Homework 5 - Commitment, Integrity, Authentication and Signatures

Cryptography and Security 2018

- You are free to use any programming language you want, although SAGE is recommended.
- Put all your answers **and only your answers** in the provided SCIPER-answers.txt file. This means you need to provide us with Q_{1a}, Q_{1b}, Q_{2a}, Q_{2b}, Q_3 and Q_4. You can download your **personal** files on `http://lasec.epfl.ch/courses/cs18/hw5/index.php`
- The answers Q_{1a} should be a list of integers, Q_{1b} should be a pair of strings in base64, Q_{2a} and Q_{2b} should be integers, Q_3 should be a hex string, and Q_4 should be an ASCII string. **Please provide nothing else. This means, we don't want any comment and any strange character or any new line** in the answers.txt file.
- 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.
- The plaintexts of most of the exercises contain some random words. Don't be offended by them and Google them at your own risk. Note that they might be really strange.
- If you worked with some other people, please list all the names in your answer file. We remind you that you have to submit your own source code and solution.
- We might announce some typos in this homework on Moodle in the "news" forum. 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.
- To connect to the server for Exercise 4, **you have to be inside the EPFL network** (for the php page you don't need to be inside the EPFL network). Use VPN if you are connecting from outside EPFL.
- Please contact us as soon as possible if a server is down.
- Denial of service attacks are **not** part of the homework and will be logged (and penalized).
- The homework is due on Moodle on **Thursday the 20th of December** at 22h00.

Exercise 1 Schnorr Signatures over Elliptic Curves

This exercise uses the same encoding library with the previous homework.

Field Elements and Curve Points.
Let $E = (a, b, G, p, q)$ denote an elliptic curve, i.e. $E = \{(x, y) \in \mathbb{F}_p^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$, where $\mathcal{O}$ denotes the neutral element. We use additive notation for groups. Let $G \in E$ be such that $|\langle G \rangle| = q$ for some prime q. Suppose $\lambda = |p| = |q|$ and λ is divisible by 8. Let $\{\texttt{0x00}, \texttt{0x01}, \ldots, \texttt{0xFF}\}$ denote the set of byte values, whose integer equivalents are $\{0, 1, \ldots, 255\}$ respectively. Let $|\cdot|$ be bitsize operator. The symbol $\perp$ is used for error. $\{0,1\}^\lambda$ denotes the set of bit strings whose size is exactly λ. We use **brainpoolP384r1** parameters for the elliptic curve.

We handle different elements as follows:

- $\mathsf{encode}_{\mathbb{F}} : \mathbb{F}_p \to \{0,1\}^\lambda$. Maps a field element x into a string of bytes $B_{\lambda/8-1} \ldots B_1 B_0$ such that $\sum_{i=0}^{\lambda/8-1} B_i \cdot 256^i = x$. Essentially, any field element $x \in \mathbb{F}_p$ is encoded to fixed-length string, that is of $\lambda/8$ characters just as in SAGE. Recall that SAGE considers strings to be byte arrays incidentally. Note that the definition requires encoding of any *small* element to be left padded with sufficient number of zeros.

- $\mathsf{decode}_{\mathbb{F}} : \{0,1\}^\lambda \to \mathbb{F}_p/\perp$. The inverse of $\mathsf{encode}_{\mathbb{F}}(\cdot)$. On input y, it returns a valid field element only if there exists $x \in \mathbb{F}_p$ whose encoding is y; **otherwise it raises error** and returns $\perp$ (your code does not necessarily need to return $\perp$ symbol, we use it to denote the occurrence of an error).

- $\mathsf{encode}_{\mathbb{Z}} : \mathbb{Z}_q \to \{0,1\}^\lambda$. Maps a ring element $x \in \mathbb{Z}_q$ into a string of bytes $B_{\lambda/8-1} \ldots B_1 B_0$ such that $\sum_{i=0}^{\lambda/8-1} B_i \cdot 256^i = x$.

