Homework 4 - Elliptic Curves and Symmetric Key Cryptography

Cryptography and Security 2017

- 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_1, Q_2, Q_{3a}, Q_{3b}, Q_4 and Q_5. You can download your **personal** files on `http://lasec.epfl.ch/courses/cs17/hw4/index.php`
- The answers should all be ASCII sentences except for Exercise 3 where we expect you to give us a compressed point on an elliptic curve, and the hexadecimal encoding of a finite field-element (see Exercise 3 for details on the compression function and encoding of finite field elements). **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.
- For Exercises 1 and 4, we recommend you to use **the php page we provide (see the respective exercises for the exact address)**. If you want to automatize your queries, the best option is then to use **the Sage code we provide**. To connect directly to the server, **you have to be inside the EPFL network**[1] (for the php page you don't need to be inside the EPFL network). Use VPN if you are connecting from outside EPFL.

  Alternatively, if you want to do it by hand, use *ncat* (you can download a Windows version on `http://nmap.org/download.html` along with nmap). To send a query QUERY to the server lasecpc28.epfl.ch under port PORT write to the command line

[1] If you are using Sage Cloud, connecting directly to the server is impossible, as the cloud itself is never inside of EPFL network.

```
echo QUERY | ncat lasecpc28.epfl.ch PORT
```

under Windows from a cmd in the same directory as the ncat.exe program

```
echo QUERY | ncat.exe lasecpc28.epfl.ch PORT
```

under MAC OS and some linux distributions, ncat doesn't work as it should and you need to replace it by

```
echo QUERY | nc -i 4 lasecpc28.epfl.ch PORT
```

Note that under Windows, you will also have an additional output "close: no error" that you can ignore. The "|" is obtained (for a Swiss keyboard) using altGr + 7.

Under Windows, using Putty is also an option. Select "raw" for "connection type" and select "never" for "close windows on exit".

- 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 30th November at 22:00.

Exercise 1 Vernam Cipher with a Twist

After the lecture about symmetric encryption, the crypto-apprentice found a brilliant idea to redeem his failure with the Vernam cipher. Firstly, he decided to use the same encoding as his previous Vernam cipher, that means only 26 lowercase alphabetic characters and the space symbol can be used for a message, and 'a' is encoded to 0, 'b' to 1, ..., 'z' to 25, and ' ' to 26.

The apprentice disliked the need to exchange a new, very long key for every encrypted message. Instead, he exchanged a single (long) sequence $K = K_0, K_1, \ldots, K_{\mathsf{maxlen}-1}$ of maxlen independent uniformly distributed variables $K_0, \ldots, K_{\mathsf{maxlen}-1} \in \mathbb{Z}_{27}$ with a friend.

His idea was to derive a new Vernam key $k = k_0, k_1, \ldots, k_{\ell-1}$, which is as long as the plaintext $m = m_0, m_1, \ldots, m_{\ell-1}$, by sampling a random $\mathsf{IV} \in \mathbb{Z}_{27}$ and computing k as a function of the IV and (a part of) K for each encryption. So, for each character m_i of the message m, it is encrypted as $(\mathsf{Encode}(m_i) + k_i \bmod 27)$ where k_i is i-th element of the derived key k.

The apprentice liked how the OFB mode makes the i-th block of key stream depend on the previous $(i-1)$ blocks, so he did something like this when computing k, but he added a little twist. He also decided not to reveal how k is computed from K and IV as he realized that the Kerckhoffs principle doesn't mean that the cryptosystem must be public.

You have discovered the web interface `http://lasec.epfl.ch/courses/cs17/hw4/query1.php` that the apprentice was using to test his cryptosystem. By using this web interface, you can ask for encryption of any (reasonably long) message with a random IV and the master key K by querying your SCIPER number and the message m. Alternatively, you can connect to the server lasecpc28.epfl.ch on port 5555 using Sage or ncat to issue queries directly (see instructions). The query should have the following format: SCIPER followed by the encoding of the plaintext you want to encrypt. For instance

```
123456 "red fox"\n
```

will return the encryption of the string "red fox".

