CPSC 418 / MATH 318 — Introduction to Cryptography

ASSIGNMENT 3

Due: **Thursday, Dec. 8, 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.

Exception. *For **Problem 6** (the MATH 318 specific problem) and **Problem 7** (the CPSC 418 specific problem), **you may optionally partner with one other student and work on the solutions together.*** Your partner must be enrolled in the same course section. Specifically, both parters must be enrolled in either CPSC 418 or MATH 318; partnerships between a CPSC 418 student and a MATH 318 student are not allowed due to the bonus credit policy.

For Problem 6, write your partner's name and student ID at the beginning of your submission, right after your own name. For Problem 7, write your partner's name and student ID in the README file. *Both partners should submit identical solutions*, so Gradescope can record grades for both authors of the solution. Feel free to use Piazza's "Search for team mates" feature (see post @5). Note that only teams of up to two students are allowed.

Problems 1-5 are worth 62 marks and are required for both CPSC 418 and MATH 318 students.

Problem 6 is worth 38 marks and is for MATH 318 students only; CPSC 418 students may do these problems for limited extra credit.

Problem 7 is worth 38 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.

Problem 6 requires typesetting Legendre and Jacobi symbols. To facilitate this, include at the beginning of your assignment file, before the `\begin{document}` command, the two lines

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

The command `$\Leg{a}{n}$` will produce the typeset output $\left(\frac{a}{n}\right)$, which is much easier than producing a fraction with large parentheses around it.

The LaTeX template provided with this assignment already includes this command. If you are using your own template, be sure that you copy these lines *verbatim*; the easiest is to copy and paste them right from this PDF file.

Written Problems for CPSC 418 and MATH 318

Problem 1 — A modified variant of the Diffie-Hellman protocol (6 marks)

Consider the following modification of the Diffie-Hellman protocol.

Set-up:

- All participants agree on a public prime p and a primitive root of p.
- Each participant generates their own random exponent u_i with $1 < u_i < p-1$ and computes $U_i \equiv g^{u_i} \pmod{p}$.

The public parameters g, p as well as the numbers U_i are stored in a public database. Each participant keeps their exponent u_i secret.

Suppose Alice and Bob wish to establish on a shared cryptographic key. Denote Alice's database entry by $X \equiv g^x \pmod{p}$ and Bob's database entry by $Y \equiv g^y \pmod{p}$. They proceed as follows:

Key Agreement Protocol

1) Alice generates a secret exponent a with $1 < a < p-1$, computes $A \equiv g^a \pmod{p}$ and sends A to Bob.
   Bob generates a secret exponent b with $1 < b < p-1$, computes $B \equiv g^b \pmod{p}$ and sends B to Alice.
2) Alice looks up Bob's public database entry Y and computes $K \equiv B^x Y^a \pmod{p}$.
   Bob looks up Alice's public database entry X and computes $K \equiv A^y X^b \pmod{p}$.

The key shared between Alice and Bob is K. The man-in-the-middle attack on ordinary Diffie-Hellman discussed in class no longer works here as long as the database is tamper-proof (of course any attacker with the ability to hack into the database can simply replace Bob's database entry by her own and impersonate Bob).

a. (2 marks) Formally prove that Alice and Bob compute the same quantity K in step 2.

b. (4 marks) Recall the Diffie-Hellman problem whose solution breaks the original Diffie-Hellman protocol:

   Given p, g, $A \equiv g^a \pmod{p}$ and $B \equiv g^b \pmod{p}$, compute $g^{ab} \pmod{p}$.

   Suppose Eve is able to solve the Diffie-Hellman problem efficiently for any inputs p, g, A and B. Explain how she can use this capability when eavesdropping on Alice and Bob to find any key generated by Alice and Bob using the modified Diffie-Hellman protocol described above.[1]

[1] You may *not* assume here that Eve has the capability of extracting discrete logarithms; this is a stronger assumption. Remember that an algorithm for solving the discrete logarithm problem solves the Diffie-Hellman problem, and it is easy to see that it also makes finding keys for this variant very simple. But the reverse direction is unknown; that is, it is unknown whether anyone with the capability of discovering Diffie-Hellman keys or keys for this modified variant can use this ability to compute discrete logarithms.

Problem 2 — Properties of cryptographic hash functions (12 marks)

Recall the two properties of collision resistance and pre-image resistance that a cryptographic hash function needs to satisfy. In class, we asserted that these properties do not imply each other. This problem introduces two hash functions that confirm this fact; one is pre-image resistant but not collision resistant, the other is collision resistant but not pre-image resistant.

a. Let $h : \{0,1\}^* \to \{0,1\}^n$ be a pre-image resistant hash function. Define a new hash function $H : \{0,1\}^* \to \{0,1\}^n$ via

$$H(M) = h(M') \ ,$$

where $M' \in \{0,1\}^*$ is the bit string obtained by replacing the last bit of M by 0. That is, if the last bit of M is 0, then $M' = M$, whereas if the last bit of M is 1, then M' agrees with M except for the last but which is changed from 1 in M to 0 in M'.

