Testbed Implementation of Cryptographic Operations using Standard Libraries

Submitted by: **Hardik Thami, Kanhaiya Kumar, Shubham Kumar**

Submitted to: **Prof. Manoj Mishra (Supervisor)**

Enrollment no's: **19114035, 19114042, 19114081**

Department of Computer Science and Engineering

Lab Based Project Endterm Report

May 4, 2022

Contents

1 Problem statement

To analyze the performance of the cryptographic operations using the standard libraries such as MIRACL, PBC Library, and Crypto++.

2 Introduction

All these libraries are used to compute the cost of cryptographic operations that are used in encryption or decryption like RSA, AES, and SHA (hashing algorithm only for encryption).

We were enthusiastic to measure the performance of some standard cryptographic algorithms using these standard libraries and to see how efficiently they perform.

While running the programs, we have measured the time taken in encrypting and decrypting the input data and we have mentioned the total time taken in the performance table.

The main significance of this project is that we were able to measure the performance difference of these algorithms against these three libraries.

For the SHA, we have checked 2 versions which are SHA-1 and SHA-3. The generated hash values were of 224, 256, 384, and 512 bits.

Input message for both SHA-1 and SHA-3 is "abc"
Input message for SHA-2 is "Yoda said, Do or do not. There is no try.".
Details about each library has been discussed separately.

3 Implementation

3.1 Practical Implementation using MIRACL library

We have checked the performance of this library by using it in some physical systems having different architecture like Linux based system (x86 architecture, 8GB Ram, i5-8250U CPU @1.60GHz×8), Raspberry Pi (1.5 GHz 64-bit quad core ARM Cortex-A72 processor, 8Gb Ram).

On both the system, we have run three encryptive algorithmic programs such as AES (Advanced Encryption Standard), RSA, and SHA (Secure Hash Algorithm) and We analysed the time taken by all the three programs while using this library to generate the required result.

And so in the below Table 1, We have mentioned the performance (i.e. Time taken) of this library.

Table 1: Computational time under Raspberry PI 4 and Linux based system for cryptographic-primitives using MIRACL

	RSA	**AES**	**SHA-1**	**SHA-3**
Linux based system	0.521s	0.171s	0.096s	0.107s
Raspberry Pi	4.686s	1.539s	0.864s	0.963s

To run the AES algorithm, the following commands can be used.

```
gcc mraes.c -o mraes  miracl.a -lm
./mraes
```

To run the SHA-1 algorithm, the following commands can be used.

```
gcc mrsha1.c -o mrsha1  miracl.a -lm
./mrsha1
```

To run the SHA-3 algorithm, the following commands can be used.

```
gcc mrsha3.c -o mrsha3 miracl.a -lm
./mrsha3
```

To run the RSA algorithm, the following commands can be used.

```
gcc rsa.c -o rsa p1363.c  miracl.a -lm
./rsa
```

3.2 Practical implementation using Crypto++ library

We have tested this library and measured the performance of this library by using it in our physical system like Linux based system (x86 architecture).

We ran 3 cryptographic algorithms such as AES (Advanced Encryption Standard), RSA, and SHA2 (Secure Hash Algorithm) and also analysed the time taken by all the 3 programs while using this library to generate the required result.

And so in the below Table 2, we have mentioned the time taken (run time + compile time) to run each of these algorithms with this library.

Table 2: Computational time under Linux based system for cryptographic-primitives using Crypto++

	RSA	**AES**	**SHA-2**
Linux (Ubuntu 18.04.5 LTS, 8GB RAM, i7-9750H CPU)	0.918s	0.78s	0.831s
Raspberry Pi (1.5 GHz 4-Core ARM Cortex-A72 CPU, 8GB RAM)	8.721s	7.41s	7.894s