In your parameter file, you will find the ciphertext c_1 which is the encryption of a phrase in English Q_1 by using the master key K. Recover Q_1 and write it in your answer file. (This means that you have to provide a **"meaningful" English phrase in ASCII!**)

Hint: The decryption cannot be done without knowing IV, so IV must be somehow included in the ciphertext.

Hint′: Try playing with the provided interface, submitting queries that differ in a single letter etc. The "neutral" character "a" is useful.

Exercise 2 Counter Mode with Updated Key

The apprentice cryptographer turned his attention to symmetric cryptography. He studied the blockcipher modes of operation, and liked CTR mode in particular, because of its simplicity. He decided to implement it straight away, using the AES blockcipher. The implementation of CTR takes a 128 bit secret key, a variable length message, and a 120-bit IV. To encrypt the i^{th} block of plaintext, the 8-bit representation of i is appended to the IV to form a counter, which is processed by AES and the secret key to derive keystream bits. The exact implementation can be found in Figure 1.

After his failures with public key cryptography, the apprentice was hesitant about using a public key exchange protocol. Instead, he decided to exchange a symmetric key with a friend manually, and design a mechanism, that would update the key for each communication session. What could go wrong?

The apprentice's scheme permanently stores three variables:

- a secret key K,
- a 128-bit session counter S,
- a 120-bit message counter N.

The initial values of these variables are the manually exchanged uniformly distributed key for K, $\langle 0 \rangle_{128}$ for S and $\langle 0 \rangle_{120}$ for N.

The scheme allows apprentice to confidentially talk with his friend in communication sessions. To encrypt a message M during a communication session, he uses the function

$$N, C \leftarrow \textsc{Enc}(K, N, M)$$

which calls AES-CTR with the current key K and the current value of the message counter N as IV to get the ciphertext C, and then updates the value of N. At the end of he session, the function

$$K, S, N \leftarrow \textsc{Next-Session}(K, S, N)$$

is called to update the key K using the AES blockcipher and the current value of the session counter S, then increment the session counter S by one, and reset the message counter N to all zeroes. The exact implementation of the functions Enc and next-Session is in Figure 1.

Algorithm AES-CTR(IV, K, M)
  $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
  for $i = 0$ **to** $m - 2$ **do**
    $T_i \leftarrow \mathsf{AES}(K, \mathrm{IV} \| \langle i \rangle_8)$
    $C_i \leftarrow M_i \oplus T_i$
  end for
  $T'_{m-1} \leftarrow \mathsf{AES}(K, \mathrm{IV} \| \langle m-1 \rangle_8)$
  $T_{m-1} \leftarrow \mathrm{trunc}_{|M_{m-1}|}(T'_{m-1})$
  $C_{m-1} \leftarrow M_{m-1} \oplus T_{m-1}$
  return $C_0 \| C_1 \| \ldots \| C_{m-1}$
end Algorithm

Algorithm NEXT-SESSION(K, S, N)
  $K' \leftarrow \mathsf{AES}(K, S)$
  $S' \leftarrow \langle \mathsf{int}(S) + 1 \rangle_{128}$
  $N' \leftarrow \langle 0 \rangle_{120}$
  return K', S', N'
end Algorithm

Algorithm ENC(K, N, M)
  $C \leftarrow$ AES-CTR(N, K, M)
  $N' \leftarrow \langle \mathsf{int}(N) + 1 \rangle_{120}$
  return N', C
end Algorithm

Figure 1: AES-CTR (left), NEXT-SESSION and ENC (right). The function $\mathrm{trunc}_\ell(X)$ truncates the binary string X to ℓ leftmost bits (e.g. $\mathrm{trunc}_3(\mathtt{101111}) = \mathtt{101}$), $\langle i \rangle_\ell$ denotes the ℓ-bit binary representation of the integer i (e.g. $\langle 5 \rangle_4 = \mathtt{0101}$), and $\mathsf{int}(x)$ denotes the representation of a binary string as an integer (e.g. $\mathsf{int}(\mathtt{0101}) = \mathsf{int}(\mathtt{0000101}) = 5$). Note that for AES-CTR, we have $|\mathrm{IV}| = 120$.

