SYSC 4810: Introduction to Network and Software Security

Module 3 Assignment

Fall 2021

Dr. Mostafa Taha
Carleton University
Department of Systems and Computer Engineering

Posted: Friday November 4, 2022

Due on Sunday, November 13, 2022 by 11:59PM

This assignment contains 7 pages (including this cover page) and 2 Tasks, with 5 total problems. You are responsible for ensuring that your copy of the assignment is complete. Bring any discrepancy to the attention of your instructor.

Special Instructions:

1. Do as many problems as you can.
2. Start early as this assignment is much more time consuming than you might initially think!
3. The burden of communication is upon you. Solutions not properly explained will not be considered correct. Part of proper communication is the appearance and layout. If we cannot "decode" what you wrote, we cannot grade it as a correct solution.
4. You may consult outside sources, such as textbooks, but any use of any source **must** be documented in the assignment solutions.
5. You are permitted to discuss *general aspects* of the problem sets with other students in the class, but you must hand in your own copy of the solutions.
6. Your assignment solutions are due by 11:59PM on the due date and must be submitted on Brightspace.
    - Late assignments will be graded with a late penalty of 20% of the full grade per day up to two days past the deadline.
7. You are responsible for ensuring that your assignment is submitted correctly and without corruption.

Problem	Task 1, P1	Task 1, P2	Task 1, P3	Task 1, P4	Task 2, P1	Task 2, P2	Total
Points	10	10	10	10	10	20	70

In this assignment, you will participate in activities related to the operation and use of cryptographic tools and techniques. This assignment aims to assess your understanding of the **basic principles underlying Public-Key Cryptographic, including topics in Discrete Math, DH Key Exchange Protocol, RSA, Elliptic Curves and working with Binary Fields.**

Submission Requirements

Please read the following instructions very carefully and follow them precisely when submitting your assignment!

The following items are required for a complete assignment submission:

1. **PDF Assignment Report**: Submit a detailed report that carefully and concisely describes what you have done and what you have observed. Include cover page, appropriate code snippets and listings, as well as screenshots of program outputs and results. You also need to provide an adequate explanation of the observations that are interesting or surprising. You are encouraged to pursue further investigation beyond what is required by the assignment description.
2. **ZIP Archive of Source Code**: In addition to embedding source code listings in your assignment report, create and submit a ZIP archive of all programs that you write for this assignment. Please name each of your source code files with the problem number to which they correspond (e.g., for Task1 Problem 1, the source code file should be named Task1_Problem1.c). Your source code must compile and run, producing the desired output. Also, please remember to provide sufficient comments in your code to describe what it does and why.

Grading Notes

An important part of this assignment is following instructions. As such, the following grade penalties will be applied for failure to comply with the submission requirements outlined above:

- Failure to submit an Assignment Report will result in a grade of 0 for the assignment.
- Failure to submit the Source Code files will result in deduction of 10% of the full grade of the assignment.
- Failure of Source Code to compile/run will result in a grade of 0 for the corresponding problem(s).
- Failure to submit any deliverable in the required format (PDF or ZIP) will result in deduction of 5% of the full grade of the assignment.

General Note:

Any command starts with $ should be typed directly into the terminal.
Any command starts with >>> should be typed inside Python.

Task 1 Topics in Discrete Math

Problem 1: Euclidean and Extended Euclidean Algorithms:

A. Write a simple Python code to execute the Euclidean Algorithm.
Use the Euclidean Algorithm to compute the GCD of 91261 and 117035. Let's denote this number as D.

B. Write a simple Python code to execute the Extended Euclidean Algorithm.
Use the Extended Euclidean Algorithm to express the number D, found in Step A in terms of the input numbers. In other words, find a and b such that
$a \times 91261 + b \times 117035 = D$

C. Use the Extended Euclidean Algorithm to compute the Multiplicative Inverse of the element '171' in the prime field $\mathbb{Z}_{271}$. Write the result as an equation similar to Step B.