(i) (4 marks) Formally prove that H is pre-image resistant.
Hint: Proof by contradiction works nicely here. Specifically, prove that if you can find a pre-image of some $x \in \{0,1\}^n$ under H, then you can also find a pre-image of $x \in \{0,1\}^n$ under h, contradicting the pre-image resistance of h. Also, don't overthink this; this is a really easy problem.

(ii) (2 marks) Prove that H is not collision resistant by exhibiting an explicit example of a collision for H. Explain your example.

b. Let $h : \{0,1\}^* \to \{0,1\}^n$ be a collision resistant hash function. Define a new hash function $H : \{0,1\}^* \to \{0,1\}^{n+1}$ via

$$H(M) = \begin{cases} 1\|0^n & \text{if } M = 0, \\ 0\|h(M) & \text{otherwise,} \end{cases} \ ,$$

where, as usual, "$\|$" denotes string concatenation and $1\|0^n$ is the string of length $n+1$ that begins with a 1, followed by n 0's.

(i) (4 marks) Formally prove that H is collision resistant. *Hint:* Go with proof by contradiction again. Specifically, prove that if you can find a collision for H, then you can also a collision for h, contradicting the collision resistance of h.

(ii) (2 marks) Prove that H is not pre-image resistant by exhibiting an explicit example of a string $x \in \{0,1\}^{n+1}$ for which it is easy to find a pre-image under H. Explain your example.

Problem 3 — CFB-MAC (6 marks)

Consider a block cipher whose plaintexts and ciphertexts are bit strings of length $n \in \mathbb{N}$, where encryption using a key K is denoted by E_K. Recall the the message authentication code CBC-MAC, where the MAC of a message M is the last ciphertext block when encrypting M using CBC mode:

CBC-MAC

1) Pad M with zeros so the length of the padded string is a multiple of the block length n.
2) Split the padded string into L blocks $M_1, M_2, \ldots, M_L$ where each block M_i has length n.

3) Put $C_0 = 0^n$ (the string of n zeros).
4) For $1 \leq i \leq L$, compute $C_i = E_K(M_i \oplus C_{i-1})$.
5) CBC-MAC$(M) = C_L$.

Similarly, define a message authentication code CFB-MAC based on using the same block cipher in CFB mode:[2]

CFB-MAC

1) Pad M with zeros so the length of the padded string is a multiple of the block length n.
2) Split the padded string into L blocks $M_1, M_2, \ldots, M_L$ where each block M_i has length n.
3) Put $C'_0 = M_1$ (the first message block)
4) For $1 \leq i \leq L-1$, compute $C'_i = M_{i+1} \oplus E_K(C'_{i-1})$.
5) CFB-MAC$(M) = E_K(C'_{L-1})$.

Now Let M be any message, K a key, C_i $(1 \leq i \leq L)$ the sequence of ciphertext blocks obtained when computing CBC-MAC(M) and C'_i $(1 \leq i \leq L-1)$ the sequence of ciphertext blocks obtained when computing CFB-MAC(M).

a. (4 marks) Use induction on i to prove that $C_i = C'_i \oplus M_{i+1}$ for $0 \leq i \leq L-1$.

b. (2 marks) Prove that CBC-MAC(M) =CFB-MAC(M).

Problem 4 — A variant of RSA (10 marks)

Let a, b be integers that are not both zero. The *least common multiple* of a, b, denoted $\text{lcm}(a, b)$, is the unique positive integer satisfying the following properties:

(A) a divides $\text{lcm}(a, b)$ and b divides $\text{lcm}(a, b)$.
(B) If L is an integer such that a divides L and b divides L, then $\text{lcm}(a, b)$ divides L.

You may use without proof the fact that the integer $\text{lcm}(a, b)$ is unique for any a, b and satisfies

$$\gcd(a, b) \cdot \text{lcm}(a, b) = ab \ .$$

Now consider the following variant of RSA in which the quantity $\phi(n) = (p-1)(q-1)$ in the key generation procedure is replaced by $\text{lcm}(p-1, q-1)$.

Set-up: In order to generate their public/private key pair, each user does the following.

1) Generates two large distinct primes p, q.
2) Computes $n = pq$ and $l = \text{lcm}(p-1, q-1)$.
3) Selects an element $e \in \mathbb{Z}_l^*$.
4) Solves the congruence $ed \equiv 1 \pmod{l}$ for $d \in \mathbb{Z}_l^*$.

The user's public and private keys are (e, n) and d, respectively.

Encryption and decryption proceed exactly as in ordinary RSA. That is, plaintexts and ciphertexts are elements in $\mathbb{Z}_n^*$, the encryption of a plaintext $M \in \mathbb{Z}_n^*$ is $C \equiv M^e \pmod{n}$, and the decryption of a ciphertext $C \in \mathbb{Z}_n^*$ is $M \equiv C^d \pmod{n}$.

