CPSC 418 / MATH 318 — Introduction to Cryptography ASSIGNMENT 3

Due: **Saturday, April 18, 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 — Flawed MAC designs, 13 marks

For this problem, you need to carefully trace through the given MAC algorithms to understand the given attacks and explain how computation resistance is defeated.

Recall that iterated hash functions employ a *compression function* f in multiple rounds; SHA-1 is an example of such a hash function. Here, f takes as input pairs of n-bit blocks and outputs n-bit blocks, for some $n \in \mathbb{N}$. The compression function f is assumed to be public, i.e. anyone can compute values of f. As a result, the hash function is also public, so any one can compute hash values. For simplicity, we assume that the bit length of any message to be hashed is a multiple of n, i.e. messages have been padded appropriately prior to hashing. Then the input to an iterated hash function is a message M consisting of L blocks $P_1, P_2, \ldots, P_L$, each of length n. An algorithmic description of an n-bit iterated hash function is given as follows (as usual. "$\|$" denotes concatenation of strings).

```
Algorithm ItHash
Input: A message $M = P_1 \| P_2 \| \cdots \| P_L$
Output: An $n$-bit hash of $M$
1: Initialize $H := 0^n$   ($n$ zeros)
2: for $i = 1$ to $L$ do
3:     $H := f(H, P_i)$
4: end for
5: Output $H$
```

The following attacks show that the two "obvious" designs for using an iterated hash function as the basis for a MAC — prepending or appending the key to the message and then hashing — are not computation resistant. These attacks were informally discussed in class by Randy on March 4; in this question, you are asked to present a formal argument for the defeat of computation resistance. For simplicity, assume that keys also have length[1] n.

(a) (5 marks) Define a message authentication function PHMAC ("Prepend Hash MAC") via

$$\mathrm{PHMAC}_K(M) := \textsc{ItHash}(K \| M) = \textsc{ItHash}(K \| P_1 \| P_2 \| \cdots \| P_L)$$

[1]The attack in part (a) will in fact work for any key length, but for part (b), this key length restriction is required.

for any message $M = P_1 \| P_2 \| \cdots \| P_L$.

Suppose an attacker knows a message/PHMAC pair $(M_1, \text{PHMAC}_K(M_1))$. Let X be an arbitrary n-bit block and put $M_2 = M_1 \| X$. Show how an attacker can compute $\text{PHMAC}_K(M_2)$ without knowledge of K, thereby defeating computation resistance for PHMAC.

Hint: Suppose M_1 consists of L blocks. Tracing through the ItHash algorithm, compare the outputs of the first $L+1$ rounds of $\text{PHMAC}_K(M_1)$ and $\text{PHMAC}_K(M_2)$.

(b) (8 marks) Define a message authentication function AHMAC_K ("Append Hash MAC") via

$$\text{AHMAC}_K(M) := \text{ItHash}(M \| K) = \text{ItHash}(P_1 \| P_2 \| \cdots \| P_L \| K)$$

for any message $M = P_1 \| P_2 \| \cdots \| P_L$.

Assume that ItHash is not weakly collision resistant, and suppose an attacker knows a message/AHMAC pair $(M_1, \text{AHMAC}_K(M_1))$. Show how she can find (without knowledge of K) a second message/AHMAC pair $(M_2, \text{AHMAC}_K(M_2))$, thereby defeating computation resistance.

Hint: Note that on input any L-bit message, the first L rounds of the computation of $\text{AHMAC}_K(M)$ do not depend on K, just on M.

Problem 2 — Fast RSA decryption using Chinese remaindering, 8 marks)

In this problem, as usual, a user Alice has an RSA public key (e, n) with corresponding private key d. Here, $n = pq$ for distinct large primes p and q.

If Alice does not discard p and q after computing n and $\phi(n)$, she can employ an alternative decryption procedure as described below (based on the *Chinese Remainder Theorem* which some of you may have seen before). For a given ciphertext C $(1 \leq C \leq n-1, \gcd(C, n) = 1)$, she proceeds as follows:

Step 1. Compute

$$\begin{aligned} d_p &\equiv d \pmod{p-1}, \quad 0 \leq d_p \leq p-2\ , \\ d_q &\equiv d \pmod{q-1}, \quad 0 \leq d_q \leq q-2\,. \end{aligned}$$

Step 2. Compute

$$\begin{aligned} M_p &\equiv C^{d_p} \pmod{p}, \quad 1 \leq M_p \leq p-1\ , \\ M_q &\equiv C^{d_q} \pmod{q}, \quad 1 \leq M_q \leq q-1\ . \end{aligned}$$

Step 3. Use the Extended Euclidean Algorithm to find integers x, y such that

$$px + qy = 1\ .$$

(Such integers exist because $\gcd(p, q) = 1$.)