AES Implementation in crypto++ is:

```
#include "cryptlib.h"
#include "rijndael.h"
#include "modes.h"
#include "files.h"
#include "osrng.h"
#include "hex.h"

#include <iostream>
#include <string>

int main(int argc, char* argv[])
{
    using namespace CryptoPP;

    AutoSeededRandomPool prng;
    HexEncoder encoder(new FileSink(std::cout));

    SecByteBlock key(AES::DEFAULT_KEYLENGTH);
    SecByteBlock iv(AES::BLOCKSIZE);

    prng.GenerateBlock(key, key.size());
    prng.GenerateBlock(iv, iv.size());

    std::string plain = "CBC Mode Test";
    std::string cipher, recovered;

    std::cout << "plain text: " << plain << std::endl;

    /*********************************\
    \*********************************/

    try
    {
        CBC_Mode< AES >::Encryption e;
        e.SetKeyWithIV(key, key.size(), iv);

        StringSource s(plain, true,
            new StreamTransformationFilter(e,
                new StringSink(cipher)
            ) // StreamTransformationFilter
        ); // StringSource
    }
    catch(const Exception& e)
    {
        std::cerr << e.what() << std::endl;
        exit(1);
    }

    /*********************************\
    \*********************************/

    std::cout << "key: ";
    encoder.Put(key, key.size());
    encoder.MessageEnd();
    std::cout << std::endl;

    std::cout << "iv: ";
    encoder.Put(iv, iv.size());
    encoder.MessageEnd();
    std::cout << std::endl;

    std::cout << "cipher text: ";
    encoder.Put((const byte*)&cipher[0], cipher.size());
    encoder.MessageEnd();
    std::cout << std::endl;

    /*********************************\
    \*********************************/

    try
    {
        CBC_Mode< AES >::Decryption d;
        d.SetKeyWithIV(key, key.size(), iv);

        StringSource s(cipher, true,
            new StreamTransformationFilter(d,
                new StringSink(recovered)
            ) // StreamTransformationFilter
        ); // StringSource

        std::cout << "recovered text: " << recovered << std::endl;
    }
    catch(const Exception& e)
    {
        std::cerr << e.what() << std::endl;
        exit(1);
    }

    return 0;
}
```

The bash script for testing time of AES in crypto++ is:

```
export TIMEFORMAT="Compile time: %3R seconds"
time g++ aes.cpp -o aes libcryptopp.a
echo -e "\nOutput:"
export TIMEFORMAT="Run time: %3R seconds"
time (./aes) | { cat; echo;}
```

RSA Implementation in crypto++ is:

```
#include <iostream>

#include <cryptopp/rsa.h>
#include <cryptopp/integer.h>
#include <cryptopp/osrng.h>

using namespace CryptoPP;

int main(){
    // Keys
  CryptoPP::Integer n("0xbeaadb3d839f3b5f"), e("0x11"), d("0x21a5ae37b9959db9");

  CryptoPP::RSA::PrivateKey privKey;
  privKey.Initialize(n, e, d);

  CryptoPP::RSA::PublicKey pubKey;
  pubKey.Initialize(n, e);

  // Encryption
  std::string message = "secret";
    std::cout << "plain text: " << message << std::endl;
  CryptoPP::Integer m((const byte *)message.data(), message.size());
  std::cout << "m: " << m << std::endl;
  // m: 126879297332596.

  CryptoPP::Integer c = pubKey.ApplyFunction(m);
  std::cout << "c: " << std::hex << c << std::endl;
  // c: 3f47c32e8e17e291h

  // Decryption
  CryptoPP::AutoSeededRandomPool prng;
  CryptoPP::Integer r = privKey.CalculateInverse(prng, c);
  std::cout << "r: " << std::hex << r << std::endl;

  // r: 736563726574h
  std::string recovered;
  recovered.resize(r.MinEncodedSize());

  r.Encode((byte *)recovered.data(), recovered.size());
  std::cout << "recovered: " << recovered << std::endl;

  // recovered: secret
  return 0;
}
```

The bash script for testing time of RSA in crypto++ is:

```
export TIMEFORMAT="Compile time: %3R seconds"
time g++ rsa_new.cpp -o rsa_new libcryptopp.a
echo -e "\nOutput:"
export TIMEFORMAT="Run time: %3R seconds"
time (./rsa_new) | { cat; echo;}
```

SHA-2 Implementation in crypto++ is:

```
#include "cryptlib.h"
#include "sha.h"
#include <iostream>
#include "files.h"
#include "hex.h"

int main (int argc, char* argv[])
{
    using namespace CryptoPP;
    HexEncoder encoder(new FileSink(std::cout));

    std::string msg = "Yoda said, Do or do not. There is no try.";
    std::string digest;

    SHA256 hash;
    hash.Update((const byte*)msg.data(), msg.size());
    digest.resize(hash.DigestSize());
    hash.Final((byte*)&digest[0]);

    std::cout << "Message: " << msg << std::endl;

    std::cout << "Digest: ";
    StringSource(digest, true, new Redirector(encoder));
    std::cout << std::endl;

    return 0;
}
```

The bash script for testing time of SHA-2 in crypto++ is:

```
export TIMEFORMAT="Compile time: %3R seconds"
time g++ sha2.cpp -o sha2 libcryptopp.a
echo -e "\nOutput:"
export TIMEFORMAT="Run time: %3R seconds"
time (./sha2) | { cat; echo;}
```

3.3 Practical Implementation using PBC library

We have tested this library and measured the performance of this library by using it in our physical systems having different architecture like Linux based system (x86 architecture), Raspberry Pi (ARM architecture) and Windows System (Windows 10).