[2] CFB-MAC can be defined more generally with any intitial value C'_0, in which case it differs from CBC-MAC. But for this problem, we specifically pick $C'_0 = M_1$.

a. (6 marks) Formally prove that this variant of RSA works. Specifically, prove that

$$(M^e)^d \equiv M \pmod{n}$$

for all $M \in \mathbb{Z}_n^*$.

b. (4 marks) Suppose p and q are safe primes, so $p = 2p' + 1$ and $q = 2q' + 1$ with primes p', q'. Formally prove how an attacker who knows n and $l = \text{lcm}(p-1, q-1)$ can find factor n, i.e. find p and q.

Problem 5 — A probabilistic encryption scheme (28 marks)

In this problem, you will investigate a *homomorphic* probabilistic public key cryptosystem.

Set-up: In order to generate their public/private key pair, each user does the following.

1) Generates two large distinct primes p, q such that p does not divide $q - 1$ and q does not divide $p - 1$;
2) Computes $n = pq$ and $\phi(n) = (p-1)(q-1)$.

The user's public and private keys are n and $\phi(n)$, respectively. The user also precomputes the modular inverse of $\phi(n)$ modulo n, i.e. the element $f \in \mathbb{Z}_n^*$ such that

$$f\phi(n) \equiv 1 \pmod{n} \ .$$

Plaintexts are elements in $\mathbb{Z}_n$ and ciphertexts are elements in the set

$$Z_{n^2}^* = \{a \in \mathbb{Z} \mid 0 \leq a < n^2 \text{ and } \gcd(a, n^2) = 1\} \ .$$

Encryption: To encrypt a plaintext $M \in \mathbb{Z}_n$, the sender does the following.

1) Generates a random element $r \in \mathbb{Z}_{n^2}^*$.
2) Computes and sends the ciphertext $C \equiv (n+1)^M r^n \pmod{n^2}$.

Decryption: To decrypt a ciphertext $C \in \mathbb{Z}_{n^2}^*$, the receiver does the following.

1) Computes $a \equiv C^{\phi(n)} \pmod{n^2}$.
2) Computes the integer $b = (a-1)/n$.
3) Computes the plaintext $M \equiv bf \pmod{n}$.

a. In this part, you will prove some mathematical preliminaries that are required for the proof that this cryptosystem works.

   (i) (2 marks) Using the properties of Euler's phi function covered in class, prove that $\phi(n^2) = n\phi(n)$.

   (ii) (3 marks) Prove that $\gcd(n, \phi(n)) = 1$. This establishes that $\phi(n)$ has an inverse modulo n, so the quantity f defined above exists.

   (iii) (2 marks) Prove that the number b in step 2 of the decryption procedure is an integer.

   (iv) (3 marks) Prove that $(n+1)^k \equiv kn + 1 \pmod{n^2}$ for all positive integers k.

b. (6 marks) Prove correctness of this cryptosystem. Specifically, prove that if C is the encryption of any plaintext $M \in \mathbb{Z}_n$, obtained via the encryption procedure above, then applying the decryption procedure above to C yields M.

c. (3 marks) For any plaintext $M \in \mathbb{Z}_n$, denote by $E(M, r)$ the encryption of M under public key n using the random number r for encryption. Prove that for any two plaintexts $M_1, M_2 \in \mathbb{Z}_n$, any elements $r_1, r_2 \in \mathbb{Z}_n^*$, and any positive integer k, we have

$$E(M_1, r_1) \cdot E(M_2, r_2) = E(M_1 + M_2, r_1 r_2) \quad \text{and} \quad E(M_1, r_1)^k = E(kM_1, r_1^k) \ .$$

In other words, if we apply decryption to the above identities, we see that decrypting a product of two encryptions yields the sum of the plaintexts, and decrypting a power of an encryption yields the corresponding multiple of the plaintext.[3]

d. Consider the following chosen ciphertext attack on the above system where an attacker wishes to obtain the decryption M of a ciphertext C and proceeds as follows.

1) Generates $r \in \mathbb{Z}_{n^2}^*$ with $r^n \not\equiv 1 \pmod{n^2}$.
2) Computes $C' \equiv Cr^n \pmod{n^2}$ (this is the chosen ciphertext).
3) Obtains the decrytion M' of C'.

(i) (2 marks) Prove that $C' \neq C$, so C' is a valid choice of ciphertext to get decrypted in step 3.

(ii) (3 marks) Prove that $M' = M$. In other words, the decryption obtained in step 3 of the CCA is precisely the plaintext that the attacker is after, and thus the attack is successful.

e. (4 marks) An element $z \in \mathbb{Z}_{n^2}^*$ is an *n-th power residue modulo n^2* if and only if there exists an element $x \in \mathbb{Z}_{n^2}$ such that $x^n \equiv z \pmod{n^2}$. Prove that a ciphertext C is the encryption of the plaintext $M = 0$ if and only if C is an n-th power residue modulo n^2. So decryption for this cryptosystem can be thought of as an "n-th power residue modulo n^2 detector".[4]

Written Problem for MATH 318 only

Problem 6 — An RSA-like public key cryptosystem (38 marks)

In this problem, you will investigate a public key cryptosystem that is similar to RSA but has the advantage that an adversary is able to break the system *if and only if* she is able to factor the modulus. The price to pay for this potential added security is the restriction on the format of plaintexts: they must have Jacobi symbol 1 with respect to n. In addition, the system is extremely vulnerable to a simple chosen ciphertext attack that does not work for RSA.