Step 4. Set $M \equiv pxM_q + qyM_p \pmod{n}$, $0 \leq M \leq n-1$.

Prove that if the above procedure decrypts correctly. That is, if $C \equiv M^e \pmod{n}$ is a ciphertext obtained by encrypting a message M the "normal" RSA way, and M' is the result of applying the procedure above to C, prove that $M' = M$.

Remark: It can be shown that this method is generally twice as fast as ordinary RSA decryption since it performs arithmetic with respect to moduli of half-size. So this is what is generally used in practice.

Problem 3 — RSA primes too close together, 18 marks)

This problem explores a *difference of squares* approach to factoring an RSA modulus due to Fermat, and hence also known as *Fermat factorization.* Fermat's idea was to attempt to find a representation of an integer n as a difference of squares $n = x^2 - y^2$ where $0 < y < x < n$, which would lead to a factorization $n = (x + y)(x - y)$. Of course if $x + y = n$ and $x - y = 1$, then this doesn't help. But if $n = pq$ is an RSA modulus for example, the hope is that

$$x + y = p \ , \qquad x - y = q$$

or vice versa; in which case

$$x = \frac{p + q}{2} \ , \qquad y = \frac{p - q}{2} \ .$$

When p and q are close together, the quantity y is very small compared to x and we will see that this represents a serious attack on RSA.

Let $n = pq$ with odd primes p, q, and assume without loss of generality that $p > q$ (so $y > 0$). In this problem, all square roots are assumed to be positive, i.e. when we write $\sqrt{z}$ for some $z > 0$, we mean the positive square root of z.

(a) (3 marks) Prove that $p > x > \sqrt{n}$.

(b) (8 marks) Consider the following algorithm:

```
Algorithm Fermat Factorization
Input: $n = pq$ with $p > q$
Output: $q$
1: Put $a = \lceil \sqrt{n} \rceil$ ($\sqrt{n}$ rounded up to the nearest integer)
2: $b := \sqrt{a^2 - n}$
3: while $b$ is not an integer do
4:     $a := a + 1$; $b := \sqrt{a^2 - n}$
5: end while
6: Output $a - b$.
```

Prove that this algorithm terminates when $a = x$ and outputs q. Note that there are three items to show here:

- The "while" clause is satisfied when $a = x$;
- The algorithm does not terminate sooner, i.e. it does not terminate for any value $a < x$;
- When the algorithm terminates with $a = x$, it outputs q.

(c) (2 marks) Show that the number of loop iterations executed by the algorithm is $x - \lceil \sqrt{n} \rceil + 1$.

(d) (3 marks) Prove that $x - \lceil \sqrt{n} \rceil < \dfrac{y^2}{2\sqrt{n}}$.

(*Hint:* Consider $(x - \sqrt{n})(x + \sqrt{n})$.)

(e) (2 marks) Finally, the coup-de-grâce. Suppose $p - q < 2B\sqrt[4]{n}$ for some integer B that is very small compared to n; e.g. B could be on the order of a power of $\log_2(n)$. In other words, p and q are very close together; they agree in nearly half of their most significant bits. Prove that the above algorithm factors n after at most

$$\frac{B^2}{2} + 1$$

loop iterations.

Problem 4 – The El Gamal public key cryptosystem is not semantically secure, 12 marks

This problems requires typesetting Legendre symbols. To facilitate this, include at the beginning of your assignment file, right after the line `\documentclass{assignment}` the two lines

```
\usepackage{amsmath}
\providecommand{\Leg}[2]{\genfrac{(}{)}{}{}{#1}{#2}}
```

Be sure that you copy these *verbatim*; the easiest is to copy and paste them right from this PDF file. The assignment template provided on the course website already includes these lines. The command `$\Leg{a}{n}$` will produce the typeset output $\genfrac{(}{)}{}{}{a}{n}$, which is much easier than producing a fraction with large parentheses around it.

