CPSC 418 / MATH 318 — Introduction to Cryptography
ASSIGNMENT 2

Due: **Wednesday, Mar. 11, 2020** at **11:55 PM** Total marks: **100** + 10 bonus marks

Prior to submission, be sure to familiarize yourself with the **Policies and Guidelines** as well as the **Specifications and Submission Procedure** as detailed on the assignments course webpage

http://people.ucalgary.ca/~rscheidl/418/assignments.html.

Assignments that don't follow these instructions will incur penalties, possibly even a score of zero.

Written Problems for CPSC 418 and MATH 318

Problem 1 — Arithmetic in the AES MixColumns operation (22 marks)

Recall that the MixColumns operation in AES performs arithmetic on 4-byte vectors using the polynomial $M(y) = y^4 + 1$. In this arithmetic, we have $M(y) = 0$, so $y^4 = 1$.

(a) In this part of the problem, we consider multiplication of 4-byte vectors (viewed as polynomials of degree ≤ 3 whose coefficients are bytes) by powers of y.

i. (2 marks) Formally prove that in this arithmetic, multiplication of any 4-byte vector by y is a circular left shift of the vector by one byte.

ii. (2 marks) Prove that in this arithmetic, $y^i = y^j$ for any integer $i \geq 0$, where $j \equiv i \pmod 4$ with $0 \leq j \leq 3$.

iii. (4 marks) Use part (a) (ii) to formally prove that multiplication of any 4-byte vector by y^i ($i \geq 0$) is a circular left shift of the vector by j bytes, where $j \equiv i \pmod 4$ with $0 \leq j \leq 3$.

(b) Next, we consider arithmetic involving the coefficients of the polynomial

$$c(y) = (03)y^3 + (01)y^2 + (01)y + (02) ,$$

that appears in MixColumns, where the coefficients of $c(y)$ are bytes written in hexadecimal (i.e. base 16) notation. Arithmetic involving this polynomial requires the computation of products involving the bytes (01), (02) and (03) in the Rijndahl field GF(2^8). Recall that in this field, arithmetic is done modulo $m(x) = x^8 + x^4 + x^3 + x + 1$.

i. (2 marks) Write the bytes (01), (02), (03) as their respective polynomial representatives $c_1(x)$, $c_2(x)$ and $c_3(x)$ in the Rijndahl field GF(2^8).

ii. (4 marks) Let $b = (b_7\, b_6 \cdots b_1\, b_0)$ be any byte, and let $d = (02)b$ be the product of the bytes (02) and b in the Rijndahl field GF(2^8). Then d is again a byte of the form $d = (d_7\, d_6 \cdots d_1\, d_0)$. Provide symbolic expressions for the bits d_i, $0 \leq i \leq 7$, in terms of the bits b_i of b.

iii. (3 marks) Provide analogous expressions as in part (b) (ii) for the byte product $e = (03)b$, where $b = (b_7\, b_6 \cdots b_1\, b_0)$ is any byte.

(c) The MixColumns operation performs multiplication of 4-byte vectors by the polynomial $c(y)$ of part (b). In this part of the problem, you will evaluate such products symbolically.

i. (3 marks) Let $s(y) = s_3y^3+s_2y^2+s_1y+s_0$ be a polynomial whose coefficients are bytes. Symbolically compute the product $t(y) = s(y)c(y) \bmod y^4 + 1$. The result should be a polynomial of the form $t(y) = t_3y^3+t_2y^2+t_1y+t_0$ where t_3, t_2, t_1, t_0 are bytes. Provide symbolic expressions for the bytes t_i, $0 \le i \le 3$, in terms of the bytes s_i. The equations should consist of sums of byte products of the form $01s_i$, $02s_i$, $03s_i$, $0 \le i \le 3$. You need *not* compute these individual byte products as you did in part (b).

ii. (2 marks) Write your solution of part (c) (i) in matrix form; i.e. give a 4×4 matrix C whose entries are bytes such that

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = C \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Note that this yields the matrix representation of MixColumns presented (without proof) in class.

Problem 2 — Error propagation in block cipher modes (12 marks)

Error propagation is often an important consideration when choosing a mode of operation in practice. In this problem, you will analyze the error propagation properties of an arbitrary block cipher in various such modes; note that these properties are independent of the cipher used.