[3] Cryptosytems satisfying these two properties are called *linearly homomorphic*. The first property in particular makes this system very suitable for electronic voting, especially for yes/no elections. Every user encrypts their vote, which is 1 for "yes" and 0 for "no", using the election administrator's public key n. The administrator multiplies all the encrypted votes together and decrypts the product. By the first property, this decryption is the sum of all the votes, which is just the number of "yes" votes (note that the administrator need not know the random numbers used by the voters to encrypt their votes). This voting procedure preserves secrecy of individual votes and anonymity of voters. Moreover, the election administrator only needs to compute a modular product and perform one decryption, which is much more efficient than decrypting each vote individually.

[4] Detecting n-th power residues modulo n^2 is generally believed to be computationally intractable when n is the product of two distinct large primes.

In part(c), we will extend the plaintext space to be all of $\mathbb{Z}_n^*$ (like RSA), but at the additional cost of more complicated encryption and decryrption procedures as well as reduced efficiency: in addition to a modular exponentiation (like in RSA), encryption in the extended scheme requires the evaluation of a Jacobi symbol.

Set-up: In order to generate their public/private key pair, each user does the following.

1) Generates two large primes p and q with $p \equiv 3 \pmod 4$ and $q \equiv 3 \pmod 4$.
2) Computes $n = pq$ and $f = \phi(n)/4$.
3) Selects $e \in \mathbb{Z}$ with $1 < e < f$ and $\gcd(2e, f) = 1$
4) Solves the congruence $2ed \equiv 1 \mod f$ for $d \in \mathbb{Z}$, $1 \leq d < f$.

The user's public and private keys are (e, n) and d, respectively. Here, as always, $\phi()$ denotes the Euler phi function.

The plaintext space is the set of all $M \in \mathbb{Z}_n^*$ such than $\left(\frac{M}{n}\right) = 1$. Encryption of such a plaintext M with public key (e, n) is given by

$$C \equiv M^{2e} \pmod n \ .$$

Decryption of a ciphertext C is given as

$$M \equiv C^d \pmod n \ .$$

Unfortunately, the 2 in the encryption exponent introduces the same abiguity as squaring of ordinary integers. Specifically, it makes it impossible to distinguish M from $n - M$ (which is just $-M \pmod n$) after decryption. Now that one way to distinguish these two numbers is by their parity: one of them is odd and the other is even (since n is odd). So the encrypter simply needs to send the parity bit distinguishing M from $n-M$, along with the ciphertext. Specifically, encryption and decryption proceed as follows:

Encryption: To encrypt a plaintext $M \in \mathbb{Z}_n^*$ with $\left(\frac{M}{n}\right) = 1$, the sender does the following.

1) If M is even, put $b = 0$, else put $b = 1$.
2) Compute $C \equiv M^{2e} \pmod n$.
3) Send (C, b).

Decryption: To decrypt a pair (C, b) , the receiver does the following.

1) Compute $M' \equiv C^d \pmod n$.
2) If M' is even and $b = 0$ or M' is odd and $b = 1$, put $M = M'$, else put $M = n - M'$.

Remark: The restrictions $\left(\frac{M}{n}\right) = 1$ and $p \equiv q \equiv 3 \pmod 4$ ensure the condition

$$\left(\frac{M}{p}\right)^{(q-1)/2} = \left(\frac{M}{q}\right)^{(p-1)/2} \tag{1}$$

for all $M \in \mathbb{Z}_n^*$ which is crucial for this cryptosystem to work; you will use this in part (a) (iii).

a. (Correctness, 14 marks)

(i) (2 marks) Prove that $(p-1)/2$ and $(q-1)/2$ are odd.

(ii) (3 marks) Prove that $\left(\frac{M}{p}\right) = \left(\frac{M}{q}\right)$ for all $M \in \mathbb{Z}_n^*$. Then use part (a) (i) to conclude that property (1) above holds.

(iii) (4 marks) Prove that $M^f \equiv \pm 1 \pmod{n}$ for all $M \in \mathbb{Z}_n^*$ with $\left(\frac{M}{n}\right) = 1$.

Hint: Specifically, prove $M^f \equiv e \pmod{n}$ where $e = \left(\frac{M}{p}\right)^{(q-1)/2} = \left(\frac{M}{q}\right)^{(p-1)/2}$ is the quantity in (1).

(iv) (5 marks) Use part (a) (iii) to prove correctness of this cryptosystem. Specifically, prove that if C is the encryption of any plaintext $M \in \mathbb{Z}_n*$ with $\left(\frac{M}{n}\right) = 1$, obtained via the encryption procedure above, then applying the decryption procedure above to C yields M.

b. (Security, 12 marks)

Just as for RSA, it is evident that if an attacker can factor n, i.e. find the primes p and q, she can proceed as the designer and compute f and the private key d, thereby breaking the scheme. In this part, you will prove the "converse": if an attacker can break the scheme via a successful chosen ciphertext attack, she can factor n.