In the CTR mode, we denote by $M_0, M_1, \ldots, M_{m-1} \xleftarrow{n} M$ partitioning a binary string M into blocks of n bits, with the last block being possibly shorter. I.e. $M = M_0 \| M_1 \| \ldots \| M_{m-1}$ ($\|$ denotes simple concatenation of strings) with $|M_i| = n$ for $0 \leq i < m-1$ and $1 \leq |M_{m-1}| \leq n$ and $m = \lceil M/n \rceil$.

To be able to process ASCII strings with AES (which works with binary strings), the crypto-apprentice encodes the ASCII value of every character as an 8-bit binary string, and then concatenates these 8-bit strings. For example, "Red Fox!" would be encoded as `01010010 01100101 01100100 00100000 01000110 01101111 01111000 00100001` (we included spaces just for clarity!). As this representation is not very efficient, the crypto apprentice uses the base64 encoding when storing and sending binary strings that may contain unprintable characters, such as ciphertexts. For example, the binary string `01010010 01100101 01100100 00100000 01000110 01101111 01111000 00100001` would be encoded as `UmVkIEZveCE=`.

When he finished the implementation, the apprentice did the manual exchange of the secret key K with a friend and set $S = \langle 0 \rangle_{128}$ and $N = \langle 0 \rangle_{120}$. He then encrypted a few messages and sent them to the friend. You managed to intercept both the very first plaintext M_{21} of the first session and the corresponding ciphertext C_{21}. The apprentice then called NEXT-SESSION once to start the second session, and is has just finished encrypting Q_2, the i_2-th message in this session. You have intercepted C_{22}, the ciphertext of Q_2 (which is an english ASCII-encoded phrase).

In your parameter file, you will find the ASCII string M_{21} and two base64-encoded strings C_{21} and C_{22}, and the integer i_2. Recover Q_2, the plaintext of C_{22} and write it in your answer file as an ASCII string. (This means that you have to provide a **"meaningful" English phrase in ASCII!**)

Exercise 3 3-party Diffie-Hellman key agreement

Let $K = \mathbb{F}_{p^2} = \mathbb{Z}_p[Z]/(P(Z))$ be a finite field where p is an odd prime such that $p \bmod 3 = 2$ and $P(Z)$ is an irreducible polynomial in $\mathbb{Z}_p$. Then, the elliptic curve $E(K) = \{\mathcal{O}\} \cup \{(x, y) \in K^2 \mid y^2 = x^3 + 1\}$ is a supersingular elliptic curve and there exists a function $e : \mathsf{G} \times \mathsf{G} \to \mathbb{F}_{p^2}$

called *pairing*, which is *bilinear* and *non-degenerate*, where G is a subgroup of points of the elliptic curve $E(K)$.

Alice wants to establish a secure communication channel with you and Bob by using 3-party Diffie-Hellman key exchange (See slide 385 of the lecture). Alice generated an elliptic curve $E_3(K_3) = \{\mathcal{O}\} \cup \{(x, y) \in K_3^2 \mid y^2 = x^3 + 1\}$ over $K_3 = \mathbb{F}_{p_3^2} = \mathbb{Z}_{p_3}[Z]/(Z^2 + 1)$, a point $G_3 \in \mathrm{E}_3(K)$ that generates a subgroup $\langle G_3 \rangle \subset E_3(K)$ of order n_3, and the pairing e_3 which is given in the tutorial.

You did the 3-party DH key exchange with Alice and Bob, and you received the public messages A_3 and B_3 from them. For transmission and storage, the points on the elliptic curve E_3, namely G_3, A_3 and B_3, are compressed by using the function comp which returns a hex string, and is defined as follows,

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

where $P = (P_x, P_y) = (\alpha Z + \beta, \gamma Z + \delta)$, and $a||b$ is a concatenation of two hex string a and b, and $\mathsf{hex}(P_x = \alpha Z + \beta) = a \| b$ 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, $\mathsf{hex}(5Z + 13) = \mathtt{050D}$ when $p = 17$.