- $\mathsf{decode}_{\mathbb{Z}} : \{0,1\}^\lambda \to \mathbb{Z}_q/\perp$. is defined as the inverse of $\mathsf{encode}_{\mathbb{Z}}(\cdot)$. On input y, it returns a ring element only if there exists $x \in \mathbb{Z}_q$ whose encoding is y; **otherwise it raises error**.

- $\mathsf{encode}_P : E \to \{0,1\}^{\lambda+8}$. Maps any point P of elliptic curve into a string of $(\lambda/8) + 1$ bytes. Let $B_{\lambda/8} \ldots B_1 B_0$ be the encoding of P,
  - if $P = \mathcal{O}$, then $B_i = \texttt{0x00}$ for all $i \in \{0, 1, \ldots, \lambda/8\}$.
  - if $P \neq \mathcal{O}$, then let (x, y) be the coordinates of P. Then $B_{\lambda/8} \ldots B_2 B_1 = \mathsf{encode}_{\mathbb{F}}(x)$. B_0 is set based on the parity of y, that is $B_0 = \texttt{0x0F}$ if y is odd, and $B_0 = \texttt{0xFF}$ if y is even.

- $\mathsf{decode}_P : \{0,1\}^{\lambda+8} \to E/\perp$. is the inverse of $\mathsf{encode}_P(\cdot)$. For decoding, any $(\lambda/8+1)$-byte string is valid only if there is a point P on E whose encoding returns the given input. For example, $(\texttt{0x00})^{\lambda/8}\texttt{0x10}$ should **raise error**.

- We can use arbitrary length of messages with this signature scheme thanks to the hash function. More precisely, we consider messages as multiple bytes of strings, (i.e. string of ASCII characters), who are handled by the hash function.

- Since field elements, elliptic curve points and messages can be encoded as strings, any string operation such as concatenation (denoted with $|$) or XOR-ing can be done with them.

```
$\underline{\mathsf{Kg}(\mathcal{G})}$:
  1: $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_q$
  2: $\mathsf{sk}_{\mathsf{en}} \leftarrow \mathsf{encode}_{\mathbb{Z}}(\mathsf{sk})$
  3: $\mathsf{vk} \leftarrow \mathsf{sk} \cdot G$
  4: $\mathsf{vk}_{\mathsf{en}} \leftarrow \mathsf{encode}_{\mathsf{P}}(\mathsf{vk})$
  5: $\textbf{return } (\mathsf{sk}_{\mathsf{en}}, \mathsf{vk}_{\mathsf{en}})$
```

```
$\underline{\mathsf{Sign}(\mathcal{G}, \mathsf{sk}_{\mathsf{en}}, m)}$:
  1: $\mathsf{sk} \leftarrow \mathsf{decode}_{\mathbb{Z}}(\mathsf{sk})$
  2: $k \leftarrow_{\$} \mathbb{Z}_q$
  3: $R \leftarrow k \cdot G$
  4: $R_{\mathsf{en}} \leftarrow \mathsf{encode}_{\mathsf{P}}(R)$
  5: $h \leftarrow \mathsf{SHA256}(m|R_{\mathsf{en}})$
  6: $s \leftarrow (k - \mathsf{sk} \cdot h) \bmod q$
  7: $s_{\mathsf{en}} \leftarrow \mathsf{encode}_{\mathbb{Z}}(s)$
  8: $\sigma \leftarrow (s_{\mathsf{en}}, R_{\mathsf{en}})$
  9: $\textbf{return } \sigma$
```

```
$\underline{\mathsf{Ver}(\mathcal{G}, \mathsf{vk}_{\mathsf{en}}, m, \sigma)}$:
  1: $\mathsf{vk} \leftarrow \mathsf{decode}_{P}(\mathsf{vk})$
  2: $\sigma \rightarrow (s_{\mathsf{en}}, R_{\mathsf{en}})$
  3: $R \leftarrow \mathsf{decode}_{P}(R_{\mathsf{en}})$
  4: $s \leftarrow \mathsf{decode}_{\mathbb{Z}}(s_{\mathsf{en}})$
  5: $h \leftarrow \mathsf{SHA256}(m|R_{\mathsf{en}})$
  6: $R' \leftarrow s \cdot G + h \cdot \mathsf{vk}$
  7: $\textbf{return } 1_{R'=R}$
```

