Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG 4136 Computer Architecture III (3 units)
Fall 2021

Laboratory #3

Demonstration is due: Monday 22 or Tuesday 24 depending on your lab period
Report is due: Wednesday, November 24, 2021

Goals

The lab's goal is to apply parallel programming to cryptography using the Intel® oneAPI DPC++/C++ platform.

Details

In this lab, first you will design and implement a client-server application that allows you to exchange an encrypted file. Second, it is requested to develop a module that allows decryption of a file without knowing the key using parallel programming and especially the Intel® oneAPI DPC++/C++ platform.

In the field of cryptography, there are two main families: symmetric cryptography (or secret key cryptography) and asymmetric cryptography (also known as public key cryptography). Symmetric encryption, also referred to as conventional encryption, secret-key, or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the late 1970s [1] [2]. It remains by far the most widely used of the two types of encryptions. Public key cryptography, on the other hand, is based on another concept involving a pair of keys: one for encryption and the other for decryption. In this lab, we will use public key cryptography.

The RSA public-key encryption algorithm

One of the first public-key schemes was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978 [1] [2]. The RSA scheme has since that time reigned supreme as the most widely accepted and implemented approach to public-key encryption. RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and n - 1 for some n.

Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$C = M^e \bmod n$
$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$

Both sender and receiver must know the values of n and e, and only the receiver knows the value of d. This is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$.
2. It is relatively easy to calculate M^e and Cd for all values of $M < n$.
3. It is infeasible to determine d given e and n.

The first two requirements are easily met. The third requirement can be met for large values of e and n.

More should be said about the first requirement. We need to find a relationship of the form $M^{ed} \bmod n = M$

The preceding relationship holds if e and d are multiplicative inverses modulo $\varphi(n)$, where $\varphi(n)$ is the Euler totient function. For p, q prime, $\varphi(pq) = (p - 1)(q -1)$. $\varphi(n)$, referred to as the Euler totient of n, is the number of positive integers less than n and relatively prime to n. The relationship between e and d can be expressed as:

$$ed \bmod \varphi(n) = 1$$

This is equivalent to saying

$$ed \bmod \varphi(n) = 1$$
$$d \bmod \varphi(n) = e^{-1}$$

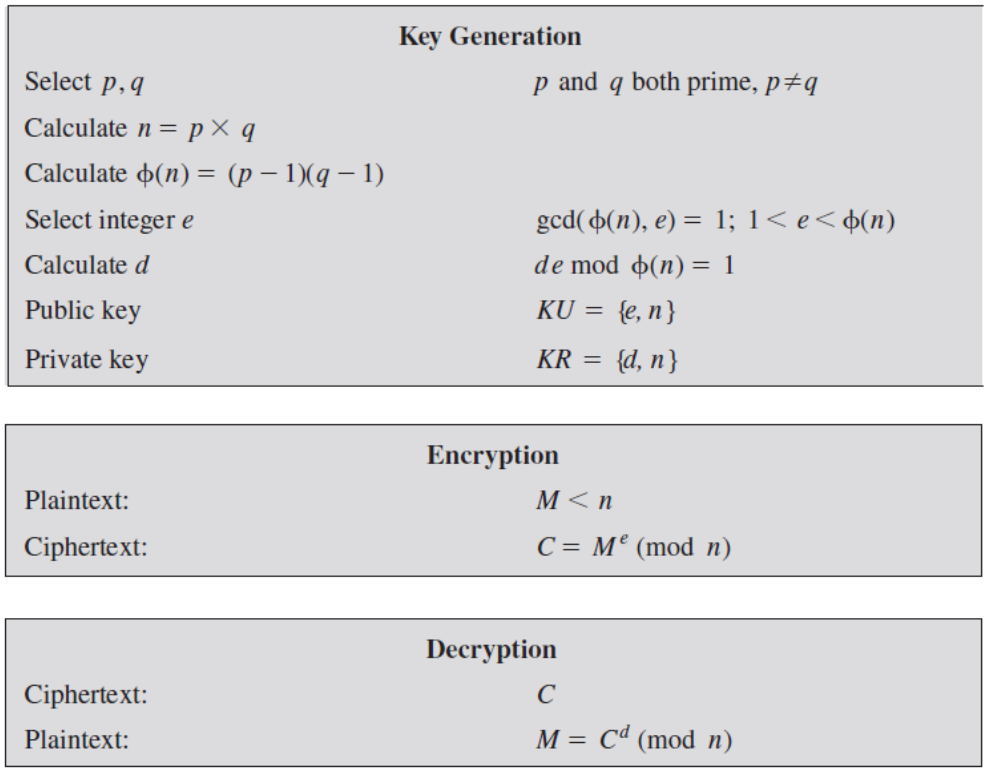
That is, e and d are multiplicative inverses mod $\varphi(n)$. According to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to $\varphi(n)$. Equivalently, $\gcd(\varphi(n),d) = 1$; that is, the greatest common divisor of $\varphi(n)$ and d is 1.