(a) Suppose Alice wants to send a sequence of message blocks $M_0, M_1, M_2, \ldots$ to Bob, encrypted to ciphertext blocks $C_0, C_1, C_2, \ldots$ using some fixed key K. Assume that an error occurs during transmission of a particular block of ciphertext C_i. Justifying all your answers, explain which plaintext blocks are affected after Bob decrypts this (faulty) ciphertext block C_i when the cipher is used in

i. (2 marks) ECB mode?
ii. (2 marks) CBC mode?
iii. (2 marks) OFB mode?
iv. (2 marks) CFB mode with one register?
v. (2 marks) CTR mode?

(b) (2 marks) Suppose now that an error occurs in a particular plaintext block M_i before Alice encrypts it and sends the corresponding ciphertext C_i to Bob. Upon decryption of C_i, which plaintext blocks M_j are affected when using the cipher in CBC mode?

Problem 3 — Binary exponentiation (13 marks)

Recall the exponentiation algorithm given in class for evaluating $a^n \pmod m$ ($a \in \mathbb{Z}$, $m, n \in \mathbb{N}$):

1. Compute the binary representation of n:

$$n = b_0 2^k + b_1 2^{k-1} + \cdots + b_{k-1} 2 + b_k \ ,$$

with $b_0 = 1$, $b_i \in \{0, 1\}$ for $1 \le i \le k$, and $k = \lfloor \log_2 n \rfloor$.

2. Initialize $r_0 \equiv a \pmod{m}$.
3. For $0 \le i \le k-1$ compute $r_{i+1} = \begin{cases} r_i^2 \pmod{m} & \text{if } b_{i+1} = 0 \ , \\ r_i^2 a \pmod{m} & \text{if } b_{i+1} = 1 \ . \end{cases}$
4. Output r_k.

(a) (3 marks) To warm up with a toy example, compute $17^{11} \pmod{77}$ using the procedure above; answers that don't use the binary exponentiation algorithm will receive no credit, even if they are correct. Show all your work, and write down all your intermediate quantities b_i and r_i. Your answer should be an integer between 0 and 76.

(b) In this problem, you will formally prove that the binary exponentiation algorithm is correct.

i. (4 marks) Define $s_0 = 1$ and $s_{i+1} = 2s_i + b_{i+1}$ for $0 \le i \le k-1$. Use induction on i to prove that

$$s_i = \sum_{j=0}^{i} b_j 2^{i-j} \quad \text{for } 0 \le i \le k \ .$$

ii. (4 marks) Let r_i, $0 \le i \le k$, be defined as in steps 2 and 3 of the exponentiation algorithm. Use induction on i to prove that $r_i \equiv a^{s_i} \pmod{m}$ for $0 \le i \le k$.

iii. (2 marks) Prove that $a^n \equiv r_k \pmod{m}$, so the algorithm above does indeed compute $a^n \pmod{m}$ as claimed.

Problem 4 — A modified man-in-the-middle attack on Diffie-Hellman (10 marks)

Suppose Alice and Bob wish to generate a shared cryptographic key using the Diffie-Hellman protocol. As usual, they agree on a large prime p and a primitive root g of p. Suppose also that $p = mq + 1$ where q is prime and m is very small (so $p - 1 = mq$ has a large prime factor, as is generally required). Since g and p are public, it is easy for anyone to deduce m and q; for example by successively trial-dividing $p-1$ by $m = 2, 4, 6, \ldots$ and running a primality test such as the Fermat test on the quotient $q = (p-1)/m$ until primality of q is established.

Suppose an active attacker Mallory[1] intercepts $g^a \pmod{p}$ from Alice and $g^b \pmod{p}$ from Bob. She sends $(g^a)^q \pmod{p}$ to Bob and $(g^b)^q \pmod{p}$ to Alice.

(a) (2 marks) Show that Alice and Bob compute the same shared key K under this attack.

(b) (4 marks) Show that there are m possible values for K, and that Mallory can compute them all and hence easily guess the correct key K among them.

(c) (4 marks) What is the advantage of this variation of the man-in-the-middle attack over the version we discussed in class? Recall that for the attack from class, Mallory simply suppresses the messages $g^a \pmod{p}$ and $g^b \pmod{p}$ between Alice and Bob and replaces them with her own number $g^e \pmod{p}$, which results in the shared key $g^{ae} \pmod{p}$ between Mallory and Alice and the shared key $g^{be} \pmod{p}$ between Mallory and Bob.