Figure 1: sk, vk denotes signing, verifying key pair respectively; σ denotes the signature. For brevity, decoding of $\mathcal{G} \rightarrow (a, b, G, p, q)$ is removed from the code. All algorithms have access to $\mathcal{G}$. $1_{R'=R}$ means "1 if $R' = R$, and 0 otherwise".

- Let bitstring $h = h_{255} \ldots h_1 h_0$ be the output of hash function SHA256. Then we consider h as integer, i.e. $h = \sum_{i=0}^{255} h_i \cdot 2^i$.

Schnorr Signatures.
Schnorr signature scheme over curve E is a triplet of algorithms $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$ with arbitrary-length message domain as given in Figure 1.

We refer to $(\mathsf{vk}_{\mathsf{en}}, m, \sigma)$ (which is $(\mathsf{vk}_{\mathsf{en}}, m, (s_{\mathsf{en}}, R_{\mathsf{en}})))$ as a valid Schnorr tuple if feeding it to Ver yields 1. It is left implicit that, if Ver or Sign encounters a decoding error, then they immediately return 0 (which can be seen as special error/invalid symbol). In general, algorithms should check whether given input belongs to the proper domain, and abort on failure.

In your parameter file, you will find a list of such tuples given as $Q1_{list}$. Each element of the list is a Schnorr tuple $(\mathsf{vk}_{\mathsf{en}}, m, \sigma)$ and each component is encoded with base64. In other words, $Q1_{list}$ looks like [(base64, base64, (base64, base64)), ...]. You should implement the verification algorithm fully (including decoding errors) and decide which tuples are valid. Provide your answers in Q_{1a} as a list of integers, i.e. put 1 for valid tuples and 0 for invalid ones. The sequence is randomly generated, therefore it is possible to have all valid or all invalid tuples. We also provide a list of valid/invalid signature tuples so that can make sure you are using the encoding and SHA256 algorithms correctly. $Q1_{test\_list}$ contains a list of such tuples whose validity is encoded in list $Q1_{test\_list\_answers}$.

Then, you will also find a (secret) signing key $\mathsf{sk}_{\mathsf{en}}$ (as base64 encoded Q_{sk} on top of $\mathsf{encode}_{\mathbb{Z}}$), verification key $\mathsf{vk}_{\mathsf{en}}$ (as base64 encoded Q_{vk} on top of $\mathsf{encode}_{\mathsf{P}}$), and a message m (as base64 encoded Q_m). You are expected to implement signing algorithm and sign given message **with the random coins provided**, i.e. do not sample but use given k (as base64 encoded Q_k on top of $\mathsf{encode}_{\mathbb{Z}}$). Compute the signature σ and put it as $Q1_b$, which should look like a tuple (base64, base64). Do not forget to encode two elements of this tuples as described by signing algorithm, before applying base64 encoding.

Exercise 2 Trapdoor Overuse

Our crypto-apprentice was interested about Pedersen commitment, because he realized that the trapdoor makes the commitment breakable. He generated the parameters (p, q, g, h) and published them, and kept the trapdoor to himself. We recall the Pedersen Commitment over $\mathbb{Z}_p^*$:

Setup: Generate two primes p and q such that q has 160 bits, p has 1024 bits and $q \mid (p-1)$. Then find two elements $g, h \in \mathbb{Z}_p^*$ of order q (to have a trapdoor τ, one can pick $h = g^\tau$). The domain parameters are (p, q, g, h).

Commit: To commit to message m (which is an integer), pick a random $r \in \mathbb{Z}_q$ and compute $c = g^m \cdot h^r$.

