CPSC 418 / MATH 318 — Introduction to Cryptography

ASSIGNMENT 2

Due: **Thursday, Nov. 4, 2021** at **11:55 PM** Total marks: **100**

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

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

Assignments that don't follow these instructions will incur penalties of varying degree, up to a score of zero.

Problems 1-5 are worth 60 marks and are required for both CPSC 418 and MATH 318 students.

Problem 6 and 7 are worth 40 marks and are for MATH 318 students only; CPSC 418 students may do these problems for limited extra credit.

Problem 8 is worth 40 marks and is for CPSC 418 students only; MATH 318 students may do this problem for limited extra credit.

The bonus credit policy can also be found under the link above.

Some of the problems use letters in caligraphy font, such as $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$, $\mathcal{X}$ and $\mathcal{Y}$. You can generate that font with the LaTeX command

```
\mathcal{Z}
```

which produces the output $\mathcal{Z}$. To avoid having to type such a rather lengthy command repeatedly, the Latex template provided includes aliases for these five letters. Specifically, `$\M$` produces $\mathcal{M}$, `$\C$` produces $\mathcal{C}$ etc.

Written Problems for CPSC 418 and MATH 318

Problem 1 — Conditional entropy (9 marks)

Let X, Y be two random variables with respective sample spaces $\mathcal{X}$, $\mathcal{Y}$ and probability distributions $p(x), p(y)$ ($x \in \mathcal{X}, y \in \mathcal{Y}$). Recall that the *joint probability* $p(x, y)$ is the probability that $p(X = x)$ and $p(Y = y)$; it is related to the conditional probability via the formula

$$p(x, y) = p(x|y)p(y) \quad \text{for all } x \in \mathcal{X} \text{ and } y \in \mathcal{Y}.$$

The *conditional entropy* or *equivocation* of X given Y is defined as

$$H(X|Y) = \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p(x, y) \log_2 \left(\frac{1}{p(x|y)} \right) = \sum_{y \in \mathcal{Y}} p(y) \sum_{x \in \mathcal{X}} p(x|y) \log_2 \left(\frac{1}{p(x|y)} \right) ,$$

where the sums $\sum_{x \in \mathcal{X}}$ and $\sum_{y \in \mathcal{Y}}$ run over all outcomes of X and of Y, respectively, such that $p(x|y) > 0$. Informally,[1] the equivocation $H(X|Y)$ measures the uncertainty about X given Y.

a. Consider a cryptosystem with plaintext space $\mathcal{M} = \{M_1, M_2, M_3, M_4\}$ and ciphertext space $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$. Suppose that plaintexts and ciphertexts are uniformly distributed, i.e. $p(M_i) = p(C_j) = 1/4$ for $1 \leq i, j \leq 4$. Now suppose each ciphertext has only two possible decryptions as follows:

- The decryption of C_1 is either M_1 or M_2, with either possibility equally likely;
- The decryption of C_2 is either M_3 or M_4, with either possibility equally likely;
- The decryption of C_3 is either M_2 or M_3, with either possibility equally likely;
- The decryption of C_4 is either M_1 or M_4, with either possibility equally likely.

(i) (2 marks) Determine $p(M_i|C_j)$ for all i, j with $1 \leq i, j \leq 4$.

(ii) (1 mark) Does this system provide perfect secrecy? Explain your answer?

(iii) (3 marks) Compute $H(M|C)$.

b. (3 marks) Suppose a cryptosystem provides perfect secrecy and assume that $p(M) > 0$ for all $M \in \mathcal{M}$. Prove that $H(M|C) = H(M)$.

Problem 2 — Fun with binary polynomials (11 marks)

In this problem, we consider polynomials with binary coefficients, i.e. coefficients 0 or 1 where arithmetic is done modulo 2 (or equivalently, as x-or).

a. (3 marks) List all the polynomials of degree 3 with binary coefficients in lexicographical order.

b. (3 marks) Recall that a polynomial is *reducible* if has a factorization into non-constant polynomials of smaller degree, and *irreducible* otherwise. List all the reducible polynomials of degree 3 with binary coefficients. For each of these polynomials, provide a nontrivial[2] factorization.

