Homework 1 – Quadratic Residues, Diffie-Hellman Cryptography

Cryptography and Security 2024

- ◇ You are free to use any programming language you want, although Python/SAGE is recommended.
- ◇ Put all your answers **and only your answers** in the provided `[id]-answers.txt` file where `[id]` is the student ID.[1] This means you need to provide us with all `Q`-values specified in the questions below. Personal files are to be found on Moodle under the feedback section of Parameters HW1.
- ◇ **Please do not put any comment or strange character or any new line** in the submission file and do **NOT** rename the provided files.
- ◇ Do **NOT** modify the `SCIPER`, `id` and `seed` headers in the `[id]-answers.txt` file.
- ◇ **Submissions that do not respect the expected format may lose points.**
- ◇ We also ask you to submit your **source code**. This file can of course be of any readable format and we encourage you to comment your code. Notebook files are allowed, but we prefer if you export your code as normal textual files containing Python/SAGE code. If an answer is incorrect, we may grant partial marks depending on the implementation.
- ◇ Be careful to always cite external code that was used in your implementation if the latter is not part of the public domain and include the corresponding license if needed. Submissions that do not meet this guideline may be flagged as plagiarism or cheating.
- ◇ Some plaintexts may contain random words. Do not be offended by them and search them online at your own risk. Note that they might be really strange.
- ◇ Please list the name of the **other** person you worked with (if any) in the designated area of the answers file.
- ◇ Corrections and revisions may be announced on Moodle in the "News" forum. By default, everybody is subscribed to it and does receive an email as well. If you decided to ignore Moodle emails, we recommend that you check the forum regularly.
- ◇ The homework is due on Moodle on **November 20, 2024** at 23h59.

[1] Depending on the nature of the exercise, an example of parameters and answers will be provided on Moodle.

Exercise 1 Identity-Based Encryption

Identity-Based Encryption (IBE) is a public-key encryption system where any string (e.g., email address) can serve as a public key. This eliminates the need for certificates as used in traditional public-key systems. A trusted authority possessing a master secret key generates private keys for users based on their identities.

An IBE scheme consists of four algorithms:

- **Setup**$(\lambda) \to (pp, \mathsf{msk})$: Generates a public parameter pp and master secret key msk
- **Extract**$(\mathsf{msk}, \mathsf{id}) \to sk_{id}$: Generates secret key for identity id using the master secret key msk.
- **Encrypt**$(pp, id, m) \to c$: Encrypts message m to identity id
- **Decrypt**$(pp, sk_{id}, c) \to m$: Decrypts ciphertext c using secret key for id

Prerequisites: Legendre and Jacobi Symbols

Legendre Symbol: Let p be an odd prime and a be an element of $\mathbf{Z}_p$.

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p} \\ 1 & \text{if } a \text{ is a quadratic residue in } \mathbf{Z}_p^* \\ -1 & \text{if } a \text{ is a quadratic non-residue in } \mathbf{Z}_p^* \end{cases}$$

Essentially,

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

Jacobi Symbol: Let $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \dots p_k^{\alpha_k}$ be a composite odd number where p_i is a prime number for all $i \in [k]$. The Jacobi symbol $(\frac{a}{n})$ is defined as:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \left(\frac{a}{p_2}\right)^{\alpha_2} \dots \left(\frac{a}{p_k}\right)^{\alpha_k}$$

where $(\frac{a}{p_i})$ is the Legendre symbol.

Key properties of the Jacobi symbol:

- if a is a quadratic residue in $\mathbf{Z}_n^*$, $\left(\frac{a}{n}\right) = 1$.
- $\left(\frac{a}{n}\right) = -1$ does not necessarily imply a is a quadratic non-residue in $\mathbf{Z}_n^*$.
- For odd n: $\left(\frac{1}{n}\right) = 1$
- For $a \equiv b \pmod{n}$: $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
- $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
- $\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \pmod{8} \\ -1 & \text{if } n \equiv \pm 3 \pmod{8} \end{cases}$

Cocks' IBE

Cocks' IBE is based on the quadratic residuosity assumption modulo a composite number. In this exercise, you will implement the scheme described in Figure 1. You will notice that we need a encode function that maps the id to $\mathbf{Z}_N^*$. This function is provided to you as a black box as `encode(id, N)` under `utils.sage` (available on Moodle).