In your parameter file, you will find the prime number p_3, the generator G_3 of a subgroup in the curve $E_3(\mathbb{F}_{p_3^2})$ defined as above and its order n_3, the public messages A_3 and B_3, and your own secret parameter c_3. Your task is to compute the point C_3 that you should send to Alice and Bob, and the shared key K which is the output of the three party DH key exchange between Alice, Bob, and you. Then, compute $Q3a = \mathsf{comp}(C_3)$ and $Q3b = \mathsf{hex}(K)$, and write them in your answer file.

Hint: Be careful to implement the hexadecimal encoding for field elements, and the point compression function *exactly* as we describe them *in this document*. In particular, make sure that you concatenate values in the correct order.

Exercise 4 A new Feistel Scheme

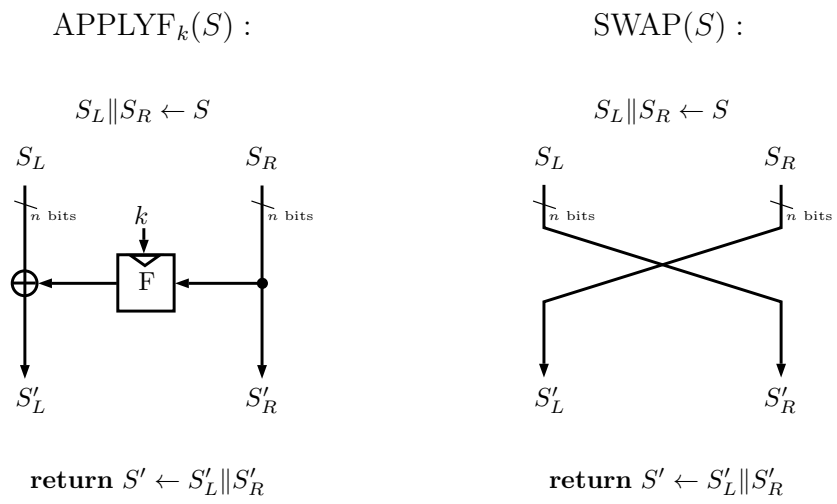

Figure 2: The APPLYF and SWAP operations used in a round of a Feistel scheme.

The apprentice spent an hour studying the Feistel schemes. He appreciated how they can double the effective blocklength of a keyed cryptographic function and thought that a Feistel scheme is a perfect way of extending the blocksize of AES to 256 bits. A Feistel round consists of two steps: one is to apply a round function F with the round key k to a state S ($\text{APPLYF}_k(S)$) and the second step is to swap the two halves of the state ($\text{SWAP}(S)$). Thus, a normal Feistel round is $\text{SWAP}(\text{APPLYF}_k(m))$ (see Figure 2). Aware of the need of making cryptography fast, the apprentice used the smallest number of rounds that felt secure to him: two.

Determined to test his idea, the apprentice implemented the following blockcipher, that uses two keys k_{41} and k_{42} of 128 bits each to encrypt a block M of 256 bits, outputs a 256-bit ciphertext C, and uses AES in place of the round function F:

1: **Algorithm** FEISTEL 2(k_{41}, k_{42}, M)
2: R1: $S = \text{SWAP}(\text{APPLYF}_{k_{41}}(M))$
3: R2: $C = \text{APPLYF}_{k_{42}}(S)$
4: **return** C
5: **end Algorithm**

The apprentice decided to challenge you, his personal cryptanalytic nemesis, and encrypted a 32-character long (English) sequence of ASCII characters Q_4 using FEISTEL 2 with two independent uniformly distributed keys k_{41} and k_{42}, and gave you the corresponding ciphertext C_4. To tease you even more, the apprentice implemented a network interface that will encrypt any plaintext with FEISTEL 2 using k_{41}, k_{42} for you.