Figure 1 summarizes the RSA algorithm. Begin by selecting two prime numbers, p, and q, and calculating their product n, which is the modulus for encryption and decryption. Next, we need the quantity $\varphi(n)$. Then select an integer e that is relatively prime to $\varphi(n)$ [i.e., the greatest common divisor of e and $\varphi(n)$ is 1]. Finally, calculate d as the multiplicative inverse of e, modulo $\varphi(n)$. It can be shown that d and e have the desired properties.

Suppose user A has published its public key and user B wishes to send the message M to A. Then B calculates $C = M^e \pmod n$ and transmits C. On receipt of this ciphertext, user A decrypts by calculating $M = C^d \pmod n$.

An example, from [4], is shown in Figure 2. For this example, the keys were generated as follows [1]:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \text{ x } 11 = 187$.
3. Calculate $\varphi(n) = (p - 1)(q - 1) = 16 \text{ x } 10 = 160$.
4. Select e such that e is relatively prime to $\varphi(n) = 160$ and less than $\varphi(n)$; we choose $e = 7$.
5. Determine d such that $de \bmod 160 = 1$ and $d < 160$. The correct value is $d = 23$, because $23 \text{ x } 7 = 161 = (1 \text{ x } 160) + 1$.



Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 1: The RSA Algorithm [1]

The resulting keys are public key *PU* = 57, 1876 and private key *PR* = {23, 187}. The example shows the use of these keys for a plaintext input of *M* = 88. For encryption, we need to calculate *C* = 88^7 mod 187. Exploiting the properties of modular arithmetic, we can do this as follows:

88^7 mod 187 = [(88^4 mod 187) x (88^2 mod 187) x (88^1 mod 187)] mod 187
88^1 mod 187 = 88
88^2 mod 187 = 7744 mod 187 = 77
88^4 mod 187 = 59,969,536 mod 187 = 132
88^7 mod 187 = (88 x 77 x 132) mod 187 = 894,432 mod 187 = 11

For decryption, we calculate *M* = 11^{23} mod 187:

11^{23} mod 187 = [(11^1 mod 187) x (11^2 mod 187) x (11^4 mod 187) x (11^8 mod 187) x (11^8 mod 187)] mod 187
11^1 mod 187 = 11
11^2 mod 187 = 121
11^4 mod 187 = 14,641 mod 187 = 55
11^8 mod 187 = 214,358,881 mod 187 = 33
11^{23} mod 187 = (11 x 121 x 55 x 33 x 33) mod 187 = 79, 720, 245
mod 187 = 88

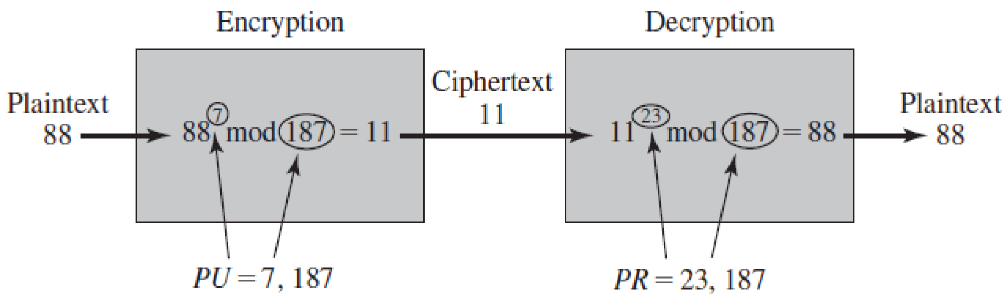


Figure 2: Example of RSA Algorithm [1]

The Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows [1]:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

The defense against the brute force approach is the same for RSA as for other cryptosystems; namely, use a large key space. Thus, the larger the number of bits in *d*, the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

The Factoring Problem

We can identify three approaches to attacking RSA mathematically [1]:

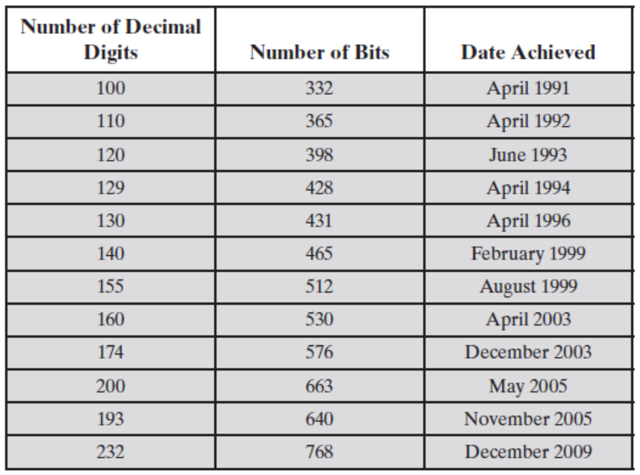
- Factor *n* into its two prime factors. This enables calculation of $\varphi(n) = (p - 1)(q - 1)$, which, in turn, enables determination of $d \equiv e^{-1} (\bmod\ \varphi(n))$.
- Determine $\varphi(n)$ directly, without first determining *p* and *q*. Again, this enables determination of $d \equiv e^{-1} (\bmod\ \varphi(n))$.
- Determine *d* directly, without first determining $\varphi(n)$.

Most discussions of the cryptanalysis of RSA have focused on the task of factoring *n* into its two prime factors. Determining $\varphi(n)$ given *n* is equivalent to factoring *n* [5]. With presently known algorithms, determining *d* given *e* and *n* appears to be at least as time consuming as the factoring problem. Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA.

With presently known algorithms, determining *d* given *e* and *n* appears to be at least as time consuming as the factoring problem. Hence, we can use factoring performance as a benchmark against which to evaluate the security of RSA.

For a large *n* with large prime factors, factoring is a hard problem, but not as hard as it used to be. Just as it had done for DES, RSA Laboratories issued challenges for the RSA cipher with key sizes of 100, 110, 120, and so on, digits. The latest challenge to be met is the RSA-768 challenge with a key length of 232 decimal digits, or 768 bits. Table 1[1] shows the results to date.

Table 1: Progress in Factorization



Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Tasks

In this project, you will implement a client-server system where two machines will exchange an encrypted file. As shown in figure 3, a third machine (machine 2) will intercept the file and try all possible private keys to decrypt the file using the power of parallel programming.

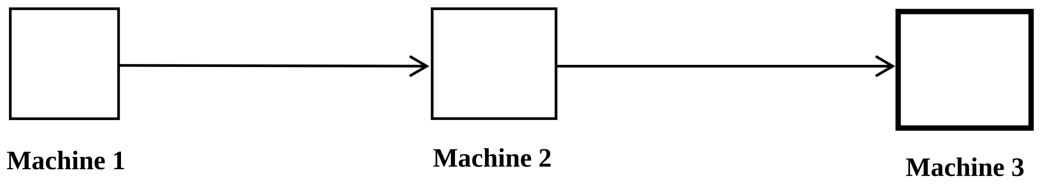


Figure 3: Communication scheme

To do this, machines 1 and 3 choose two distinct primes p and q (in practice, these are very large numbers, up to hundreds of digits). In this academic project, you can use prime numbers that are between 2 and 2^{32}. Machine 2 must decrypt the intercepted file without knowing the private key by trying the prime numbers that are between 2 and 2^{32}.

Deliverables

a. Brief description of the purpose and theory of the problem.
b. Brief explanation of your solution algorithm.
c. Amdahl's law defines the speed gain that can be obtained by using the GPU, multiple processors, or other execution accelerators. What is the speed gain compared to a single processor?
d. Design document using UML diagrams.
e. Screenshots of the demonstration for each application
f. Discussion
g. Conclusion

Evaluation

- The proper functioning of the executable code: 40%.
- The report: 60%.

References

1. Stallings, William, Brown, Lawrie, Computer Security: Principles and Practice, Fourth Global edition, 2018
2. Public-key encryption was first described in the open literature in 1976; the US National Security Agency (NSA) and the (then) UK CESG claim to have discovered it some years earlier.
3. Rivest, R.; Shamir, A.; and Adleman, L. “A Method for Obtaining Digital Signatures and Public Key Cryptosystems.” Communications of the ACM, February 1978.
4. Singh, S. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. New York: Anchor Books, 1999.
5. Ribenboim, P. The New Book of Prime Number Records. New York: Springer-Verlag, 1996.