[1] Shannon measured the security of a cipher in terms of the key equivocation $H(K|C)$, i.e. the amount of information about a key K that is not revealed by a given ciphertext C.

[2] I.e. not $f(x) = 1 \cdot f(x)$ or $f(x) = f(x) \cdot 1$.

c. Consider the finite field $\mathrm{GF}(2^4)$ obtained via arithmetic modulo the irreducible degree 4 polynomial $p(x) = x^4 + x + 1$. That is, $\mathrm{GF}(2^4)$ consists of polynomials of degree ay most 3 with binary coefficients, and addition and multiplication are performed modulo $p(x)$.

(i) (3 marks) Compute the product of $f(x)g(x)$ in $\mathrm{GF}(2^4)$ where $f(x) = x^2 + 1$ and $g(x) = x^3 + x^2 + 1$.

(ii) (2 marks) Find the inverse of $f(x) = x$ in $\mathrm{GF}(2^4)$, i.e. the polynomial $g(x)$ such that $f(x)g(x) = 1$ in $\mathrm{GF}(2^4)$.
Hint: Be smart about this. You do *not* need to resort to long division here. Instead, use the fact that $p(x) = 0$ in $\mathrm{GF}(2^4)$.

Problem 3 — Arithmetic in the AES MixColumns operation (20 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. If you use polynomial arithmetic for your answer, don't forget to convert your final polynomial back to a 4-byte vector.

(ii) (4 marks) Generalizing part (i) to other powers of y, formally prove that multiplication of any 4-byte vector by y^i $(0 \leq i \leq 3)$ is a circular left shift of the vector by i bytes. Again, remember to convert your polynomial result back to a 4-byte byte vector.

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 $\mathrm{GF}(2^8)$. 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 $\mathrm{GF}(2^8)$.

(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 $\mathrm{GF}(2^8)$. Then d is again a byte of the form $d = (d_7\, d_6 \cdots d_1\, d_0)$. Provide formulas for the bits d_i, $0 \leq i \leq 7$, in terms of the bits b_i.

(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 formulas for the bytes t_i, $0 \leq i \leq 3$, in terms of the bytes s_i. The equations should consist of sums of byte products of the form $01s_i$, $02s_i$, $03s_i$, $0 \leq i \leq 3$. 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 4 — 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?
Hint: Think about this carefully; it's easy to make a conceptual mistake here.

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 CPSC 418 students are asked to implement in Problem 8 (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 safe prime[3] $N = 2q+1$, with q prime, and a public primitive root g of N. Each client has their own password p, interpreted as an integer between 0 and $N-2$ (inclusive). 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.[4]

Protocol:

1. Client generates a random value a with $0 \leq a \leq N-1$, 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 $0 \leq b \leq N-1$, 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 Mallory[5] obtains client Ian's authentication data (I, v), for example, 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.
   Hint: Concoct a value A from Ian's credential v such that the server's computation of K_{server} in step 2 produces a value that Mallory can easily compute. Note that $A = 0$ is not a valid choice because A must belong to $\mathbb{Z}_p^*$.

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 any valid 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 the Diffie-Hellman key $g^{ab} \pmod{N}$.

   Solving the first problem breaks the key agreement protocol above, solving the second problem breaks the Diffie-Hellman protocol.

   Assume a passive attacker Eve has an algorithm for solving any instance of the key recovery problem. Show how she can use her algorithm to solve any instance of the Diffie-Hellman

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

[4] 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 8.

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

problem. Informally, this means that breaking the key agreement protocol above is at least as hard as breaking Diffie-Hellman. Note that the exponents a and b are assumed to be unknown for both these problems, and Eve cannnot find them because we are not assuming that she has the ability to extract discrete logarithms.
Hint: Don't overthink this. The answer is very simple.

