} = (C^{de1} \bmod e_2)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2f_1g_1h_1*e_1Inv$;
$m_{14} = (C^{df1} \bmod f_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2g_1h_1*f_1Inv$;
$m_{15} = (C^{dg1} \bmod g_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1h_1*g_1Inv$;
$m_{16} = (C^{dh1} \bmod h_1)*pqrstuvwabcd_1e_1fghp_1q_1r_1s_1t_1u_1v_1w_1a_1b_1c_1d_2e_2f_1g_1*h_1Inv$;
$M= (m_1+m_2+m_3+m_4+m_5+m_6+m_7+m_8+m_9+m_{10}+m_{11}+m_{12}+m_{13}+m_{14}+m_{15}+m_{16}$
$+m_{17}+m_{18}+m_{19}+m_{20}+m_{21}+m_{22}+m_{23}+m_{24}+m_{25}+m_{26}+m_{27}+m_{28}+m_{29}+m_{30}+m_{31}+m_{32})\%n$

FIGURE 3. The proposed method: Encryption and decryption algorithm of modified RSA with multi prime numbers

The modified RSA algorithm as described above has been developed and ran in Python 3 programming with key sizes bits variated in 800, 1024, 1600, 2048, 4096, 5000, 6000, 7000, 8192, 1200, 16384, 20000, 24000, 28000, 32768, 48000, 50240. Application program ran in computer with Intel®Core ^{TM}i3-4030U CPU@1.90 GHz processor, 8.00 GB installed memory (RAM), 64-bit Operating System, x64-based processor, and in Windows 10 Pro operating system.

RESULT AND DISCUSSION

The standard RSA basically generates a cryptosystem based on two prime numbers (p and q). As a basic condition, standard RSA combined with CRT method was run in our Python program. Figure 4 describes performance of key generation the algorithm.

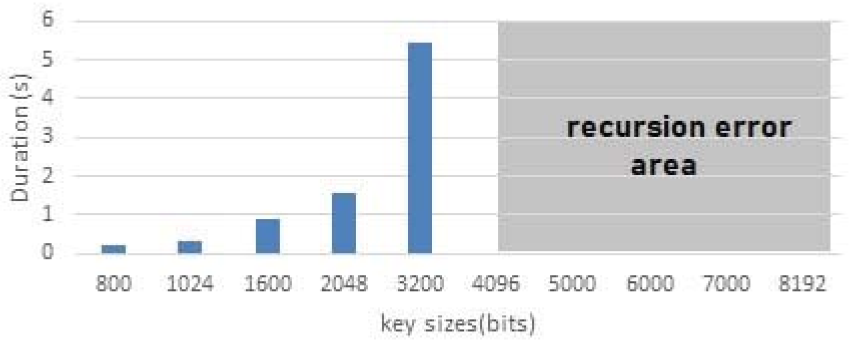


FIGURE 4. Duration of standard RSA key generation

The time duration of the key generation increases with longer key sizes increasing. Its algorithm works perfectly until 3200 bits of key size. For more 3200 bits, recursion error was happened and stopped the Python program. The bits key sizes of 4096, 5000, 6000, 7000, and 8192 became a recursion error area. By implementing modified RSA based on 2, 4, and 8, the longer key sizes in the key generation process can be achieved without recursion error, as performed in Figure 5.

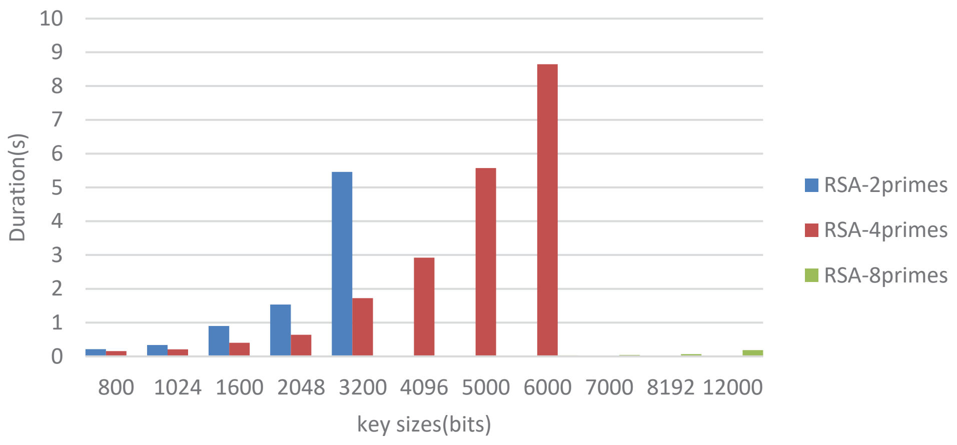


FIGURE 5. Duration of modified RSA key generation based on two compared with four and eight prime numbers

From Figure 5, we know that by expanding the number of used prime numbers, the algorithm also can be expanded from 3200 bits to 6000 bits based on four prime numbers and to 12000 bits based on eight prime numbers. The proposed method is perfect for solving the recursion error problem. Besides hints recursion error effectively, by expanding the number of used prime numbers, the method also can decrease duration time for generating a pair of keys. A pair of key generation will be faster when more multi-prime numbers are used. Based on Figure 5, the modified RSA algorithm based on eight prime numbers is the fastest. Their duration in all key sizes is relatively very small compared to others. We almost could not see them.