$\mathsf{IBE.KeyGen}(\lambda)$

$p \leftarrow\!\$\ \mathsf{GenPrime}(\lambda/2)$
$q \leftarrow\!\$\ \mathsf{GenPrime}(\lambda/2)$
$N \leftarrow p \cdot q$
$\mathsf{msk} \leftarrow (p, q)$
return (N, msk)

$\mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

$\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{encode}(\mathsf{id})^{1/2} \bmod N$

$\mathsf{IBE.Encrypt}(\mathsf{N}, \mathsf{id}, \mathsf{m})$

$t \leftarrow\!\$\ \mathbf{Z}_N^*$
while $\left(\frac{t}{N}\right) \neq m$:
  $t \leftarrow\!\$\ \mathbf{Z}_N$
$c \leftarrow \frac{4t^2 + \mathsf{encode}(\mathsf{id})}{4t} \bmod N$
return c

$\mathsf{IBE.Decrypt}(\mathsf{N}, \mathsf{sk}_{\mathsf{id}}, \mathsf{c})$

$m \leftarrow \left(\frac{c + \mathsf{sk}_{\mathsf{id}}}{N}\right)$
return m

Figure 1: Cocks' IBE for square roots.

Since the message space for the Cocks' IBE is rather small (i.e. {-1, 1}). You will be encrypting the plaintext one bit at a time. You are supplied with two functions under `utils.sage` (available on Moodle) to aid with this process:

- `to_bits(s)` which takes a string `s` and returns a Python list of -1s and 1s.
- `to_string(bit_list)` which takes a Python list of -1s and 1s `bit_list`, returns a string.

Note that for all questions in this exercise, the format of the ciphertext is a Python list of individual ciphertext corresponding to each bit of the plaintext.

Question 1.1

- Note that for this question, we provide an additional list of values for the variable t sampled during the encryption. This is because we would like to have deterministic answers. Hence, we have already sampled the t value for each bit you are going to encrypt. Therefore you should not implement the sampling of t yourself.
- You are given a public parameter $\mathsf{Q1a\_N}$, a plaintext $\mathsf{Q1a\_m}$, a list of values for t $\mathsf{Q1a\_t}$, an id $\mathsf{Q1a\_id}$. Report the resulting ciphertext under $\mathsf{Q1a\_c}$.

Question 1.2

You are given a public parameter $\mathsf{Q1b\_N}$, master secret key $\mathsf{Q1b\_msk}$, an id $\mathsf{Q1b\_id}$ and a ciphertext $\mathsf{Q1b\_c}$. Report the resulting plaintext from decrypting $\mathsf{Q1b\_c}$ under $\mathsf{Q1b\_m}$

Question 1.3

Consider the following modification to the key generation shown in Figure 2.

$\mathsf{IBE.KeyGen}(\lambda)$

1 : $p \leftarrow\!\!\$\ \mathsf{GenPrime}(\lambda/2)$
2 : $q \leftarrow\!\!\$\ \mathsf{NextPrime(p)}$
3 : $N \leftarrow p \cdot q$
4 : $\mathsf{msk} \leftarrow (p, q)$
5 : **return** (N, msk)

Figure 2: Cocks' IBE for square roots with modified KeyGen.

You are given a public parameter Q1c_N, an id Q1c_id and a ciphertext Q1c_c. Report the resulting plaintext from decrypting Q1c_c under Q1c_m

Cubic Cocks' IBE

Now we slightly modify the Cocks' IBE to work with cubic roots instead of square roots. The resulting scheme is shown in Figure 3.