D. Write a simple Python code to compute
$$27^{350} \pmod{569}$$

E. Use Python math library to confirm correct operation of your code:

```
>>> import math
>>> math.gcd(91261,117035)
>>> pow(171,-1,271)
>>> pow(27,350,569)
>>> pow(27,350)
```

Note the difference between running the exponentiation within the modulus, and without the modulus.

Take screenshot from the two codes and include them in your report, along with screenshots of the result of executing your code.

Problem 2: Generate large prime numbers:

In this problem, we are going to use the OpenSSL library to generate really large prime numbers and use OpenSSL as well as Python to check if it is actually prime.

A. Let's start with generating a small prime number. In the terminal of the SCE VM machine used in the previous Assignment, type:
$openssl prime -generate -bits 4
This simply means generate a prime number with 4 bits. This should return the number '11' or '13', since these are the only 4-bit numbers that are prime.

B. Generate a large prime number of 2048 bits.
Note that the output is a random prime number with 2048 bits. It is almost impossible for

two assignment reports to generate the same prime number.
There are roughly 1.138×10^{613} prime numbers in this range with 2048 bits! For reference, there are around 10^{82} atoms in the whole observable universe [link]!
The number of prime numbers less than or equal to x is called the Prime-counting function $\pi(x) = x/\ln(x)$. Hence, we need $\pi(2^{2048}) - \pi(2^{2047}) = 1.138 \times 10^{613}$.

C. OpenSSL can itself test if a number is prime:
$ openssl prime 7
Replace the number '7', with the number generated in Step B.

D. In Python, we need the sympy library to test if a number is prime. In order to install this library, type:
$ python3 -m pip install sympy
Then, type
$ python3
To get inside python
Then,

```
>>> from sympy import isprime
>>> isprime(7)
```

Replace the number '7', with the number generated in Step B.

Write down the generated prime number in your report and take screenshot from OpenSSL and Python output of prime test and include them in your report.

Problem 3: Test a small prime finite field:

In this question, we will study the prime field $\mathbb{Z}_{19}$, as a sample toy prime field. Answer the following questions:

1- Write down the Multiplication tables in $\mathbb{Z}_{19}$.
2- List all the generators of $\mathbb{Z}_{19}$

You can use Python to simply multiply each element in $\mathbb{Z}_{19}$ by each other element and write down the result in the form of a table.
In order to check if any element is a generator, test

```
>>> for i in range(1,19):
...       if pow(x,i,19)==1:
...               print(i);break
```

While replacing x with an element in $\mathbb{Z}_{19}$.
If the result is 18, the selected element is a generator, which means x^1 to x^{17} generates all the other elements in the field, and only x^{18} brings the element back to 1.
If the result is anything less than 18, the selected element is not a generator. If you selected this element, all the possible powers of this element will create a subgroup, not the full group. Note that we expect to have subgroups of sizes {1, 2, 3, 6, 9, 18} since they all devices $p - 1 = 18$.

Take a screenshot from your code and write down the multiplication table and the list of generators.

Task 2 RSA

Problem 1: A toy example of unpadded RSA:

Let $p = 71$, $q = 89$, $n = pq$, the public exponent $e = 3$. Use basic unpadded RSA operations to find:

A. The private exponent d.
B. Signature of $m_1 = 5016$. Signature is the encryption using private exponent, denoted C_1.
C. Signature on $m_2 = 2097$. Signature is the encryption using private exponent, denoted C_2.
D. Verify correctness of the two signatures are valid by using the public exponent $e = 3$ to decrypt the messages and bring back the original text.
E. Compute $C_3 = C_1 \times C_2 (\text{mod}\, n)$,
   This is an encrypted value that is computed based on the two previous ciphertexts. We didn't need the private exponent to find this value.
F. Check if C_3 is the actual signature of the message $m_1 \times m_2 \ (\text{mod}\ n)$

Report all the values in your report.

Problem 2: RSA using basic commands from OpenSSL and Python:

In this problem, we are going to build an RSA private and public key set by using basic OpenSSL and Python commands.