Recall that for the El Gamal public key cryptosystem, a user Alice produces her public and private keys as follows:

Step 1. Selects a large prime p and a primitive root g of p.

Step 2. Randomly selects x such that $0 < x < p - 1$ and computes
$y \equiv g^x \pmod{p}$.

Alice's public key: $\{p, g, y\}$

Alice's private key: $\{x\}$

To encrypt a message $M \in \mathbb{Z}_p^*$ intended for Alice, Bob selects a random integer $k \in \mathbb{Z}_{p-1}$, computes $C_1 \equiv g^k \pmod{p}$ and $C_2 \equiv My^k \pmod{p}$, and sends $C = (C_1, C_2)$ to Alice.

Alice decrypts the ciphertext $C = (C_1, C_2)$ by computing $C_2 C_1^{p-1-x} \equiv M \pmod{p}$.

In this problem, you will prove that the El Gamal PKC is not polynomially secure, and hence not semantically secure. This is because an attacker Mallory can distinguish messages according to whether they are quadratic residues or quadratic nonresidues modulo p.

Mallory mounts her attack with the following procedure:

Step 1. Selects two messages M_1 and M_2 such that $M_1 \in QR_p$ and $M_2 \in QN_p$,
and obtains ciphertext $C = (C_1, C_2) = E(M_i)$ where $i = 1$ or 2
(Mallory's task is precisely to ascertain whether $i = 1$ or $i = 2$).

Step 2. Computes the Legendre symbols $\genfrac{(}{)}{}{}{y}{p}$, $\genfrac{(}{)}{}{}{C_1}{p}$ and $\genfrac{(}{)}{}{}{C_2}{p}$.

Step 3. If $\left(\frac{y}{p}\right) = 1$ and $\left(\frac{C_2}{p}\right) = 1$, she asserts that $C = E(M_1)$.

If $\left(\frac{y}{p}\right) = 1$ and $\left(\frac{C_2}{p}\right) = -1$, she asserts that $C = E(M_2)$.

If $\left(\frac{y}{p}\right) = -1$, $\left(\frac{C_1}{p}\right) = 1$ and $\left(\frac{C_2}{p}\right) = 1$, she asserts that $C = E(M_1)$.

If $\left(\frac{y}{p}\right) = -1$, $\left(\frac{C_1}{p}\right) = 1$ and $\left(\frac{C_2}{p}\right) = -1$, she asserts that $= E(M_2)$.

If $\left(\frac{y}{p}\right) = -1$, $\left(\frac{C_1}{p}\right) = -1$ and $\left(\frac{C_2}{p}\right) = 1$, she asserts that $C = E(M_2)$.

If $\left(\frac{y}{p}\right) = -1$, $\left(\frac{C_1}{p}\right) = -1$ and $\left(\frac{C_2}{p}\right) = -1$, she asserts that $C = E(M_1)$.

Note that this procedure requires three Legendre symbol computations — which can be done with modular exponentiation by Euler's Criterion — and hence always takes polynomial time. Note also that Mallory states her assertions with certainty, i.e. probability 1.

Prove that Mallory's assertions are correct, so the El Gamal system is not semantically secure.

Problem 5 — An IND-CPA, but not IND-CCA secure version of RSA, 12 marks

Consider the following semantically secure variant of the RSA public key cryptosystem:

Parameters:

- m — length of plaintext messages to encrypt (in bits)
- (n, e) — Alice's RSA public key (n has k bits)
- d — Alice's RSA private key
- $H : \{0,1\}^k \mapsto \{0,1\}^m$ a public random function

Encryption of an m-bit message M:

Step 1. Generate a random k-bit number $r < n$.

Step 2. Compute $C = (s \| t)$ where $s \equiv r^e \pmod{n}$ and $t = H(r) \oplus M$.

Decryption of ciphertext C:

Compute $M \equiv H(s^d \pmod{n}) \oplus t$.

Prove that this cryptosystem is not IND-CCA secure.