Open: In order to open, provide (m, r) (or open to anything with the trapdoor!).

We simply refer to the tuple (c, m, r) as Pedersen tuple if c is actually the commitment with respect to (m, r).

In your parameter file, you will find a list Q_{2list} of tuples where each tuple contains three integers (c, m, r) (which are not necessarily correct Pedersen tuples). The public parameters are also given as $Q_{2p}, Q_{2q}, Q_{2g}, Q_{2h}$. Your first goal is to recover the trapdoor τ and provide it as Q_{2a} (make sure to provide $(\tau \mod q)$ as the canonical form).

Secondly, you are given a commitment Q_{2c} and a message Q_{2m} to which you need to provide a valid opening random coin r and put it in Q_{2b}, i.e. (Q_{2c}, Q_{2m}, Q_{2b}) is a valid commitment tuple. All given parameters are integers, so should your answer be.

Hint: You will find many c collisions in given list.

Exercise 3 Identity-based encryption with master decryption key

The identity-based encryption consists of four following algorithms

- $\mathsf{Setup}(1^\lambda) = (mk, params)$
- $\mathsf{Extract}(mk, ID, params) = S_{ID}$
- $\mathsf{Enc}(ID, m, params) = ct$
- $\mathsf{Dec}(S_{ID}, ct, params) = m$

and for any $mk, params$ which are generated by Setup, for any message m, and for any identity ID,

$$\mathsf{Dec}(\mathsf{Extract}(mk, ID, params), \mathsf{Enc}(ID, m, params), params) = m$$

Now, consider the following construction:

- $\mathsf{Setup}(1^\lambda)$: Pick two prime numbers p and q such that $q|(p \pm 1)$. Pick the finite field $K = \mathbb{F}_{p^2}$ and a supersingular elliptic curve $E(K)$ of cardinality $(p \pm 1)^2$. Then, compute q-torsion subgroup[1] $E[q]$ and the Weil pairing $e : E[q] \times E[q] \longrightarrow \mu_q$ where μ_q is the group of q-th roots of unity in K. Pick P and Q from $E[q]$ such that $|\langle P \rangle| = |\langle Q \rangle| = q$ and $P \notin \langle Q \rangle$, and pick a, b, c, d uniformly from $\mathbb{Z}_q^*$ until $\langle (1, a) \rangle$, $\langle (b, 1) \rangle$ and $\langle (c, d) \rangle$ are distinct subgroups of $\mathbb{Z}_q \times \mathbb{Z}_q$. Output the master key

  $$mk = (a, b, c, d, P, Q)$$

  and the public parameters

  $$params = (K, E, q, e, U, V, W)$$

  where $U = P + aQ$, $V = bP + Q$ and $W = cP + dQ$.

[1] In other words, the group of all points P such that $qP = O$, and this set is a group.

- $\mathsf{Extract}(mk, ID, params)$: Pick s uniformly from $\mathbb{Z}_q^*$. Compute and output

$$S_{ID} = s(d + ID)^{-1}P + (s + ab - 1)(c + b \cdot ID)^{-1}Q.$$

- $\mathsf{Enc}(ID, m, params)$: Output $\perp$ if $m \notin K^*$. Pick r_1 uniformly from $\mathbb{Z}_q^*$ and pick r_2 uniformly from K^*. Then, compute

$$\begin{aligned} ct_0 &= m \cdot r_2, \\ ct_1 &= r_1 \cdot ID \cdot V + r_1 \cdot W, \\ ct_2 &= r_2 + e(U, V)^{r_1} \end{aligned}$$

  and output $ct = (ct_0, ct_1, ct_2)$.

- $\mathsf{Dec}(S_{ID}, ct, params)$: Output

$$m = ct_0 \cdot (ct_2 - e(S_{ID}, ct_1))^{-1}.$$

In this construction, for any $mk, params$ which are generated by Setup, there exists a master decryption key S which satisfies

$$\mathsf{Dec}(S, \mathsf{Enc}(ID, m, params), params) = m$$

for any message m, and for any identity ID.