The same condition is true when the use of prime numbers continues to increase by 16 and 32 prime numbers that expand key size to 24000 and 50240 bits. Figure 6 shows that phenomena.

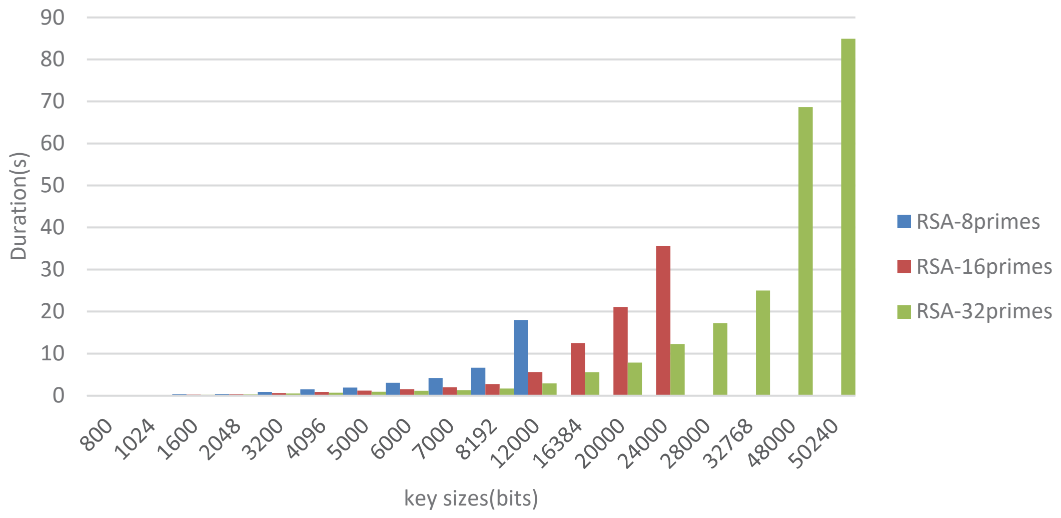


FIGURE 6. Duration of modified RSA key generation based on 8, 16, and 32 prime numbers

Table 1 shows the speed comparison of each pair of key generation processes based on the number of prime numbers used in our modified RSA algorithm. Based on Table 1, the speed increasing of key generation processes can reach 1.4 – 3.2 in using four prime numbers, 1.7 – 6.2 in using eight prime numbers, 1.9 – 9 in using sixteen prime numbers, and 2.8 – 10.5 using thirty-two prime numbers.

TABLE 1. Speed comparison of key generation process

Key sizes	RSA Algorithm				
(bits)	RSA-2primes	RSA-4primes	RSA-8primes	RSA-16primes	RSA-32primes
800	1	1,35871	1,71434	1,941291	2,810327
1024	1	1,626022	2,087734	2,708715	2,268515
1600	1	2,233209	2,711971	3,738396	4,423592
2048	1	2,39479	3,782583	5,097537	5,928417
3200	1	3,166595	6,186505	8,99737	10,46741

The second phase of the RSA algorithm is the encryption process. It is a process to convert a plain text to a ciphertext conducted by the sender. This experiment sets a fixed public key as 65537. Its public key is recommended as the maximum number to ensure that the encryption process can run fast [1][12]. Figure 7 shows the duration of encryption processes based on key sizes.