[1] This is a standard name for active attackers and is meant to be reminiscent of the word "malicious".

Problem 5 — A simplified password-based key agreement protocol (8 marks)

The following is a simplified (and hence problematic) version of the key generation phase of the password-based key agreement protocol that you are being asked to implement in Problem 9 (the programming problem). Here, a client first performs a one-time registration of their authentication credentials with a server. These credentials can then be used to generate authenticated session keys between server and client via communication over an insecure channel.

All participants agree on a large public prime[2] $N = 2q + 1$, with q prime, and a public primitive root g of N. Each client has their own password p. To register with the server, a client computes $v \equiv g^p \pmod{N}$ and provides the server with the pair (I, v) where I is the client's user id.[3]

Protocol:

1. Client generates a random value a with $0 \le a \le N - 1$, computes $A \equiv g^a \pmod{N}$, and sends (I, A) to server, where I is the Client's user id.

   Server generates a random value b with $0 \le b \le N - 1$, computes $B \equiv g^b \pmod{N}$, and sends B to client.
2. Client computes $K_{\text{client}} \equiv B^{a+p} \pmod{N}$.

   Server retrieves client's authentication data (I, v) and computes $K_{\text{server}} \equiv (Av)^b \pmod{N}$.

Note that this protocol is similar to Diffie-Hellman, except that the client's password p and authentication credential v are incorporated in the key computation.

(a) [2 marks] Prove that Client and Server have a shared key after executing this protocol, i.e. prove that $K_{\text{server}} = K_{\text{client}}$.

(b) [3 marks] Suppose an adversary Mallory obtains client Ian's authentication data (I, v) (by intercepting Ian's transmission to the server upon his registration or by hacking into the server's database). Show how Mallory can masquerade as Ian, i.e. execute the protocol with the server (using a value A of her choice) and generate a valid key K_{client} that the server believes is shared with Ian.

(c) [3 marks] Consider the following two problems:

- *Key Recovery Problem:* Given any values $A \equiv g^a \pmod{N}$ and $B \equiv g^b \pmod{N}$ and any $v \in \mathbb{Z}_N^*$, find a key K produced by the protocol above.
- *Diffie-Hellman Problem:* Given any values $A \equiv g^a \pmod{N}$ and $B \equiv g^b \pmod{N}$, find a Diffie-Hellman key $K \equiv g^{ab} \pmod{N}$.

Note that the exponents a and b are assumed to be unknown for both these problems. Show how an attacker Mallory who can solve any instance of the key recovery problem can solve any instance of the Diffie-Hellman problem. (So informally, breaking the key agreement protocol above is at least as hard as breaking Diffie-Hellman.)

[2]We denote this prime by N, rather than p, because the letter p is reserved for the client's password.

[3]In practice, this needs to be done in a secure and tamper-proof manner. Also, in the computation of v, the client would use a hash of their password p rather than just p. For details, see the protocol description in Problem 9.

Written Problems for MATH 318 only

Problem 6 — Primitive roots for safe primes (6 marks)

Let $q \geq 3$ be a prime such that $p = 2q + 1$ is also prime. Let g be any primitive root of p. Prove that with the exception of $g^q \pmod{p}$, all the odd powers of g (i.e. $g, g^3 \pmod{p}, g^5 \pmod{p}, \ldots, g^{p-2} \pmod{p}$), are primitive roots of p.

(*Hint:* the following fact about divisibility, which you may use without proof, might come in handy: for any three nonzero integers a, b, c, if a is a divisor of the product bc and $\gcd(a, b) = 1$, then a is a divisor of c.)

Problem 7 — Discrete logarithms with respect to different primitive roots (8 marks)

Prove that the difficulty of the discrete logarithm problem is independent of the primitive root. Specifically, for any prime p, assuming that it is computationally feasible to extract discrete logarithms with respect to one primitive root of p, show how one can feasibly extract discrete logarithms with respect to any other primitive root of p.

Problem 8 — An algorithm for extracting discrete logarithms (21 marks)

Let p be a large prime and g a fixed primitive root of p. Let $h \in \mathbb{Z}_p^*$ be the modular inverse of g, so $gh \equiv 1 \pmod{p}$. Let $a \in \mathbb{Z}_p^*$. Define the following lists of elements in $\mathbb{Z}_p^*$:

$$y_i \equiv ah^i \pmod{p},\ 0 \leq i \leq m-1;$$
$$z_j \equiv (g^m)^j \pmod{p},\ 0 \leq j \leq k-1.$$

Here, m is a positive integer (an as yet unspecified parameter) and k is the smallest integer with $k \geq (p-1)/m$.