In your parameter file, you will find p_3, q_3, $\mathsf{comp}(U_3)$, $\mathsf{comp}(V_3)$, $\mathsf{hex}(ct_{3,0})$, $\mathsf{comp}(ct_{3,1})$, $\mathsf{hex}(ct_{3,2})$ and a set of 100 decryption keys SS_3 which contains the master decryption key S. The function comp which returns a hex string, is defined as follows,

$$\mathsf{comp}(P) = \begin{cases} \mathsf{00}, & \text{if } P = \mathcal{O} \\ \mathsf{02}||\mathsf{hex}(P_x), & \text{if } \gamma \equiv 0 \pmod 2 \\ \mathsf{03}||\mathsf{hex}(P_x), & \text{if } \gamma \equiv 1 \pmod 2 \end{cases}$$

where $P = (P_x, P_y) = (\alpha + \beta Z, \gamma + \delta Z)$, and $a||b$ is a concatenation of two hex string a and b, and $\mathsf{hex}(P_x = \alpha + \beta Z) = a||b$ such that a is the hexadecimal representation of α in $\lceil \log_{16}(p) \rceil$ characters and b is the hexadecimal representation of β in $\lceil \log_{16}(p) \rceil$ characters where $\alpha, \beta \in \mathbb{Z}_p$. For example, $\mathsf{hex}(5 + 13Z) = \mathtt{050D}$ when $p = 17$.

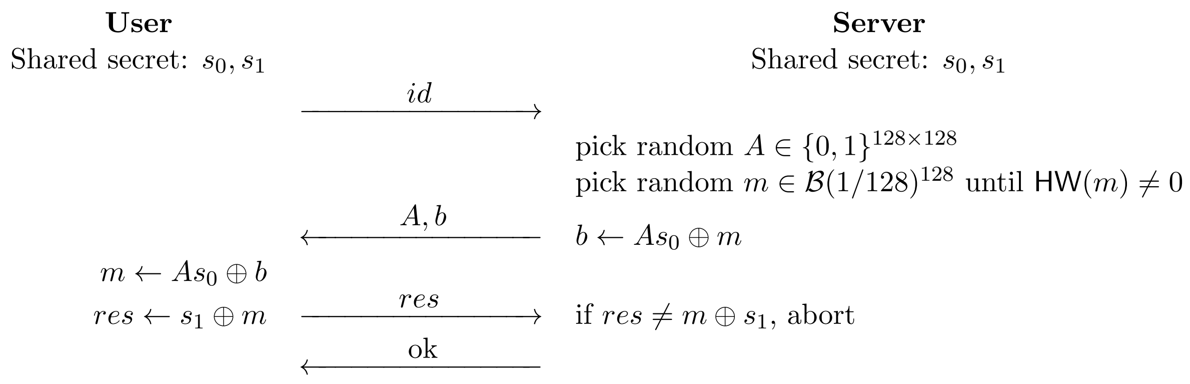
The ciphertext $c_3 = (ct_{3,0}, ct_{3,1}, ct_{3,2})$ is an encryption of a message m_3 with an unknown identity ID_3 and $params_3 = (K_3, E_3, q_3, e, U_3, V_3, W_3)$ where $K_3 = \mathbb{Z}_{p_3}[Z]/(Z^2 + 1)$, $E_3 = \{\mathcal{O}\} \cup \{(x, y) \in K_3^2 \mid y^2 = x^3 + 1\}$ and W_3 is unknown. The Weil pairing can be computed by using embedded function in Sage. For instance, $e(U_3, V_3)$ can be computed by calling

```
U3.weil_pairing(V3, q3)
```

where U_3 and V_3 are points on E_3. In order to define a finite field with the specific modulus, you can use

```
_.<z> = GF(p3)[]
K3 = GF(p3).extension(z^2 + 1, 'x')
```

Compute $Q_3 = \mathsf{hex}(m_3)$, and write it in your answer file.