In your parameter file, you will find the ciphertext C_4 encoded in base64. On the address `http://lasec.epfl.ch/courses/cs17/hw4/query2.php`, you will find a web interface that will take your SCIPER and a plaintext encoded in base64, and will return the encryption of this plaintext under FEISTEL 2 with k_{41}, k_{42}, also encoded in base64. Alternatively, you can connect to the server lasecpc28.epfl.ch on port 6666 using Sage or ncat to issue queries directly (see instructions). The query should have the following format: SCIPER followed by the base64 encoding of the plaintext you want to encrypt. For instance

```
123456 UmVkIEZveCEAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=\n
```

will return the FEISTEL 2 encryption of the ASCII string "Red Fox" padded with 24 zero bytes.

Recover Q_4 and write it in your answer file as an ASCII string. (This means that you have to provide a **"meaningful" English phrase in ASCII!**). **Explain in your source code which queries did you use, and why.**

Remark: Note that the implementation of AES in Sage works directly with ASCII strings, so you do not need to do the conversion to binary.

Exercise 5 A Bad Streamcipher

After his failures with blockciphers and modes of operation, the apprentice turned his attention to stream ciphers. He liked A5/1, which combines several linear feedback shift registers into a (more or less) secure streamcipher. Feeling inspired, he designed his own streamcipher that combines two linear elements into what he hoped would be a secure design.

The apprentice called his new design NONSENSE (as in "New Optimally Nimble Streamcipher for ENcrypting Securely and Efficiently "). It works as follows. The secret key of NONSENSE consists of two invertible matrices $K_{51}, K_{52} \in \mathbb{Z}_2^{64\times 64}$. To encrypt a plaintext M

of ℓ bits, NONSENSE takes a 64-bit IV, generates an ℓ-bit key stream k and computes the ciphertext $C = M \oplus k$. The keystream is generated in 64-bit blocks as follows:

Algorithm NONSENSE($K_{51}, K_{52}, \mathrm{IV}, \ell$)
  $k \leftarrow$ empty string
  $i \leftarrow 0$
  $S_{-1} \leftarrow \mathrm{IV}$
  while $i \cdot 64 < \ell$ **do**
    if $i \equiv 0 \pmod 2$ **then**
      $S_i \leftarrow K_{51} \cdot S_{i-1} \bmod 2$
    else
      $S_i \leftarrow K_{52} \cdot S_{i-1} \bmod 2$
    end if
    $k \leftarrow k \| S_i$
    $i \leftarrow i + 1$
  end while
  return $\mathsf{trunc}_\ell(k)$
end Algorithm

where $K_{5b} \cdot S_{i-1} \bmod 2$ ($b \in \{1, 2\}$) means interpreting S_{i-1} as a column vector of 64 bits and performing the matrix multiplication with K, with the arithmetic done modulo 2.

The apprentice wanted to be sure that this time, you won't be able to disrupt his secret communication, so he decided to include IV into the secret key as well. He exchanged the secret key consisting of $K_{51}, K_{52}, \mathrm{IV}$ with a friend, and increments the IV after every encryption query by 1, i.e. $\mathrm{IV} \leftarrow \langle \mathsf{int}(\mathrm{IV}) + 1 \bmod 2^{64} \rangle_{64}$.

As usual, you managed to get your hands on a plaintext M_{51} and its encryption C_{51} under NONSENSE with $K_{51}, K_{52}, \mathrm{IV}$. You have also intercepted the ciphertext C_{52} that is an encryption of the message Q_5, which the apprentice encrypted right after M_{51}. You are pretty sure that you can recover Q_5.

In your parameter file, you will find the (English) ASCII sequence M_{51} and the two ciphertexts C_{51} and C_{52}, both encoded in base64. Recover the secret message Q_5 and write it in your answer file as an ASCII string. (This means that you have to provide a **"meaningful" English phrase in ASCII!**).