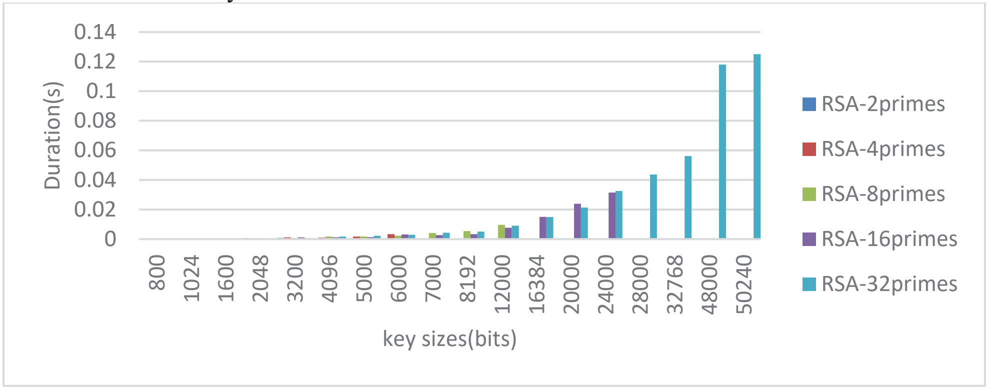


FIGURE 7. Duration of modified RSA encryption based on 2, 4, 8, 16, and 32 prime numbers

In principle, the designed encryption algorithm should be simple, easy, and fast to produce adequate complex ciphertext. On another side, the decryption algorithm should be designed and conducted so that an algorithm is complex, uneasy, and needs more time execution and computational resources. This is an implementation of the trapdoor principle [12]. It is implemented to hint attacks, such as factorization or common modulus attack. By implementing the same and fixed public key sizes, the time duration to execute the encryption algorithm will always be the same for the same key sizes.

The third phase of the RSA algorithm is the decryption process. It is a process to get an original message from ciphertext conducted by the receiver. Only an authorized receiver has a private key to decipher a ciphertext correctly. The decryption process needs a long duration time caused by the big value of the private key compared to the public key.

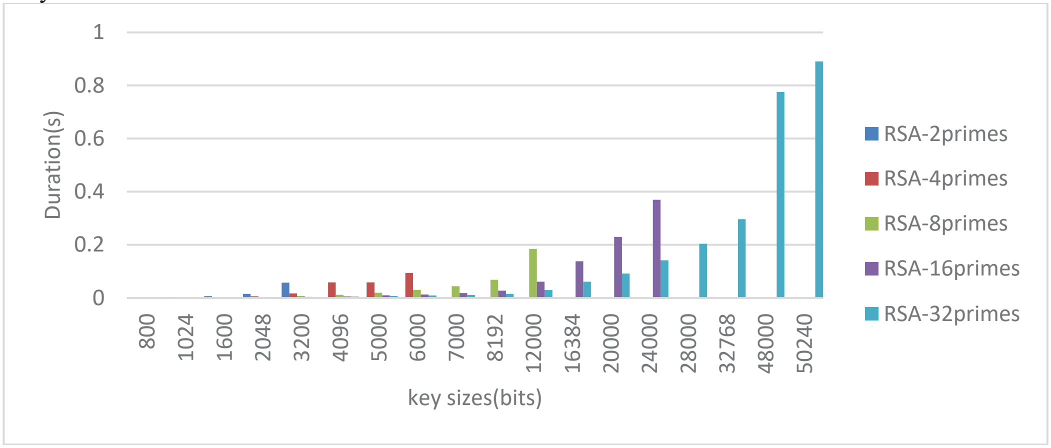


FIGURE 8. Duration of modified RSA decryption based on 2, 4, 8, 16, and 32 prime numbers

The implementation of multi-prime numbers significantly affects the decryption speed. By increasing the number of used prime numbers, decryption speed also significantly increases. The application of the CRT method increasingly magnifies the increase of decryption speed. Table 2 shows the decryption speed comparison of decryption processes' duration based on the number of prime numbers used in our modified RSA algorithm.

TABLE 2. Speed comparison of decryption process

Key sizes (bits)	RSA Algorithm RSA-2primes	RSA-4primes	RSA-8primes	RSA-16primes	RSA-32primes
800	1	2	2.4992	-	4.990415
1024	1	4.995736	4.995736	1.8744	3.7488
1600	1	2.99872	5.6232	3.7488	9
2048	1	2.461602	6.856033	6.402475	5.645841
3200	1	3.384981	7.851423	14.19675	21.69853

Based on Table 2, the decryption process's speed increasing can reach 2 – 5 in using four prime numbers, 2.5 – 7.9 in using eight prime numbers, 1.8 – 14.2 in using sixteen prime numbers, and 3.7 – 21.7 in using thirty-two prime numbers.

Back to the main issues on implementing multi-prime numbers to expand longer key sizes of RSA combined CRT algorithm, we can plot the achieved maximum key size in implementing 2, 4, 8, 16, and 32 prime numbers as described in Figure 9. Based on the data, we can see that by using four prime number can expand key size until 6320 bits, until 12640 bits by eight prime numbers, until 25120 bits by sixteen prime numbers, and until 50240 by thirty-two prime numbers. It can be plotted correlation between some implemented prime numbers versus the maximum key size for each algorithm.