(a) (4 marks) Prove that there exist indices i, j with $0 \leq i \leq m-1$ and $0 \leq j \leq k-1$ such that $y_i = z_j$.
(*Hint:* Division with remainder of x by m where x is the (unknown) discrete logarithm of a with respect to g.)

(b) (4 marks) Consider the following procedure for computing the discrete logarithm of a with respect to g.

1. Compute the list $\mathcal{Y} = (y_0, y_1, \ldots, y_{m-1})$
2. If $y_i \equiv 1 \pmod{p}$ for some $i \in \{0, 1, \ldots, m-1\}$, then output i and quit.
3. Compute the list $\mathcal{Z} = (z_0, z_1, z_2, \ldots,)$ until an element z_j is found that appears in $\mathcal{Y}$.
4. Upon finding such a match, say $z_j = y_i$, output $x \equiv jm + i \pmod{p-1}$.

Prove that this procedure terminates and outputs the discrete logarithm of a.

(c) (4 marks) Assuming the worst case scenario (i.e. the entire list $\mathcal{Z} = (z_0, z_1, \ldots, z_{k-1})$ needs to be generated), how many modular multiplication are required to extract the discrete logarithm of a using the procedure above? Your count should be as accurate as possible (i.e. don't count modular multiplications that aren't needed). You may assume that the quantities m, k, h and $g^m \pmod{p}$ have been precomputed as they are independent of a. You may also ignore the cost of searching the list $\mathcal{Y}$ for an element $z \in \mathcal{Z}$.

(d) (4 marks) How should m be chosen to minimize the number of modular multiplication required by the procedure above? Explain your choice.

(e) Let $p = 107$.

i. (2 marks) Use the primitive root test from class to verify that 2 is a primitive root of p. Show your work.

ii. (3 marks) Use the procedure from part (b) and your choice of m from part (d) to compute the discrete logarithm of 6 with respect to 2 in $\mathbb{Z}_{107}^*$. Show your work. You may want to verify that your final answer is correct.

Programming Problem for CPSC 418 only

Problem 9 — A secure password based authentication and key exchange protocol (35 marks)

Overview. This problem considers the full, secure version of the password-based key agreement protocol introduced in Problem 5. This protocol, executed by Client and a Server, allows the Client to demonstrate to the Server knowledge of a password, but neither the password nor any other information that could be used to derive the password need to be transmitted. Additionally, the Server does not store password-equivalent data, so someone who intercepts authentication data or steals them from the Server's database is unable to masquerade as the Client without brute-forcing the Client's password.

Initialization. The following quantities are public system parameters:

- A safe prime $N = 2q + 1$, where q is a prime;
- A primitive root g of N;
- A fixed cryptographically secure hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$;
- The quantity $k = H(N||g)$ (where, as always, "$||$" denotes concatenation).

Registration. To register with the Server, a Client with user id I and password p performs the following steps:

1. Generates a *salt* s (a small random number);
2. Computes $x = H(s||p)$ and $v \equiv g^x \pmod{N}$;
3. Transmits the authenticated triple (I, s, v) securely to the Server for storage (note that this authentication and secure transmission are not covered by the protocol);
4. Disposes of x.

Protocol. To generate and verify a shared authenticated session key, the Server and Client perform the following steps:

1. Client generates a random value a with $0 \le a \le N - 1$, computes $A \equiv g^a \pmod{N}$, and sends (I, A), where I is the Client's user id.
   Server generates a random value b with $0 \le b \le N - 1$, computes $B \equiv kv + g^b \pmod{N}$, and sends (s, B), where s is the Client's salt.

2. Both compute $u \equiv H(A||B) \pmod{N}$.
3. Client computes $K_{\text{client}} \equiv (B - kv)^{a+ux} \pmod{N}$.
   Server computes $K_{\text{server}} \equiv (Av^u)^b \pmod{N}$.
4. Client computes and sends $M_1 = H(A||B||K_{\text{client}})$.
5. Server computes $H(A||B||K_{\text{server}})$ and compares the result to M_1. If they match, output a string indicating success; otherwise, abort.
6. Server computes and sends $M_2 = H(A||M_1||K_{\text{server}})$.
7. Client computes $H(A||M_1||K_{\text{client}})$ and compares the result to M_2. If they match, output a string indicating success; otherwise, abort.