Consider the following chosen ciphertext attack on the above system where an attacker attempts to factor n.

1) Chooses any $y \in \mathbb{Z}_n^*$ such that $\left(\frac{y}{n}\right) = -1$.
2) Computes $C \equiv y^2 \pmod{n}$ (this is the chosen ciphertext).
3) Obtains the decryption M of C.
4) Computes $x \equiv M^e \pmod{n}$.
5) Computes $\gcd(x-y, n)$.

(i) (2 marks) Prove that $\left(\frac{-1}{n}\right) = 1$.

(ii) (2 marks) Prove that $\left(\frac{M}{n}\right) = 1$ and hence $\left(\frac{x}{n}\right) = 1$, with M as given in step 3 and x as in step 4 of the CCA.

(iii) (2 marks) Prove that $x^2 \equiv y^2 \pmod{n}$.

(iv) (3 marks) Use parts (b) (i)-(ii) to prove that $x \not\equiv y \pmod{n}$ and $x \not\equiv -y \pmod{n}$.

Hint: Compare $\left(\frac{x}{n}\right)$ with $\left(\frac{y}{n}\right)$ and $\left(\frac{-y}{n}\right)$.

(v) (3 marks) Finally, the grand finale: use parts (b) (ii)-(iv) to prove that $\gcd(x-y, n) = p$ or q, so the attack above factors n.

c. (Expanding the plaintext space to all of $\mathbb{Z}_n^*$, 12 marks)

The idea behind this part is to choose the primes p and q such that for any $M \in \mathbb{Z}_n^*$, if $\left(\frac{M}{n}\right) = -1$ (so M is not a valid plaintext) then $\left(\frac{2M}{n}\right) = 1$ (so $2M \pmod{n}$ is a valid plaintext). In addition to sending along a parity bit, the encrypter then also needs to send another bit indicating whether $\left(\frac{M}{n}\right) = -1$ (in which case $2M$ is encrypted instead of M). You will prove in part (c) (ii) that an appropriate choice is

$$p \equiv 3 \pmod{8}\ , \qquad q \equiv 7 \pmod{8}\ ,$$

so we assume these congruence conditions from here on.

Consider the following modified encryption procedure, where the plaintext space is now all of $\mathbb{Z}_n^*$:

Encryption: to encrypt a plaintext $M \in \mathbb{Z}_n^*$, proceed as follows.

1) Compute $\left(\frac{M}{n}\right)$.
2) If $\left(\frac{M}{n}\right) = 1$, put $M_0 = M$ and $b_0 = 0$, else put $M_0 \equiv 2M \pmod{n}$ and $b_0 = 1$.
3) If M_0 is even, put $b_1 = 0$, else put $b_1 = 1$.
2) Compute $C \equiv M_0^{2e} \pmod{n}$.
3) Send (C, b_0, b_1).

Decryption: to decrypt a triple (C, b_0, b_1), proceed as follows.

1) Compute $L \equiv C^d \pmod{n}$.
2) If L is even and $b_1 = 0$ or L is odd and $b_1 = 1$, put $L_0 = L$, else put $L_0 = n - L$.
3) If $b_0 = 0$, put $M = L_0$, else put $M \equiv \frac{n+1}{2} L_0 \pmod{n}$

Note that the quantity $(n+1)/2$ in step 3 of decryption is simply the inverse of 2 modulo n as $1 \leq (n+1)/2 < n$ and $2 \cdot (n+1)/2 = n + 1 \equiv 1 \pmod{n}$.

(i) (3 marks) Prove that $\left(\frac{2}{n}\right) = -1$.
Remark: This shows that $y = 2$ is a valid choice in step 1 of the CCA above.

(ii) (2 marks) Prove that $\left(\frac{M_0}{n}\right) = 1$, with M_0 as in step 2 of the modified encryption procedure.

(iii) (4 marks) Prove that $L_0 = M_0$, with L as in step 2 of decryption and M_0 as in step 2 of decryption.

(iv) (3 marks) Use part (c) (iii) to prove correctness of this modified version of this cryptosystem. Specifically, prove that if C is the encryption of any plaintext $M \in \mathbb{Z}_n*$, obtained via the modified encryption procedure above, then applying the modified decryption procedure above to C yields M.

The scheme in thus problem is known as the *Rabin-Williams* cryptosystem, after its inventors. In 1979, Michael Rabin introduced the idea of squaring in the context of a signature scheme, which an adversary can break, i.e. forge signatures via a chosen message attack, if and only if she can factor the modulus. In 1980, Hugh C. Williams, who is a former math professor at U Calgary (and long retired by now) proposed the public key cryptosystem above which merges Rabin's squaring approach with RSA.