$\mathsf{IBE.KeyGen}(\lambda)$

1 : $p \leftarrow\!\!\$\ \mathsf{GenPrime}(\lambda/2)$
2 : $q \leftarrow\!\!\$\ \mathsf{GenPrime}(\lambda/2)$
3 : **while** $gcd(3, p-1) \neq 1$
4 : $\quad \vee\ gcd(3, q-1) \neq 1$:
5 : $\quad p \leftarrow\!\!\$\ \mathsf{GenPrime}(\lambda/2)$
6 : $\quad q \leftarrow\!\!\$\ \mathsf{GenPrime}(\lambda/2)$
7 : $N \leftarrow p \cdot q$
8 : $\mathsf{msk} \leftarrow (p, q)$
9 : **return** (N, msk)

$\mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

1 : $\mathsf{sk_{id}} \leftarrow \mathsf{encode(id)}^{1/3} \bmod N$

$\mathsf{IBE.Encrypt}(\mathsf{N}, \mathsf{id}, \mathsf{m})$

1 : $t \leftarrow\!\!\$\ \mathbf{Z}_N$
2 : $a \leftarrow \mathsf{encode(id)}$
3 : **while** $\left(\frac{t^3 + a}{N}\right) \neq m$:
4 : $\quad t \leftarrow\!\!\$\ \mathbf{Z}_N$
5 : $c \leftarrow \frac{t(t^3 - 8a)}{4(t^3 + a)} \bmod N$
6 : **return** c

$\mathsf{IBE.Decrypt}(\mathsf{N}, \mathsf{sk_{id}}, \mathsf{c})$

1 : $m \leftarrow \left(\frac{c + \mathsf{sk_{id}}}{N}\right)$
2 : **return** m

Figure 3: Cocks' IBE for cubic roots.

Question 1.4

You are given a public parameter Q1d_N, an id Q1d_id and a ciphertext Q1d_c. Report the resulting plaintext from decrypting Q1d_c under Q1d_m

Hint: This question requires a heavy amount of algebraic manupilation. A starting point could be to realize that:

$$\left(\frac{(t^3 + a)^{-1}}{N}\right) = \left(\frac{(t^3 + a)}{N}\right)^{-1} = \left(\frac{t^3 + a}{N}\right)$$

Exercise 2 Semi-Direct Discrete Logarithms in Cryptography

After the midterm, you should all be familiar with the concept of group action in cryptography (if not, refer to this link). This exercise aims to extend group actions cryptography by exploring the concept of *semi-direct discrete logarithms*. Let's start with a key definition:

Definition 1 (Holomorph)**.** Let G be a group with its automorphism set $\mathrm{Aut}(G)$. The **holomorphism** group $\mathrm{Hol}(G)$ is defined as the semidirect product of G with $\mathrm{Aut}(G)$, denoted by $G \rtimes \mathrm{Aut}(G)$. This means that it consists of ordered pairs from $G \times \mathrm{Aut}(G)$ with the following multiplication:

$$(g, \phi) \star (g', \psi) = (g\phi(g'), \phi \circ \psi),$$

with neutral element (e_G, id). We thus have that the exponentiation over $\mathrm{Hol}(G)$ is defined as $(g, \phi)^n = (g\phi(g) \cdots \phi^{(n-1)}(g), \phi^{(n)})$ and $\phi^{(n)} = \underbrace{\phi \circ \cdots \circ \phi}_{n \text{ times}}$.

The holomorph of G induces a natural group action:

$$\mathrm{Hol}(G) \circlearrowright G$$

$$(g, \phi) \star h = g\phi(h),$$

Now, restrict $\mathrm{Hol}(G)$ to a commutative structure. For any $(g, \phi) \in \mathrm{Hol}(G)$, our action induces the following map:

$$\Phi_{(g,\phi)} : \mathbb{Z}_d \times G \to G$$

$$\Phi_{(g,\phi)}(n, h) = (g, \phi)^{n-1} \star h = g\phi(g) \cdots \phi^{(n-2)}(g)\phi^{(n-1)}(h)$$

where $d = |G|$. This map has certain properties that allow the construction of a Diffie-Hellman Key Exchange scheme, which is described below:

Semi-Direct Diffie-Hellman Key Exchange

Alice		**Bob**
Pick $a \in \mathbb{Z}_d$		
$A = \Phi_{(g,\phi)}(a, g)$		
	$\xrightarrow{\text{A}}$	Verify $A \in G$
		Pick $b \in \mathbb{Z}_d$
		$B = \Phi_{(g,\phi)}(b, g)$
Verify $B \in G$	$\xleftarrow{\text{B}}$	
$K = A\phi^{(a)}(B)$		$K = B\phi^{(b)}(A)$
Return K		**Return** K

$$K = \Phi_{(g,\phi)}(a + b, g)$$

We now explore an application of this Semi-Direct Diffie-Hellman scheme using matrices. Consider $G = \mathrm{Mat}_{n\times n}(\mathbb{F}_p)$, the group of all $n \times n$ matrices over the finite field $\mathbb{F}_p$, where p is a "safe" prime. We choose $g = M$, a random matrix, and define ϕ is given by two fixed matrices H_1 and H_2 such that $H_i M \neq M H_i$. Our maps, ϕ_{H_1,H_2} are defined as follows:

$$\phi_{H_1,H_2}(T) = H_1 T H_2$$

Throuhout this exercice, the public parameters `Q2_p`, `Q2_n` and `Q2_M` are fixed. *You are given matrices in flatten mode and your answers should also be in flatten mode, i.e.*

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = [1, 2, 3, 4, 5, 6, 7, 8, 9].$$

Constructing the scheme

The goal of this exercise is to construct the Semi-Direct Diffie-Hellman Key Exchange. You are given the public parameters `Q2a_H1`, `Q2a_H2`, Alice's secret `Q2a_a` and Bob's public value `Q2a_B`. Using all these parameters, compute `Q2a_K` the shared key that correspond to the output of Semi-Direct Diffie-Hellman Key Exchange.

Hint 1:

- Note the addition notation induces that $\Phi_{(M,\phi)}(2, M) = M + H_1 M H_2$.
- To implement $\Phi_{(g,\phi)}(n, g)$, do not forget that $\phi \in \mathrm{Aut}(\mathrm{Mat}_{n\times n}(\mathbb{F}_q))$. This induces self similarity. For example:

$$\Phi_{(M,\phi)}(4, M) = M + \phi(M) + \phi^{(2)}(M) + \phi^{(3)}(M) = M + \phi(M) + \phi^{(2)}\big(M + \phi(M)\big)$$

Reducing the scheme

Let show that this scheme reduces to the discrete log problem over $\mathbb{F}_p$. Let $d = \det(H_1 H_2) \neq 0$, given `Q2b_H1`, `Q2b_H2` and `Q2b_A`, retrieve `Q2b_da` $= d^a \mod \mathbb{F}_p$.

Breaking the Scheme

To break this scheme, we will work with matrices of size n^2. Consider the following definitions:

Definition 2 (vectorisation)**.** For $M \in \mathrm{Mat}_{n\times n}(\mathbb{F}_p)$, define $\mathsf{vec}(M) \in \mathbb{F}_p^{1\times n^2}$ as the vector obtained by flattening M row-wise:

$$\mathsf{vec}(M)_{jn+i} = M_{i,j}, \quad 0 \leq i, j < n.$$

Definition 3. For $Y \in \mathrm{Mat}_{n\times n}(\mathbb{F}_p)$, define the matrix $\mathsf{L}(Y) \in \mathbb{F}_p^{n^2\times n^2}$ as:

$$\mathsf{L}(Y)_{jn+i,hn+g} = (H_1^g Y H_2^h)_{i,j}, \quad 0 \leq i, j, g, h < n.$$

Given matrices `Q2c_H1`, `Q2c_H2`, `Q2c_A` and `Q2c_B`, recover the shared key `Q2c_K`.

- *Hint 1: What is* $\mathsf{L}(H_1 M H_2)$ *compare to* $\mathsf{L}(M)$*?*
- *Hint 2: Find a vector* `t` *such that* $\mathsf{L}(M)\mathtt{t} = \mathsf{vec}(H_1^a M H_2^a)$*.*
- *Hint 3: Consider the Cayley-Hamilton Theorem.*

Conclusion: Constructing a robust group action Diffie-Hellman scheme is challenging.