User
Shared secret: s_0, s_1
Server
Shared secret: s_0, s_1
id
pick random $A \in \{0,1\}^{128\times 128}$
pick random $m \in \mathcal{B}(1/128)^{128}$ until $\mathsf{HW}(m) \neq 0$
A, b
$b \leftarrow As_0 \oplus m$
$m \leftarrow As_0 \oplus b$
$res \leftarrow s_1 \oplus m$
res
if $res \neq m \oplus s_1$, abort
ok

*$\mathcal{B}(p)^n$ returns n values which follows the Bernoulli distribution with p.
*$\mathsf{HW}(v)$ returns the number of non-zero values in the vector v.

Figure 2: The challenge-response protocol for user authentication

Exercise 4 Password-based Challenge-Response Login

A server that provides a secret service (in fact, it is streaming videos with meerkats) is using an interactive challenge-response protocol for user authentication. More precisely, when a new user signs up with a username id, the server and the user generate a shared secret $s_0 = \mathsf{AES}(0^{128}, w\|0^{128-|w|})$, $s_1 = \mathsf{AES}(1^{128}, w\|0^{128-|w|})$ from a password w which is chosen by the user upon enrolment. Here $\mathsf{AES}(K, m)$ is the encryption of a message m by using the AES with a key K, $|w|$ is the length of w, and 0^n is a string of n zero bits.

Once the user is registered, the user can authenticate itself to the server with its password w by following the protocol in Figure 2.

In the protocol, the user first computes $s_0 = \mathsf{AES}(0^{128}, w\|0^{128-|w|})$ and $s_1 = \mathsf{AES}(1^{128}, w\|0^{128-|w|})$. Then, the user sends its username id to the server. The server picks a random 128×128 binary matrix A, and a binary column vector m of degree 128, whose element follows Bernoulli distribution with $p = 1/128$, i.e. the output is 1 with probability p and 0 with probability $1 - p$. Then, the sever consider s_0 as a binary column vector of degree 128, and compute $b = As_0 \oplus m$. Then, the server sends A and b as challenge message. The user needs to recover m from A and b, and send $s_1 \oplus m$ as the response. Then, the server accepts the connection of the user if res is correct. For the storage efficiency, the binary vector of degree n is encoded as byte string of $n/8$. For example, a vector

```
[0, 0, 1, 0, 0, 1, 1, 1]
```

is encoded as

```
0x27
```

and the $n \times n$ binary matrix is encoded as a concatenation of encoded n row vectors. For example, a matrix

```
0, 0, 1, 0, 0, 1, 1, 1
0, 1, 0, 0, 0, 1, 0, 1
1, 1, 1, 1, 1, 1, 1, 1
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 1, 0, 0, 1, 1, 1
0, 1, 0, 0, 0, 1, 0, 1
```

```
1, 1, 1, 1, 1, 1, 1, 1
0, 0, 0, 0, 0, 0, 0, 0
```

is encoded as

```
0x2745FF002745FF00
```

The crypto-apprentice is one of clients of the secret meerkat-streaming service, and you would like to log into his account. You can access to the server lasecpc25.epfl.ch on port 7777 using Sage to initiate the protocol. There are two different type of queries: one to initialize the protocol and one to send a response.

In order to initialize the protocol, you need to send your username. This query should have the following format:

```
username\n
```

Then, you will get a response of the following format (You shouldn't close the connection to respond to the challenge.):

```
base64(A) base64(b)\n
```

where A is the encoded 128×128 binary matrix, b is the encoded vector of degree 128. Then, you can send a response to a server with a query of following format:

```
base64(res)\n
```

Then, you will get a response of the following format:

```
b\n
```

where b is either an abort message or accept message. If $b = 0$, your response was incorrect and the connection is aborted. If $b = 1$, your response was correct and the connection is accepted.

You know that the username of the apprentice is first 6 characters of your sciper. Recover the password of the apprentice w, and write it under Q_4 in your answer file.