Programming Problem for CPSC 418 only

Problem 7 — Vaccine Passports (38 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. It's currently August 15th, 2006. More than three years ago, SARS managed to escape from Scarborough Grace Hospital in Ontario, mutate to become more virulent, and establish widespread community spread across the globe. Governments have engaged in a rollercoaster of lockdowns and "opening up" ever since, with tens of millions of deaths. They've also thrown as much funding as they can at developing a vaccine, and the resulting flurry of development created one in just less than three years, a new record. As of June 11th, 2006, Ontario residents could get a SARS-COV-1 vaccine shot and 80% of the province has done so. So it was a bit of a shock when Ontario Health told your team the province was heading into a new health crisis. Politicians and the media have over-hyped the vaccine as a silver bullet, ignoring that it has only a 70% effective rate after two weeks and that the latest research suggests that effectiveness rapidly dwindles after a year. That's not enough to achieve herd immunity. Worse, the public has no appetite to return to the lockdown rollercoaster, and there is increasing levels of violence at anti-vaccine protests. Add in the oncoming Fall semester of school, and there's potential for the health system to be completely overrun.

They and other civil servants have come up with a compromise: vaccine passports. Society will be split in two, with the vaccinated enjoying more freedom to move thanks to their immunity while those who refuse to vaccinate are isolated to prevent more mass casualties. The length of time since the last shot needs to be tracked, to deal with the issue of waneing immunity. Naturally, these passports need to be both tamper-resistent and convenient to use.

Your team has come up with a suitable system, based around a cutting-edge technology from Japan called a "QR code". Encryption will be used to obfuscate the details stored in these codes, a MAC guards against data corruption, and a combination of a digital signature and keyed MAC provide authenticity. Your job is to create the first implementation of this system, with code that will be used in web browsers, portable hand scanners, and a web server.[5]

Passport Format. Like the first assignment, the data portion can be divided into three segments: the plaintext, the nonce, and a tag.

The plaintext is 96 bytes long. The upper four bits of the first byte contain the number of vaccines this person has had. Values greater than 15 are capped at 15. The lower four bits of the first byte are the upper portion of a 12 bit number, the lower eight bytes being contained in the second byte of the cyphertext. This value represents the week the person had their last vaccine shot, measured in weeks since June 11th, 2006.[6] Any value greater than 4,095 is capped at 4,095. If the person hasn't had a vaccine shot, this value is set to 4,095.

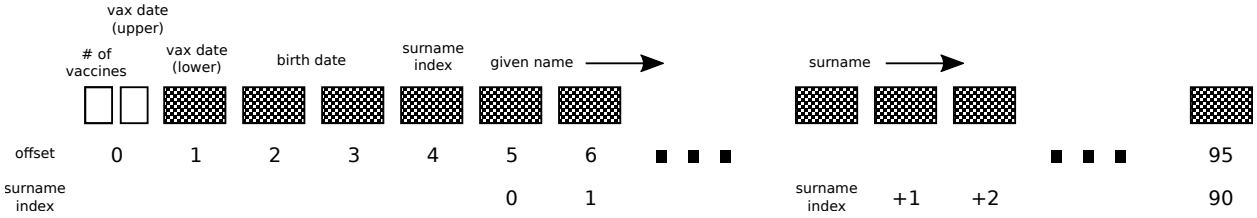
The next two bytes represent the birthdate of the person, measured in days since January 1st, 1880. This is in big endian format, and capped to 65,535 days.

The next 92 bytes represent the given name and surname, stored in that order. For maximum flexibility, both of them are contained in the same field, so that short names can make room for longer ones. To specify where the surname starts, the first byte of the text section (or the fifth byte of the overall plaintext) contains the byte index to its location, relative to the start of the text portion (not the plaintext, and excluding the index byte; see the diagram). The characters are encoded in UTF-8. If the combined length of both names is less than 91 bytes after encoding, the encoder picks a random but valid index for the surname and pads any excess space with zero byte characters. If the combined length is greater than 91 then characters are iteratively removed from the longest of the two names until their combined UTF-8 encoded size is 91 bytes or less; in

[5] This is a greatly simplified variation of the system currently used by many Canadian provinces.

[6] For instance, six days after that epoch is zero weeks while eight is one.

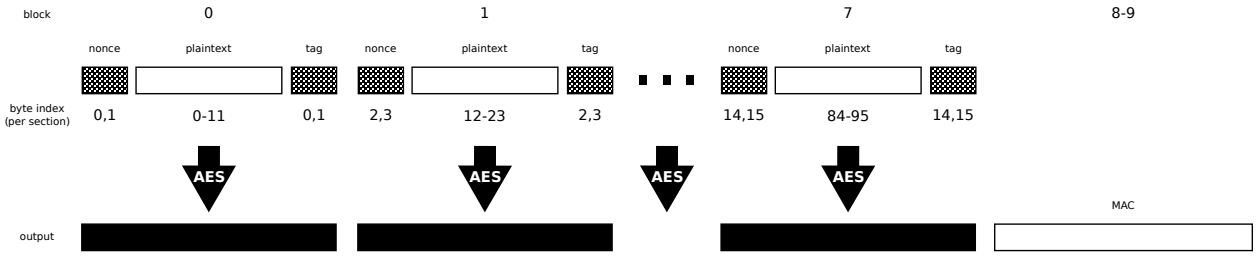
the event of a tie, remove a character from the surname.