Written Problems for MATH 318 only

Problem 6 — Joint entropy (20 marks)

Let X, Y be two random variables with respective sample spaces $\mathcal{X}, Y$ and probability distributions $p(x), p(y)$ $(x \in \mathcal{X}, y \in \mathcal{Y})$. Assume for simplicity that $p(x) > 0$ for all $x \in X$ and $p(y) > 0$ for all $y \in Y$.

a. (1 mark) Explain why $\displaystyle\sum_{x \in \mathcal{X}} p(x|y) = 1$ for all $y \in \mathcal{Y}$.

b. (2 marks) Use part (a) and the formula connecting the joint and conditional probability given in Problem 1 to prove that

$$\sum_{x \in \mathcal{X}} p(x, y) = p(y) \quad \text{for all } y \in \mathcal{Y} \ .$$

c. (2 marks) Use part (b) to prove that

$$\sum_{y \in \mathcal{Y}} p(x, y) = p(x) \quad \text{for all } x \in \mathcal{X} \ .$$

d. (6 marks) The *joint entropy of X and Y* captures the combined uncertainty of X and Y and is defined to be

$$H(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \left(\frac{1}{p(x, y)} \right) \ .$$

Prove that $H(X, Y) \leq H(X) + H(Y)$. Use parts (b) & (c) and (without proof) the fact that $p(x, y) \geq p(x)p(y)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Informally, this result says that the total uncertainty of X and Y can only decrease if X and Y occur together.

e. (3 marks) Prove that equality holds in part (d) if and only if and only if X and Y are independent, i.e. x and y are independent for all outcomes $x \in X$ and $y \in Y$. Informally, this result says that if X and Y are independent, then the fact that they occur together does not reduce their total uncertainty.

f. (5 marks) Use part (b) to prove that $H(X, Y) = H(X|Y) + H(Y)$, where $H(X|Y)$ was defined in Problem 1.

g. (1 mark) Combine the results of parts (f) and (d) to prove that $H(X|Y) \leq H(X)$. Informally, this result says that the uncertainty of X can only decrease when some side information Y is known.

Problem 7 — A characterization of perfect secrecy (20 marks)

In this problem, you will prove the following result, due to Shannon and mentioned without proof in class:

> **Theorem.** Consider a cryptosystem with plaintext space $\mathcal{M}$, ciphertext space $\mathcal{C}$ and key space $\mathcal{K}$ such that $|\mathcal{M}| = |\mathcal{C}| = |\mathcal{K}|$. Assume that[6] $p(M) > 0$ for all $M \in \mathcal{M}$ and $p(C) > 0$ for all $C \in \mathcal{C}$. Then the system provides perfect secrecy if and only if
>
> A. For every plaintext $M \in \mathcal{M}$ and every ciphertext $C \in \mathcal{C}$, there exists a unique key $K \in \mathcal{K}$ such that C is the encryption of M under key K, and
> B. every key $K \in \mathcal{K}$ is used with equal probability $1/|\mathcal{K}|$.

a. (Proof of "only if") Assume first that the system provides perfect secrecy, so $p(C|M) = p(C)$ for all $M \in \mathcal{M}$ and $C \in \mathcal{C}$ with $p(M) > 0$.

   (i) (2 marks) Prove that for every $M \in \mathcal{M}$ and every $C \in \mathcal{C}$, there exists at least one key $K \in \mathcal{K}$ such that C is the encryption of M under key K.

   (ii) (3 marks) Prove that for every $M \in \mathcal{M}$ and every $C \in \mathcal{C}$, there exists at most one key (and hence exactly one key by part (a) (i)) $K \in \mathcal{K}$ such that C is the encryption of M under key K.

   (iii) (3 marks) Prove that every key $K \in \mathcal{K}$ is chosen equally likely.

b. (Proof of "if") Suppose (A) and (B) above hold.

   (i) (2 marks) Prove that for every $C \in \mathcal{C}$ and every $K \in \mathcal{K}$, there exists at least one plaintext $M \in \mathcal{M}$ such that M is the decryption of C under key K.

   (ii) (3 marks) Prove that for every $C \in \mathcal{C}$ and every $K \in \mathcal{K}$, there exists at most one plaintext (and hence exactly one plaintext by part (b) (i)) $M \in \mathcal{M}$ such that M is the decryption of C under key K.

   (iii) (5 marks) Prove that $p(C) = 1/|\mathcal{K}|$ for all $C \in \mathcal{C}$.

   (iv) (2 marks) Prove that $p(C|M) = 1/|\mathcal{K}|$ for all $M \in \mathcal{M}$ and $C \in \mathcal{C}$.