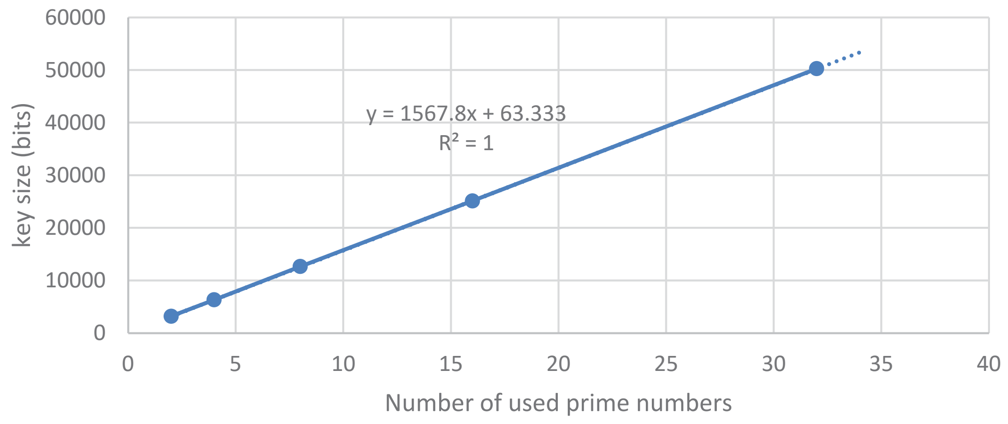


FIGURE 9. The correlation of used multi prime numbers and key size

The correlation equation $y = 1567.8x + 63.333$ represents that condition, which x represents used number of multi-prime numbers, and y represents the maximum key size. By increasing the number of used prime numbers, longer maximum key size also will be increase.

CONCLUSION

The implementation of standard RSA algorithms in Python 3 language programming has recursion error in the execution of inverse modulus calculation. A modified RSA that implements multi-prime numbers combined with the CRT method can solve the problem. Increasing the number of used prime numbers expand executed longer key size. Combining the CRT method and RSA algorithm based on four prime numbers can be applied without recursion error up to a key length of 6280 bits, with eight primes up to 12,560 bits, with sixteen primes up to 25,120 bits, and with thirty-two primes up to 50,240 bits. Implementation of more multi-prime numbers can expand to combine the CRT method and RSA Algorithm for longer key sizes by approximation relation $y = 1567{,}8x + 63{,}333$, which y is the key size and x is the number of primes numbers. Utilization of multi-prime numbers also increases key generation speed significantly, and on another side combination of the CRT, the method increases decryption speed. Expand of key size and speed increasing in key generation, and decryption process increases RSA algorithm performance significantly for this experiment.

REFERENCES

1. W. Stallings, *Cryptography and Network Security*, *Seventh Ed*., edited by M.J. Horton, T. Johnson, K. Alaura and A. Baroi (Pearson Prentice Hall, Singapore, 2017), p. 767. Available from: www.pearsonglobaleditions.com.
2. C.P. Robinson, *BSU Honors Program Theses and Projects*, (268), (2018). Available from: http://vc.bridgew.edu/honors_proj/268.
3. S.S. Al-kaabi and S.B. Belhaouari, *J Comput Inf Technol (CS IT)*, pp. 123–142 (2019).
4. R.S. Dhakar, A.G. Kumar and P. Sharma, in *Proceding of 2012 Second International Conference on Advanced Computing & Communication Technologies* (IEEE, 2012), pp. 2–5.
5. M.R. Patidar, R. Bhartiya, in *Proceeding of 2013 IEEE International Conference on Computational Intelligence and Computing Research* (IEEE, 2013).
6. R.S. Abdeldaym, H.M.A. Elkader and R. Hussein, *Int J Electron Eng.*, **10** (1), pp. 51–64 (2019).
7. "Python 3 Tutorials," Point Tutorials Point (I) Pvt. Ltd.; 2016. Available from: www.tutorialspoint.com
8. D. Mahto, D.A. Khan and D.K. Yadav, in *Proceeding of the World Congress on Engineering 2016* (London, 2016), pp. 419-422. Available from: http://www.iaeng.org/publication/WCE2016/WCE2016_pp419-422.pdf.
9. S.S. Tint, *Comput Appl An Int J.*, **1** (2), pp. 27–35 (2014).
10. L.K. Galla, V.S. Koganti and N. Nuthalapati, in 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) (IEEE, 2016), pp. 81–87.

11. R. Wardoyo, E. Setyaningsih and A.K. Sari, in 2018 Third International Conference on Informatics and Computing (ICIC) (IEEE, 2018), pp. 1–9.
12. A. Kak, *Lecture 12: Public-Key Cryptography and the RSA Algorithm* (Purdue University, 2020).