The nonce is 16 bytes long. It has the same role as the initialization vector from the first assignment, though it is used quite differently. The tag is 16 bytes long, and is defined as $H(P(\text{"OH SARS SECOND VERIFY"})||P(\texttt{hash\_key})||\texttt{nonce}||\texttt{plaintext}||0x10)$, where $H()$ is SHAKE256 with a digest size of 16 bytes and $P()$ appends zero bytes to the input until the resulting output is a multiple of 136 bytes (or appending nothing if it already is a multiple).[7]

Encryption is done slightly differently than Assignment 1. All three segments of the data are divided into eight blocks. The first tag block is appended to the first plaintext block, and the combination appended to the nonce block. This is repeated, resulting in eight blocks that each consist of a mix of nonce, plaintext, and tag. Each block is encrypted with AES-256 using `enc_key`.[8]

After encryption, $H(\texttt{nonce}||\texttt{plaintext}||\texttt{tag})$ is calculated and appended to the cyphertext, resulting in a 159 byte[9] sequence.[10]



Signature. The passport itself contains little sensitive data. It is meant to be presented to someone with a scanner along with a piece of government-issued ID; the person scans the QR code, decrypts it, and verifies the name and birthdate match those of the ID. Rather than protect data from unauthorized viewing, this system is primarily designed to be extremely difficult to forge.

Symmetric encryption can only provide this via a trusted third party, which cannot be guaranteed when anyone can hold a scanner that may not have cellular access (let alone internet access). Instead, the 159 byte sequence is signed with an RSA keypair (N_{RSA}, d) that is kept private, resulting in the 319 bytes contained within the QR code. The public keypair (N_{RSA}, e) is shared to anyone who requests it, and embedded within each scanner. This allows anyone to verify the passport was almost certainly generated by an approved device and is therefore authentic.

Communication. There is an additional security concern. To request your vaccine passport in as convenient a form as possible, people will be allowed to request one via a web form by supplying their Ontario Health Number, birthdate, and when they had their last vaccine shot. This opens up

[7] This algorithm is based on KMAC, which implies the "$0x10$" portion is the length of the digest encoded as a single byte. It is worth thinking about how KMAC relates to HMAC.

[8] Aside: we've discouraged the use of ECB encoding in class, yet invoked it here. Try to figure out why we're not hypocritical about this.

[9] This is not a typo, the just-mentioned SHAKE256 tag is 31 bytes long instead of 32.

[10] This approach is considered inferior to what was done in Assignment 1, incidentally. See Bellare (2008).

two attack vectors, the most obvious of which is a person-in-the-middle attack to link a person's name, birthdate, and OHN for identity theft.

A more subtle one involves brute force. An OHN consists of ten digits, and yet the population of Ontario is approximately 12,200,000. As a result, the odds of guessing a valid OHN are no lower than one in a thousand. As everyone rushed to get a vaccine once it was available, guessing the date someone received a vaccine shot is easier than it otherwise would be. The demographics of Ontario are well known. In sum, an attacker can simply flood the passport website with requests in the hope that they guess correctly, at which point they have a name, birthdate, and OHN.

To combat those attacks, you'll use the security protocol introduced in Assignment 2. It is a toy version of Transport Layer Security (TLS), used to negotiate a shared secret to protect against person-in-the-middle attacks. A proof-of-work system was added to discourage request flooding.

Implementation. There are three main components to this system: the QR code server, which issues and fully verifies said codes; the scanner, which partially verifies QR codes; and the web client, which can request a QR code from the server or ask that it be verified.[11] On startup, the QR code server does the following:

- Generates two RSA primes p and q of size 640-bits and computes $N_{\text{RSA}} = pq$, ensuring the result is 1280 bits in size.
- Generates its own RSA key-pair (N_{RSA}, e) and (N_{RSA}, d).[12]
- Generates a 512-bit safe prime N_{DH} as well as a primitive root g of said safe prime.

The scanner already has access to (N_{RSA}, e), and the web client can retrieve those values from the QR code server.

Registration. The registration portion of A2 is carried forward, though some parameters have been adapted. `username` is now `uuid`, and `password` is now `secret`; both of these are now computer-generated and fixed at 32 bytes in length, but otherwise have the same behaviour.[13]

Protocol. Initiate the protocol from Assignment 2, but 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.[14]
- Upon receiving $Enc(A)$ the server must 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 in Assignment 2 (derive the shared key and verify).
- With the shared key derived, the web client appends the OHN as a five byte number, three zero bytes, birthdate (days since January 1st, 1880, as two bytes), four zero bytes, and a date

[11] As `hash_key` is stored exclusively on the server, the scanner cannot verify the tag. This is by design.

[12] In a real implementation, these values would be loaded from read-only storage instead of recalculated, as otherwise vaccine passwords would become invalid after a server reset.

[13] This does make the single byte representing the length of the username redundant. Rather than force you to rewrite the registration functions with this byte removed, we'll instead keep the byte so no alterations are required.