Programming Problem for CPSC 418 only

Problem 8 — Secure password based authentication and key exchange (40 marks)

Overview. This problem considers the full, secure version of the password-based key agreement protocol introduced in Problem 5. This protocol, executed by a 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 needs 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.

To execute the protocol, there is an initialization and registration phase between the Client and Server. Doing this securely is in itself a complex problem, and so we instead perform a simplified version as described below.

The Server initializes in the following manner:

1. Generates a 512-bit safe prime by first generating a random 511-bit prime q and checking whether $N = 2q + 1$ is prime. Repeat until a valid N is found.
2. Finds a primitive root g of N, which may be up to 512 bits in size.
3. Compute the quantity $k = H(g||N)$, where '||' denotes concatenation and H is a cryptographically secure hash function.
4. Opens a socket connection on a specified IP and port and waits for a Client to connect to it. When that happens it attempts to receive a single byte which determines the Server's response. This byte is limited to the UTF-8 encoding of the characters '`r`' indicating Client wishes to initiate registration; '`p`' indicating the Client wishes to initiate the protocol; and '`q`' indicating the Client wishes the Server to shut down.[7]

The Server performs registration as follows:

1. Server sends the values g, N to the Client.
2. The Server then receives s, v, a single byte containing the length of the Client's username, and finally `username`, storing all these values.
3. The Server closes the connection.

The Client performs registration as follows:

1. The username `username` and a password `pw` are retrieved from the command line.[8]
2. Generates a random 16-byte salt s.
3. Connects and sends to the Server a one-byte character 'r';
4. Receives the values g, N;
5. Computes $x = H(s||\texttt{pw})$ and $v \equiv g^x \pmod N$;
6. Computes and sends s, v, a single byte containing the byte-length of `username`, and finally `username` encoded in UTF-8.[9]
7. The Client closes the connection.

In a similar manner, if a Client connects to the Server and wishes to initiate the protocol, they will first send a single byte corresponding to 'p' before commencing with the protocol described below.

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

1. The Client sends 'p'.

[7] A true implementation would not have this last feature, but it makes debugging much easier.

[8] The program template will do this for you, as well as check that the length of the username encodes to less than 255 bytes.

[9] This works on the assumption that the first connection to the Server is legitimate; provided that is true, and the Server does not forget the values it stored, impersonation is computationally infeasible.

2. Server sends g, N. The Client checks that these match the stored values, terminating the connection if they don't.
3. Client generates a random value a with $0 \leq a \leq N-1$, computes $A \equiv g^a \pmod{N}$, and sends A, one byte containing the length of the encoded username, and the UTF-8 encoded `username`.
4. Server generates a random value b with $0 \leq b \leq N-1$, computes $B \equiv kv + g^b \pmod{N}$, and sends s and B, where s is the Client's salt.
5. Client and Server compute $u \equiv H(A||B) \pmod{N}$.
6. Client computes $K_{\text{client}} \equiv (B - kv)^{a+ux} \pmod{N}$.

   Server computes $K_{\text{server}} \equiv (Av^u)^b \pmod{N}$.
7. The Server sends `bits`, a single byte representing a number.
8. The Client finds a value Y such that the first `bits` bits of $H(K_{\text{client}}||Y)$ are zero.[10] This value is sent to the Server.
9. Server computes $H(K_{\text{server}}||Y)$ and verifies the expected number of bits are zero. If they are not, the authentication has failed and the socket is closed.
10. Server computes and sends $M_1 = H(K_{\text{server}}||A||Y)$.
11. Client computes $H(K_{\text{client}}||A||Y)$ and compares the result to M_1. If they do not match, the authentication has failed.
12. The Server closes the socket and waits for further connections.

Steps 1-6 generate the authenticated key shared between the Client and the Server. Steps 7-12 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.

- At the end of either registration or the protocol, the Server should resume listening on the socket.
- Note that in order to reliably receive information from the socket, the byte-length of the data must typically be known. Unless otherwise specified, the size of any value taken modulo N should be the same length as N (512 bits or 64 bytes). Note that all hash digests are 256 bits or 32 bytes in size, and the salt is 16 bytes. The encoded username may be up to 255 bytes in length.
- Please note the importance of the order that values are sent over the sockets.