Hint: To mount her CCA, Mallory gets to choose two plaintexts and submit one ciphertext for encryption. Almost any two plaintexts M_1, M_2 will do. Let

$$C = (s \| t) = (r^e \pmod{n} \;\|\; H(r) \oplus M_i) \quad \text{where } i = 1 \text{ or } 2$$

be the encryption of one them; Mallory needs to ascertain whether $i = 1$ or $i = 2$. Mallory now chooses as her ciphertext $C' = (s \| t \oplus M_1)$ and obtains its decryption (be sure to chose M_1 such that $C' \neq C$; that's the only restriction on the plaintexts). Prove that Mallory can with *certainty* (i.e. probability 1) and in polynomial time detect whether C is an encryption of M_1 or M_2.

Written Problems for MATH 318 only

Problem 6 — An attack on RSA with small decryption exponent, 25 marks

This problem explores an attack on RSA with small d due to Wiener.

Preliminaries. Let r be a positive rational number and write $r = a/b$ with $a, b \in \mathbb{N}$. Let $q_0, q_1, \ldots q_m \in \mathbb{N}$ be the sequence of quotients produced by applying the Euclidean Algorithm to the numerator and denominator of r:

$$\begin{aligned}
a &= q_0 b + r_0\,, & 0 &< r_0 < b\,,\\
b &= q_1 r_0 + r_1\,, & 0 &< r_1 < r_0\,,\\
r_0 &= q_2 r_1 + r_2\,, & 0 &< r_2 < r_1\,,\\
&\vdots \\
r_{m-3} &= q_{m-1} r_{m-2} + r_{m-1}\,, & r_{m-1} &= \gcd(a,b)\,,\\
r_{m-2} &= q_m r_{m-1} + r_m\,, & r_m &= 0\,.
\end{aligned}$$

Recall the familiar sequences

$$\begin{array}{lllllll}
A_{-2} = 0, & A_{-1} = 1, & A_i = q_i A_{i-1} + A_{i-2} & \text{for } 0 \le i \le m,\\
B_{-2} = 1, & B_{-1} = 0, & B_i = q_i B_{i-1} + B_{i-2} & \text{for } 0 \le i \le m.
\end{array}$$

The quotients A_i/B_i $(0 \le i \le m)$ are called the *convergents* of r because they oscillate around and converge toward r, with $A_m = a$, $B_m = b$, and hence $A_m/B_m = r$. In fact, the following theorem (which you may use without proof) asserts that any rational number sufficiently close to r must occur as one of the convergents:

Theorem. Let $r = \dfrac{a}{b} \in \mathbb{Q}$ and let $\dfrac{A}{B} \in \mathbb{Q}$ be a fraction in lowest terms such that $\left| r - \dfrac{A}{B} \right| < \dfrac{1}{2B^2}$. Then $A = A_i$ and $B = B_i$ for some $i \in \{0, 1, \ldots, m\}$.

Now back to RSA. Let $n = pq$ where p, q are odd primes satisfying $q < p < 2q$ (these inequalities are reasonable as p and q are usually assumed to have the same bit size). Let e, d be integers with $1 < e, d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$. Let $k \in \mathbb{Z}$ satisfy $ed = 1 + k\phi(n)$ and suppose that d is small compared to n; specifically,

$$d < \frac{\sqrt[4]{n}}{\sqrt{6}}.$$

(a) (5 marks) Prove that $1 \le k < d$ and $\gcd(d, k) = 1$.

(b) (3 marks) Prove that $2 \le n - \phi(n) < 3\sqrt{n}$.

(c) (4 marks) Use parts (a) and (b) to prove that $0 < kn - ed < 3d\sqrt{n}$.

(d) (4 marks) Conclude from part (c) that $0 < \dfrac{k}{d} - \dfrac{e}{n} < \dfrac{1}{2d^2}$.

(e) (3 marks) Let $q_0, q_1, \ldots, q_m$ be the quotients obtained when applying the Euclidean algorithm to e and n, and let A_i, B_i be the associated sequences as defined above. Use the theorem from above to prove that $k = A_i$ and $d = B_i$ for some $i \in \{1, 2, \ldots, m\}$.

(Note that $i = 0$ is impossible here even though the theorem allows it in principle. This is because $e < n$ forces $q_0 = 0$, so $A_0 = 0$ and $B_0 = 1$. Since $k > 0$ by part (a), we cannot have $k/d = A_0/B_0$.)

(f) (6 marks) Use part (e) to devise a procedure for finding d and factoring n efficiently. Explain why your procedure works and why it is efficient, i.e. argue that the number of computation steps performed by your procedure is small.