[14] This is not the ideal way to do this, for instance consider the case where a client sends a vaccine passport signature instead of $Enc(A)$. Using separate keys for encryption and signing would add significant runtime without teaching you anything new, however, and the TAs can't come up with an exploit that depends on RSA key reuse.

of vaccination (days since June 11th, 2006, as two bytes) in that order.[15] The client encrypts that via AES-256 and uses the following 32 byte value as the key.

$$H(P(\text{"OH SARS KEYEXTEND 1"})||P(N_{\text{RSA}}||e)||K_{\text{client}}||\texttt{0x20}) \tag{2}$$

where $H()$ is SHAKE256 with a 32 byte digest and $P()$ is the padding function defined earlier.

- The server decrypts that value, converts it to a byte sequence, and splits that sequence back into its three components. It looks up those values in a database, and if they match an existing entry the server generates the QR code value outlined above.
- The server takes the first 32-bytes of the shared key as the encryption key, and the next 32 bytes as the key for MAC. It uses a modified version of the encryption algorithm from Assignment 1 to encrypt the QR code. It returns the length of the encrypted QR code to the client, encoded as four big-endian bytes, then returns the actual QR code.
- The web client decrypts the QR code and verifies it was transmitted successfully.

The Assignment 1 encryption algorithm is modified in the following way. Instead of allowing an arbitrary hash function to be used for encryption, AES-256 is invoked instead. Instead of allowing an arbitrary block ID generator, it always starts at 0 and counts upwards. The MAC is now calculated as $H(P(\text{"OH SARS QR MAC"})||P(\texttt{MAC\_key})||\texttt{IV}||\texttt{cyphertext}||0x20)$ where $H()$ is SHAKE256 with a 32 byte digest.

Problem Your task is to complete the template program

`vaccine_passport.py`

which will become your group's proof of concept for Ontario Health. With approval, a team of programmers will take your template and use it to develop a web interface, scanner software, and the QR code server. You may pair up with one other student to write your code, if you wish, **but be sure to mention your partner's name in the README!** As with Assignment 2, all messages over the socket should be echoed to standard output by both the sending and receiving party. All prior permitted and forbidden functions carry forward. For AES-256, use the implementation in the `pycryptodome` library.[16] You are expected to implement your own versions of the RSA related functions. *You may not use functions related to RSA from another library for this exercise.* See the course notes for the specifics of the RSA encryption and signature schemes.

Specifications. Fill in the empty functions in the template program `vaccine_passport.py`. Since this is built on top of previous assignments, you are permitted to import any and all functions from the reference code for the prior two. The template even explicitly imports the relevant ones for you. Once complete, you should be able to perform the full TLS handshake between the web client and the QR code server by running something like

```
python3 vaccine_passport.py --request_QR
python3 vaccine_passport.py --QR_server
python3 vaccine_passport.py --quit_server
```

You may also combine these actions with one invocation of `vaccine_passport.py`, as with Assignment 2. In addition, you will be able to simulate the function of the scanner by running

[15] Yes, you will have to check this; otherwise, how would you know the value was correctly decoded?

[16] Other implementations are out there, but they either aren't a core Python module or are not installed on the auto-grader.

```
python3 vaccine_passport.py --verify_QR --hash_key "" --code [HEX STRING]
```

where `[HEX STRING]` is the hexadecimal output generated by `--request_QR`. As usual, there are additional command-line options to change key parameters from the defaults and control which actions are taken. In detail, the new functions you'll need to fill in fall into the following categories:

a. Low-level Functions:

   (i) `RSA.modulus(...)`, which generates p and q for the specified bit length.

   (ii) `RSA.keypair(...)`, which generates e and d.

   (iii) `RSA.sign(...)` and `RSA.verify(...)`, which respectively sign a value and verify a signature.

   (iv) `RSA.encrypt(...)` and `RSA.decrypt(...)`, which respectively encrypt and decrypt a value.

   (v) `encode_name(...)`, which converts a given name and surname into a byte sequence as outlined prior.

   (vi) `gen_plaintext(...)`, which uses `encode_name(...)` to generate the plaintext as outlined above.

   (vii) `pseudoKMAC(...)`, a helper function which carries out the modified KMAC algorithm outlined above.

   (viii) `interleave_data(...)`, which interleaves the `nonce`, `plaintext`, and `tag` as described above.

   (ix) `encrypt_data(...)` and `decrypt_data(...)`, which perform the modified version of Assignment 1's encryption algorithm as outlined above.

b. High-level Functions:

   (i) `create_passport(...)`, which combines all of the above functions to generate a QR code.

   (ii) `verify_passport(...)`, which verifies the given passport and returns the decoded data.

   (iii) `request_passport(...)`, which duplicates the actions the web client makes when requesting a passport.

   (iv) `retrieve_passport(...)`, which duplicates the actions the QR code server makes when generating a QR code for the web client.

As per Assignment 2, some of your functions will need to translate between integers and byte object parameters. All numbers are again converted to `bytes` via network byte order. 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

```
vaccine_passport.py
```

If you've spread your code across multiple source files, submit all of them. The auto-grader will have copies of the reference code for A1 and A2 with the same file name, so there is no need to submit those files. 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:

a. A list of files 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.
e. If you worked with a partner on your answer, include their name and a one-sentence statement that you worked with them.

You may use the solution files from Assignments 1 and 2.