Problem. Your task is to complete the template program named

`basic_auth.py`

which performs the above password-based key agreement protocol. The program consists of a number of functions corresponding to establishing a connection and transmitting data through sockets, parameter generation, and the actions performed by the Client and the Server.

[10] To be explicit, the first `bits // 8` bytes in the byte representation of Y must be zero, according to Python's ordering, while the remaining `bits % 8` bits of the `(bits // 8)+1` byte must be zero, *starting from the most significant bit and ending at the least.* When printed, the zero bits will form a continuous region.

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 a template like the following:[11]

```
Server:  N = (integer value of N)
Client:  Received <(hex representation of incoming data)>
Server:  Authentication was successful.
Client:  ERROR, the socket was closed.
```

For this exercise the hash function H will be BLAKE2b-256.[12] When randomness is required, you must use a cryptographically-secure source. We will allow use of the `sympy` library for determining if a number is prime,[13] as fast primality tests are difficult to implement, but the portions of `sympy` relating to primitive roots will not be allowed.

Specifications. Fill in the empty functions in the template program `basic_auth.py`. Once complete, you should be able to perform both registration and a key agreement between two parties, one acting as the `Client` and the other as `Server`, by running

```
python3 basic_auth.py --addr 127.0.4.18:3180[14] --server
python3 basic_auth.py --addr 127.0.4.18:3180 --client
python3 basic_auth.py --addr 127.0.4.18:3180 --quit
```

You can also perform all these actions with one invocation of `basic_auth.py` by stacking them, although this makes debugging substantially more difficult. There will be additional command-line options to change the username and password from the defaults.

In detail, the functions you'll need to fill in fall into roughly three categories:

a. Networking Functions:

   (i) `create_socket(...)`, which creates a socket connection on the specified IP and port.

   (ii) `send(...)`, which sends bytes through the specified socket.

   (iii) `receive(...)`, which receives `length` bytes through the socket.

b. Low-level Functions:

   (i) `safe_prime(...)`, which takes in a positive integer `bitsize` and creates a random safe prime N with bit-length equal to `bitsize`, and whose most significant bit is 1.

   (ii) `prim_root(...)`, which takes in a prime integer N and finds a primitive root g of N.

   (iii) `calc_x(...)`, which computes the value x from the registration step using the given salt and password.

[11] The printed text is primarily for human eyes, not computer parsing, so there is no need to match these examples precisely.

[12] Note that BLAKE2b-256 is not the same as BLAKE2b-512 with part of the digest discarded.

[13] `sympy`'s routines for generating a random prime number are not suitable for cryptographic use.

[14] There will be a default IP address and port, so manually specifying them in this manner is usually redundant. Past students have had difficulty running the code with the default values on Mac OS X, however, so it's worth showing how to change those defaults.

(iv) `calc_u(...)`, which computes the value u using the provided inputs.

(v) `calc_A(...)`, which computes the value A using the provided inputs.

(vi) `calc_B(...)`, which computes the value B using the provided inputs.

(vii) `calc_K_client(...)`, which computes the value K_{client}.

(viii) `calc_K_server(...)`, which computes the value K_{server}.

(ix) `find_Y(...)`, which finds a suitable value of Y.

(x) `calc_M1(...)`, which computes the value M_1.

c. High-level Functions:

(i) `client_register(...)`, which sends and receives information to and from the Server as described in the Client registration.

(ii) `server_register(...)`, which receives and sends information from and to the Client through the socket connection, as outlined in the Server registration.

(iii) `client_protocol(...)`, which performs the Client's side of the protocol.

(iv) `server_protocol(...)`, which performs the Server's side of the protocol.

(v) `client_prepare(...)`, which computes the Client's registration tuple $(s, v, username)$.

(vi) `server_prepare(...)`, which computes the Server's registration values g, N, k.

Note that some values may be supplied as an integer or as a `bytes` object; your functions will need to translate between them as the context requires. All numbers are converted to `bytes` via network byte order, which is big-endian.[15] Additional details and documentation of these functions can be found in the template program found on the Piazza resources page.

Submission. Submit a completed version of the template program with filename

`basic_auth.py`

If you've spread your code across multiple source files, submit all of them.

Provide 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 should include the following:

- 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.

[15] Functions will be supplied with the template to help with conversion.