Problem 7 — Universal forgery attack on the El Gamal signature scheme, 12 marks)

Recall that for the El Gamal signature scheme, a user Alice produces her public and private keys as follows:

Step 1. Selects a large prime p and a primitive root g of p.
Step 2. Randomly selects x such that $0 < x < p-1$ and computes $y \equiv g^x \pmod{p}$.

Alice's public key: $\{p, g, y\}$
Alice's private key: $\{x\}$

Alice also fixes a public cryptographic hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_{p-1}$.

Alice signs a message $M \in \{0,1\}^*$ intended for Bob as follows:

Step 1. Selects a random integer $k \in \mathbb{Z}_{p-1}^*$.
Step 2. Computes $r \equiv g^k \pmod{p}$, $0 \le r < p$.
Step 3. Solves $ks \equiv [H(M\|r) - xr] \pmod{p-1}$ for $s \in \mathbb{Z}_{p-1}^*$.
Step 4. Signs the message M with the pair (r, s).

Bob verifies Alice's signature (r, s) to the message M as follows:

Step 1. Obtains Alice's authentic public key $\{p, g, y\}$.
Step 2. Verifies that $1 \le r < p$; if not, reject.
Step 3. Computes $v_1 \equiv y^r r^s \pmod{p}$ and $v_2 \equiv g^{H(M\|r)} \pmod{p}$.
Step 4. Accepts the signature if and only if $v_1 = v_2$.

The following is a universal forgery attack against a variant of this scheme that hashes only the message M, but does not include the random number r in the argument of the hash function in step 3 of the signature generation. More specifically, the second component of a signature is generated as a solution k to

$$ks \equiv H(M) - xr \pmod{p-1} ,$$

and a signature (r, s) to a message M is accepted if and only if

$$y^r r^s \equiv g^{H(M)} \pmod{p} .$$

Suppose Eve has intercepted a signature (r, s) to a message M generated by Alice, such that $\gcd(H(M), p-1) = 1$. Eve proceeds as follows:

Step 1. Chooses any $M' \in \{0,1\}^*$.
Step 2. Computes $H(M)$ and $H(M')$.
Step 3. Solves $H(M)u \equiv H(M') \pmod{p-1}$ for u via the Extended Euclidean Algorithm.
Step 4. Computes $S \equiv su \pmod{p-1}$.

Step 5. Computes $R \equiv rup - r(p-1) \pmod{p(p-1)}$.

(a) (4 marks) Prove that $R \equiv ru \pmod{p-1}$, and conclude that $y^R \equiv y^{ru} \pmod{p}$.

(b) (4 marks) Prove that $R^S \equiv r^{su} \pmod{p}$.

(c) (4 marks) Prove that (R, S) is a valid signature to the message M'. In other words, given r, s and M, Eve can generate a a valid signature (R, S) to *any* message M' that will be accepted as coming from Alice.

Remark: The quantity R will almost certainly exceed p, so this attack can be foiled if a verifier checks that $r < p$ and rejects signatures for which r exceeds p. A better fix is to include r in the argument of the hash function as presented in class (Pointcheval 1996).

Programming Problem for CPSC 418 only

Problem 8 — Secure file transfer with prior key agreement, 37 marks

Don't be daunted by the long description of this problem! Most of it is very clear specifications, including those for the autograder, to make your life easier.

Overview. Transport Layer Security (TLS) is a security protocol which aims to provide end-to-end communication security over networks. It provides both privacy and data integrity. While TLS has many steps, our focus will be on the *cipher suite*, a set of algorithms that add cryptographic security to a network connection. There are typically four components to a cipher suite:

- Key Exchange Algorithm
- Authentication algorithm (signature)
- Bulk Encryption Algorithm (block cipher and mode of operation)
- Message Authentication Algorithm

Cipher suites are specified in shorthand by a string such as

SRP-SHA3-256-RSA-AES-256-CBC-SHA3-256.

- This implies SRP-SHA3-256 as the key exchange algorithm, RSA signatures for authentication, AES-256-CBC for encryption and SHA3-256 as the MAC (used as HMAC).
- SRP is the protocol implemented in Assignment 2. Please replace the former hash algorithm SHA-256 with SHA3-256 (see cryptography library).
- Using SHA3-256 as the MAC implies the use of HMAC with SHA3-256 (see cryptography library).