A. Use OpenSSL to generate two large prime number of 1024 bits.
   $openssl prime -generate -bits 1024
   $openssl prime -generate -bits 1024
B. Go inside Python, by typing
   $python3
C. Copy the two prime numbers, call them p, and q
D. Then,

```
>>> n = p*q
>>> phi = (p-1)*(q-1)
>>> e = 65537
>>> d = pow(e,-1,phi)
```

E. Confirm that p and q are each of 1024 bits,
   Confirm that n is of 2048 bits, which is the currently recommended value of the RSA modulus.
   Confirm that indeed

```
>>> (e*d)%phi
```

   equals to 1.
F. Use m as any random number less than n.
G. Encrypt m using the private exponent (d) to generate a signature, then decrypt it using the public exponent (e), check if you bring back the same message.

```
>>> pow(m,d,n)
>>> pow(_,e,n)
```

Once again, note that there are an estimated $\pi(2^{1024}) - \pi(2^{1024}) = 1.265 \times 10^{305}$ prime numbers with 1024 bits. Since $n = p \times q$, where p and q can take any random value out of the 1.265×10^{305} possible values, there are an estimated 1.6×10^{610} possible values for the RSA public modulus n of 2048 bits.

Include screenshots of all the generated values.

Problem 3: RSA using OpenSSL:

Type in

```
$openssl genrsa 2048
```

This will generate an RSA key pair, with an n modulus of 2048 bits, and a public exponent e of 65537.
Note that the generated result, starting with:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAvrs72fxu119MonPFaB0URGwl86l0cJQrROT7cu3VGhsuNP3l
```

and ending with

```
M+L69cApeR2licnQNZdiublglwZKWjuazSuXwJ4RlPb51hEUvKDy
-----END RSA PRIVATE KEY-----
```

Is a PEM-encoded data structure that contains the p, q, n, d and some hash values for check. You can decode it using https://8gwifi.org/PemParserFunctions.jsp

Needless to say, once a private key is uploaded online, it is no longer valid for any transaction. The website now has a copy for this key pair.

Include screenshots of all the generated values.

Problem 4: Study a publicly available RSA key pair:

```
$GET https://appleid.apple.com/auth/keys
```

This will bring back some of the currently used public keys of Apple Inc. Note that the public key includes only e and n.

Include screenshots of all the generated values.

Task 3 Elliptic Curves

Problem 1: A toy example of Elliptic Curves:

In this question, we will study Curve-Cryptography over Edward Curves. The curve is defined as:

$$x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2$$

Then, the addition of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2,\ y_2)$ is defined as:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + x_2 y_1}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right),$$

For $(d\ =\ -20)$, and all the coordinates are computed in $\mathbb{Z}_{17}$, i.e., the x and y coordinates of any point can only take an integer value $0 \leq x, y \leq 16$.

Find the following:

A. List all the valid points on the curve.
B. Write down the addition table.
C. Identify all the points that can serve as a base point (generator) for this field.

Select any generator at random as a starting point on the curve, compute a point that represents the shared secret key of a simplified communication protocol using these parameters for the Edward-Curve Diffie-Hellman Key Exchange ($d = -20$ and $p\ =\ 17$) when Alice selects $x = 8$ and Bob selects $y = 11$.

Problem 2: Elliptic Curves using OpenSSL:

In this last problem, we are going to use OpenSSL to generate Elliptic Curve key pair.

A. Type:
```
$openssl ecparam -list_curves
```
to understand the set of curves supported by OpenSSL.

B. We are going to use the prime256v1 curve, which is one of the popular curves. Type:
```
$openssl ecparam -name prime256v1 -genkey -noout -out private-key.pem
```
to set the EC-parameters to use curve named prime256v1, and generate key to be stored in the output file named private-key.pem.
Check contents of the private-key.pem file using vi or any text editor. You can understand the content of this PEM-encoded file using
https://8gwifi.org/PemParserFunctions.jsp
Needless to say, once a private key is uploaded online, it is no longer valid for any transaction. The website now has a copy for this key pair.

C. Generate corresponding public key by typing:
```
$openssl ec -in private-key.pem -pubout -out public-key.pem
```

Include screenshots of all the generated values.

**********This is the end of Assignment 3**********