Steps 1-3 generate the authenticated key shared between the Client and the Server. Steps 4-7 verify that both parties have computed the same shared key. If executed honestly, K_{client} and K_{server} are equal and the Server and Client were able to authenticate each other and establish a shared session key.

Problem. Your task is to implement the password-based key agreement protocol above in Python 3. Your program should consist of two modules, a Client and a Server, which communicate via a socket connection. All messages over the socket should be echoed to standard output by both the sending and receiving party. Each echoed message should *clearly* indicate its sender and intended receiver.

Use the implementation of SHA-256 from the Python `cryptography` library for your hash function. All other protocol steps should be implemented by yourself, although you may make use of other libraries in your code. A mathematical library like `sympy` may be useful for handling prime numbers.

Specifications. Design and implement your solution as two `Python 3` programs entitled `Client` and `Server`, invoked by the respective commands

`python3 Client.py` and `python3 Server.py`

The Server program performs initialization as follows:

1. Generates a random 512-bit safe prime by first generating a 511-bit random prime q and then checking whether $N = 2q + 1$ is prime. If it isn't, generate a new prime q and try again until a valid N is found.
2. Finds a primitive root g of N.
3. Establishes a socket connection to the Client and sends (N, g) to the Client via this socket connection.

The Client program performs registration as follows:

1. Prompts user for a username I and a password p.
2. Generates a random 16-byte salt s and computes the value v as described in step 2 of the registration phase.
3. Sends the triple (I, s, v) to the Server via socket connection.[4]

[4] You may assume that the registration is done in a secure and authenticated manner for this exercise.

In addition, you must adhere to the following specifications for parameter generation and hashing:

- Generate all random numbers using a *secure* random number generator.
- Ensure that $0 \leq a, b \leq N - 2$.
- Use the implementation of SHA-256 from `cryptography.hazmat.primitives import hashes` as your hash function.

Submission. Submit a description of your implementation in a separate README file in text format. *Do **not** include the written portion of the programming problem in the PDF file containing your solutions to the written problems.* Your description must include the following:

- A short argument and citation for the security of the random number generator you used.
- The procedures you used for generating your prime N and your primitive root g of N.
- A list of the files you have submitted that pertain to the problem, and a short description of each file.
- A list of what is implemented in the event that you are submitting a partial solution, or a statement that the problem is solved in full.
- A list of what is not implemented in the event that you are submitting a partial solution.
- A list of known bugs, or a statement that there are no known bugs.

Bonus Problem for CPSC 418 and MATH 318

Problem 10 — Playfair cipher cryptanalysis, 10 marks

Decrypt the following ciphertext that was encrypted using a *Playfair* cipher. You will need to research on our own how this cipher works since we did not discuss it in class. Show all your work; including a description of any software you used and what you used it for in case you did any programming for your solution. If you wish to submit source code, please include it as text at the end of the PDF file containing your answers to the written problems. Answers without satisfactory explanation and documentation of how they were obtained will receive *no* credit. Neither will answers obtained by simply running Playfair decryption from an online crypto applet or website on the ciphertext.

A text file containing the ciphertext can be downloaded from the "assignments" page.

Hint: the letter "J" is omitted from the alphabet to obtain the required 5×5 key square.

```
DBFNE XTZMF TOVBQ BQTOB XAOFP RTZEQ RHQKQ VDXOK ABPRQ
IELTV KEEXX SFSBP WDBOB YBFRO EABOR HQKQV TXGUE LABTH
TRXNO NEAAY XHBOH NEXBS HRQBZ MSEXP HFGZU GKCBD POEAA
YXHBO XPHFK RQIAB PRQIE LBXFZ BISEF XPBRA PRQIW CBRXD
YGTBQ TEAAY XHBOH NEXBS HRQBP RQIEL BXBTH BQBNF SISEB
XNUXP BURBX BQROX BATBR HBPWD RPROG UGXQR SEZYO XBAEL
AXCWB YBASX RKROP RHBOP BDPIC NOXEM RPKRX TELAX CWEQF
ZSXEL RHROP RHBUX DASEX NZNGU ELBXF SDGDB TBZLV ERHBO
RQ
```