On each system, We have run RSA algorithm and We analysed the time taken by the program while using this library to generate the required result.

Having done this, we have noted the average performance of RSA algorithm corresponding to each system which is shown below in the Table 3

Table 3: Computational time under different operating systems using PBC library

	RSA
Linux based system (Ubuntu 20.04 8 GB RAM,R5 processor)	1.03ms
Windows 10 (8 GB RAM,R5 processor)	1.06ms
Raspberry Pi (1.5 GHz 64-bit quad core ARM Cortex-A72 processor, 8Gb Ram)	4.143ms

The code that was tested

```

// name = filenamersa.c

#include <pbc/pbc.h>
#include <pbc/pbc_fp.h>
#include <pbc/pbc_test.h>

int main(void) {
  mpz_t p, q, N, d;
  mpz_t dmp1, dmq1;
  mpz_t ipmq, iqmp;
  mpz_t adq, adp;

  field_t f;
  element_t a, b;
  double t0, t1, tnaive = 0, tcrt=0;
  int i, n;

  mpz_init(p);
  mpz_init(q);
  mpz_init(N);
  mpz_init(d);
  mpz_init(dmp1);
  mpz_init(dmq1);
  mpz_init(ipmq);
  mpz_init(iqmp);
  mpz_init(adp);
  mpz_init(adq);
  pbc_mpz_randomb(p, 512);
  pbc_mpz_randomb(q, 512);
  mpz_nextprime(p, p);
  mpz_nextprime(q, q);
  mpz_mul(N, p, q);
  mpz_invert(ipmq, p, q);
  mpz_invert(iqmp, q, p);

  field_init_fp(f, N);
  element_init(a, f);
  element_init(b, f);
  n = 10;
  for (i=0; i<n; i++) {
    pbc_mpz_random(d, N);
    element_random(a);
    t0 = pbc_get_time();
    element_pow_mpz(b, a, d);
    t1 = pbc_get_time();
    tnaive += t1 - t0;

    mpz_sub_ui(p, p, 1);
    mpz_sub_ui(q, q, 1);

    mpz_mod(dmp1, d, p);
    mpz_mod(dmq1, d, q);

    mpz_add_ui(p, p, 1);
    mpz_add_ui(q, q, 1);

    element_to_mpz(adq, a);
    element_to_mpz(adp, a);

    t0 = pbc_get_time();
    mpz_powm(adp, adp, d, p);
    mpz_powm(adq, adq, d, q);

    // Garner's algorithm
    mpz_sub(adq, adq, adp);
    mpz_mul(adq, adq, ipmq);
    mpz_mod(adq, adq, q);
    mpz_mul(adq, adq, p);
    mpz_add(adp, adp, adq);

    t1 = pbc_get_time();
    tcrt += t1 - t0;
    element_set_mpz(b, adp);
  }
  printf("average RSA exp time = %lf\n", tnaive / n);
  printf("average RSA exp time (CRT) = %lf\n", tcrt / n);
  return 0;
}
```

The command for compiling and executing the above code

```
gcc filenamersa.c -o filenamersa -L. -lgmp -lpbc

./filenamersa
```

4 Achievables

Tested and measured performance of **RSA, AES, SHA encryption algorithms** on **PBC, MIRACL, Crypto++ libraries** by running it in our physical systems having different architectures like Linux based system (x86 architecture), Raspberry Pi (ARM architecture) and Windows System (Windows 10).

5 Future Work

1. The project can be further expanded upon by testing advanced encryption algorithms on different systems using different libraries

2. The performance criteria only takes time into account, hence there is room for improvement by considering memory usage as well.

6 References

1. MIRACL Github: https://github.com/miracl/MIRACL

2. MIRACL Website: https://miracl.com/

3. Crypto++ Github: https://github.com/weidai11/cryptopp

4. Crypto++ Website: https://www.cryptopp.com/

5. PBC Github: https://github.com/blynn/pbc

6. PBC Website: https://crypto.stanford.edu/pbc/