In a *TLS handshake*, upon connection two parties automatically negotiate TLS settings, including the cipher suite to use. Through an exchange of messages the two parties verify each other and establish a session key. In this assignment, the two parties will be a server and client. The server will authenticate itself to the client by means of a *certificate*, and if successful, will derive a shared session key. The two parties are then able to use the session key to exchange encrypted and authenticated messages.

In reality, a certificate is an electronic document which contains a public key and information about the owner of the key. The certificate must be signed by a *trusted authority* to verify the contents of the certificate. For us, the certificate will merely be a *name* concatenated with a *public key*, concatenated with the signature of a *trusted third party* (TTP).

Problem specifications. For this assignment, assume that the Client and Server have negotiated SRP-SHA3-256-RSA-AES-256-CBC-SHA3-256 as the TLS cipher suite.[2] At a high-level, your task is to implement a toy version of the TLS handshake using the cipher suite specified above. The protocol will have two parties a Client and Server, each run by the commands

```
python3 Client.py filename1          python3 Server.py filename2
```

where `filename1` is the name of a file that will be encrypted and sent to the server and `filename2` is the name of the server's decrypted output file. For testing purposes, we will assume the sent file is **small**, i.e. less than 1 MB. Our simplified version of the TLS handshake proceeds as follows:

(a) The Client first registers itself to the server as in Assignment 2 (see registration step).

(b) The Client sends its username I to the Server.
   - First encode I as a byte array in 'utf-8' format, I_{bytes}. Then encode the length of I_{bytes}, denoted $len(I)$ into a 4-byte array in big-endian. Send the concatenation $len(I) \,||\, I_{bytes}$.

(c) The Server sends a *certificate* consisting of the server's name and public key as well as a signature of these two values by a Trusted Third Party (details are described in the next section).
   - The server encodes its name in a byte array S in 'utf-8' format.
   - Encode the length of S in a 4-byte array $len(S)$ and concatenate it along with the server's public key Server_PK. Note that the Server's public key should consist of the modulus N concatenated with the public exponent e, each encoded as 128-byte big-endian numbers.
   - Finally, append the TTP's *signature* TTP_SIG (obtained at an earlier time) to this byte array. The signature TTP_SIG will be encoded in 128-bytes big-endian. The Server's certificate should be of the form $len(S) \,||\, S \,||\,$ Server_PK $||$ TTP_SIG.
   - Upon receiving the certificate, the Client should verify the signature of the TTP.

(d) Initiate the SRP protocol from Assignment 2 (the protocol, not the registration) with the following modifications:
   - Instead of the client sending A as plaintext, they will send $Enc(A)$, the RSA encryption of A under the Server's public key.
   - Upon receiving $Enc(A)$ the server will decrypt to obtain A.
   - In case you did not implement the following check in Assignment 2: the server should ensure that the value A is not congruent to 0 under the SRP modulus and abort otherwise (it may be fun to think about why).
   - The remainder of the protocol proceeds as n Assignment 2 (derive the shared key and verify).

(e) With the shared key derived, the client encrypts and authenticates the file using the shared key and the symmetric algorithms specified in the suite (Note, Bob's protocol from Assignment 1 will help with this).

[2]With TLS 1.3, there has been a move towards using *authenticated encryption schemes* such as AES-GCM. For pedagogical reasons, we will look at a more classic cipher-suite.

(f) Have the server decrypt and verify the tag, creating a new file called `PTXT` that contains the decrypted message.

Requirements.

(a) You must implement your own version of the RSA parameter/key generation, encryption, decryption, signature generation and verification. *You should have a separate function for each, 5 in total.* For efficiency we have chosen relatively small parameter sizes: The server and TTP will pick separate RSA primes p and q to be 512 bit safe primes (see Assignment 2 for how to generate these). This means that the modulus N will be 1024 bits (128 bytes). See the course notes for the specifics of the RSA encryption and signature schemes.

(b) All other primitives should make use of the *cryptography library*.