Exercise 3 Privacy Preserving Neural Network

In this exercise we are going to create a very simple neural network that will compute its output in an encrypted manner (preserving the confidentiality of the inputs). Let's start by defining our neural network. For simplicity, our neural network will be composed of a single node called a neuron. Given fixed weights $(w_1, \ldots, w_n)$, for input $(x_1, \ldots, x_n)$ the output y of the neuron will be the output of an activation function f where the input to the activation function is the weighted sum of the neuron's inputs with an added bias b. See Figure 4 for a diagram that describes the neuron.

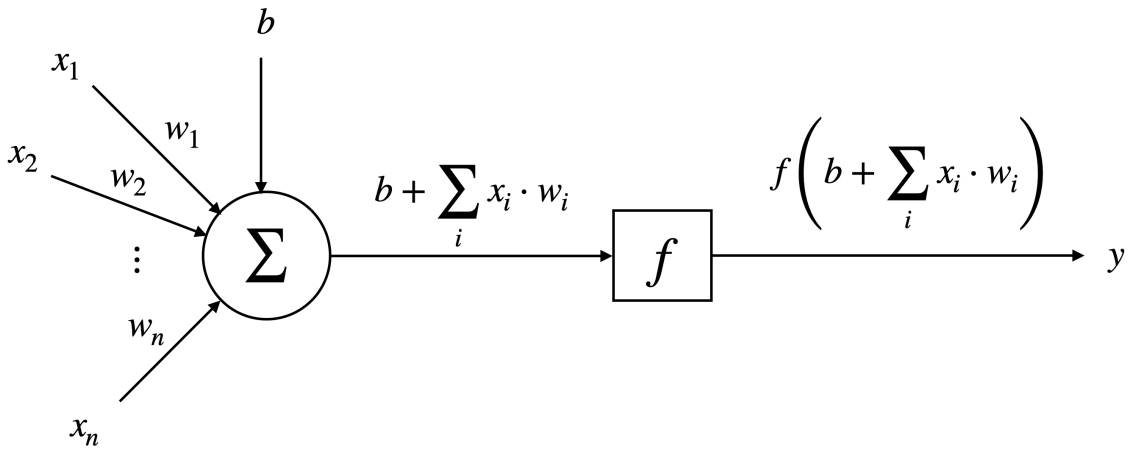

Figure 4: Neuron on input $(x_1, \ldots, x_n)$.

We recall a variant of ElGamal encryption from the course (with a group generator g and a prime modulus p) where the input first goes through an exponential. I.e. instead of encrypting m, we encrypt g^m. See Figure 5 for details. For compatibility with the cryptosystem we use, our neuron will operate modulo $p - 1$. Our activation function will be $f(x) = \frac{x}{1009}$.

$\mathsf{KeyGen}(1^\lambda)$		$\mathsf{Enc}(\mathsf{pk}, m, pp)$	
1 :	$(p, g) \leftarrow\!\$\ \mathsf{GroupGen}(\lambda)$	1 :	$\mathsf{parse}\ pp \to (p, g)$
2 :	$x \leftarrow\!\$\ \mathbf{Z}_p^*$	2 :	$r \leftarrow\!\$\ \mathbf{Z}_p^*$
3 :	$pp \leftarrow (p, g)$	3 :	$u \leftarrow g^r$
4 :	$\mathsf{pk} \leftarrow g^x$	4 :	$v \leftarrow \mathsf{pk}^r \cdot g^m$
5 :	**return** (x, pk, pp)	5 :	$C \leftarrow (u, v)$
		6 :	**return** C

Figure 5: ElGamal encryption with message in the exponent

Question 3.1

Given public parameters Q3a_pk, Q3a_p and Q3a_g, a list of $n+1$ ciphertexts Q3a_c $= (c_1, \ldots, c_n, cb)$ corresponding to the encryptions of individual inputs and the bias $(x_1, x_2, \ldots, x_n, b)$, a list of n weights Q3a_w, compute the ciphertext that would yield the neuron output when decrypted and return it under Q3a_cy. That is, Q3a_cy should be the encryption of $f(b+\sum_i x_i \cdot w_i)$.

Note that in this question the ciphertext Q3a_cy should be a Python tuple with two components.