(c) Alongside the Client and Server, you will need a TTP, which should be run via the command

```
python3 TTP.py
```

- Upon execution, the TTP should first obtain RSA primes p and q of size 512-bits and calculate $N = pq$. It should then create an RSA key pair for itself.
- Afterwards, the TTP should open a socket with specified port 31802 and IPV4 address 127.0.4.18 and listen.
- Upon a connection, the TTP should wait to receive one of two messages: "REQUEST SIGN" or "REQUEST KEY".
- Upon receipt of "REQUEST SIGN", the TTP should next receive a byte array consisting of 3 (concatenated) pieces of information: a 4-byte encoding of the length of *name*, a utf-8 encoded *name*, and an RSA public key PK encoded as a 256-byte array.
  - The TTP should proceed to hash the byte array (name || PK) using **SHA-512** to obtain a 512 bit value t, store it, and then hash t a second time, to produce t'. Interpret the byte array $t||t'$ as big-endian integer, and reduce it by the TTP's RSA modulus. The purpose of this is to make full use of the RSA message space.
  - Finally, compute the RSA signature on this value, then send N concatenated with the signature to the requester.
  - Note that N and the TTP-signature should each be encoded as 128-byte numbers in big endian.
- When receiving "REQUEST KEY", the TTP should simply send its modulus N concatenated to its public key. For simplicity, have the TTP close the socket connection after this communication.

(d) When your server script is run, prompt the user for a *server name*, then create an RSA key pair for the server. The server should connect to the TTP and send the message "REQUEST SIGN".

(e) Upon receiving the signature from the TTP, the Server can now create a socket connection on port 31803, IPV4 address 127.0.4.18 and begin listening for a client.

(f) When the Client program is executed, it should connect to the TTP and send "REQUEST KEY". Upon receiving the public key, store it, and connect to the Server and proceed with the derivation of a shared key.

Autograder requirements. For the purposes of the autograder, *whenever* a value is computed please print out a statement with the following format:

- “Server: Server_PK = 1234...”

here the word ‘Server’ is replaced by the relevant party (Client or TTP), ‘Server_PK’ is replaced by the relevant variable, and 1234.. is replaced by the corresponding value. This *includes* secret values (just think of them as debug statements).

- **When sending a value through a socket**, please print a statement with the following format:

  Server: Sending Server_N = <*hex value of N*>

  When receiving a value via socket, please interpret the value first, then print a statement with the format

  Client: Receiving Server_N = 1234 . . .

- **Naming conventions:** For printing names, please use the naming convention Client_name, Server_name and len(Client_name), len(Server_name). Do not use a different name for byte encodings. For RSA parameters p, q, N, use Server_N, TTP_N, Server_q etc. SRP parameters are the same as in Assignment 2. For the public and private keys, please use the notation Server_PK, Server_SK, where PK stands for public key, and SK stands for secret key. Denote the TTP signature by TTP_SIG. For the encrypted version of A, use Enc(A).

Example: For the TTP we would expect a print statement for the following values:

- Each of p and q as well as the value N.
- Each of the public and private RSA signature values e and d.
- The signature of the server information after it has been computed.
- The received message from any connections (“REQUEST SIGN” or “REQUEST KEY”).
- Any values received and sent, interpreted as described above.

Submission:

Please submit a separate README file in text format. Your description must include the following:

(a) A list of files you have submitted that pertain to the problem and a short description of each file.

(b) A list of what is implemented in the event that you are submitting a partial solution or a statement that the problem is fully solved.

(c) A list of what is not implemented in the event that you are submitting a partial solution,

(d) A list of known bugs or a statement that there are no known bugs.

You may use your own code from previous assignments as part of your solution, or make use of the solution files from Assignments 1 and 2.

Bonus Problem for Everyone

Problem 9 — Columnar transposition cryptanalysis, 10 marks

Decrypt the following ciphertext that was encrypted using a *columnar transposition* cipher. Show all your work; this includes source code if you used programming. Answers without satisfactory explanation and documentation of how they were obtained will receive no credit. Neither will answers obtained by simply running columnar transposition decryption from an online crypto applet website on the ciphertext.

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

```
EPAHR ELNFO CPOIO ASROS MWIWA SOTAV TIOES INDED DXTAT
YTNTU OMEPC ASFCR LTORN MOGRA SAORO SLLAH AMGAB OGTEW
DOSME ILOOO DRFTA TVABT RHDER DEXPH NOYIP WIITO CEDON
WIRTU LMRNX LWAIN OVTUH LIONN XARLU SOTSS RNNMI EGERH
AATCE FRMOS NS
```

Hint: The word “cavalry” appears in the plaintext.