Homework 4 - Elliptic Curves and Symmetric Key Cryptography

Cryptography and Security 2016

- 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_3, Q_4 and Q_5. You can download your **personal** files on `http://lasec.epfl.ch/courses/cs16/hw4/index.php`
- The answers should all be ASCII sentences except for Exercise 4 where we expect you to give us a point on an elliptic curve, i.e. a pair of integers. **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 3, we recommend you to use **the php page we provide you (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 you**. To connect to the servers, **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.

  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

  ```
  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 the 24th of November** at 22h00.

Exercise 1 Modes of Operation with Bad Key Management

The apprentice cryptographer felt new inspiration after he attended the lectures about symmetric key cryptography. He liked AES, and he liked the modes of operation, so he decided to implement not one, but two of them. The two he liked the most were the Cipher Feedback mode (CFB) and the Counter Mode (CTR).

The crypto apprentice's implementation of CFB mode takes a 128 bit secret key, and a variable length message as inputs, and internally samples a random IV which is used for the CFB encryption. The CFB encryption then proceeds as described in the course, with AES in place of the 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 of both modes can be found in Figure 1.

```
1: Algorithm AES-CFB($K, M$)
2:     $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
3:     Sample random IV $\in \{0,1\}^{128}$
4:     $C_{-1} \leftarrow \mathrm{IV}$
5:     for $i = 0$ to $m-2$ do
6:         $T_i \leftarrow \mathsf{AES}(K, C_{i-1})$
7:         $C_i \leftarrow M_i \oplus T_i$
8:     end for
9:     $T_{m-1} \leftarrow \mathrm{trunc}_{|M_{m-1}|}(\mathsf{AES}(K, C_{m-2}))$
10:    $C_{m-1} \leftarrow M_{m-1} \oplus T_{m-1}$
11:    return $\mathrm{IV} \| C_0 \| C_1 \| \ldots \| C_{m-1}$
12: end Algorithm
```

```
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
```

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

In both modes, we denote by $M_0, M_1, \ldots, M_{m-1} \overset{n}{\leftarrow} M$ 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 hexadecimal encoding when storing and sending binary strings, such as ciphertexts. For example, the binary string `01010010 01100101 01100100 00100000 01000110 01101111 01111000 00100001` would be encoded as `52656420466F7821`.

The crypto-apprentice was quite pleased by his implementations. He picked a secret key K_1 and used it with the AES-CFB mode to exchange messages with a friend. He was then too lazy to keep track of two independent keys, and therefore he decided to use K_1 with the AES-CTR mode as well (he thought that having two modes which are completely different will serve as sufficient separation). When testing his implementation, he wrote a piece of code that listened for encryption queries (consisting of an IV and a plaintext) and returned the corresponding ciphertext, and he was then careless enough to let it run.

In your parameter file, you will find an intercepted ciphertext C_1 that is an encryption of a secret message Q_1 under the AES-CFB mode with key K_1. On the address `http://lasec.epfl.ch/courses/cs16/hw4/query1.php`, you will find a web interface that will take your SCIPER, an IV, and a plaintext and will return the encryption of this plaintext under AES-CTR, with K_1 and the provided IV. 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 15-byte IV, and the plaintext you want to encrypt (both IV and plaintext have to in hexadecimal). For instance

```
123456 00112233445566778899aabbccddee 0102030405
```

will return the CTR encryption of `0102030405` with IV `00112233445566778899aabbccddee`.

Recover Q_1 and write it in your answer file. Note that we expect an ASCII string that contains an English phrase and nothing else. **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 2 Bad Streamcipher

Slightly discouraged by his failure with blockcipher modes of operation, our crypto-apprentice turned his attention to streamciphers. He liked the simplicity and speed of linear feedback shift registers (LFSR), so he decided to use a 256-bit LFSR to hold and update the state of his streamcipher (the bigger the state, the more difficult it is to recover the secret key!). He also observed that all streamciphers based on LFSR(s) used some non-linear function to derive the keystream bits from the state (e.g. the conditional XOR with majority function in A5/1).

The apprentice had a brilliant idea - why not applying a much stronger non-linear function, but only once, to the initial state of the LFSR and then simply use the output of the LFSR as the keystream? This should make all the keystream bits non-trivially dependent on the

```
1: Algorithm UNSAFE(IV, $K$, $M$)
2:     $K' \leftarrow \mathsf{AES}(\mathrm{IV}, K)$
3:     $R \leftarrow \mathrm{IV} \| K'$
4:     for $i = 0$ to $|M| - 1$ do
5:         $T \leftarrow R_0$
6:         $C_i \leftarrow M_i \oplus T$
7:         $F \leftarrow R_{10} \oplus R_5 \oplus R_2 \oplus R_0$
8:         $R \leftarrow R_1 \| R_2 \| \dots \| R_{255} \| F$
9:     end for
10:    return
       $\mathrm{IV} \| C_0 \| C_1 \| \dots \| C_{|M|-1}$
11: end Algorithm
```

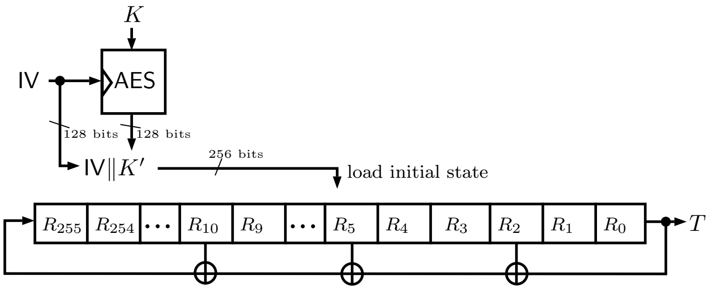

Figure 2: The streamcipher UNSAFE. Here $|X|$ denotes the length of a binary string X, X_i denotes i^{th} bit of X (indexing starts from 0) and $X \| Y$ denotes concatenation of two binary strings X and Y.

secret key, and the actual keystream computation extremely fast. He picked AES as a strong non-linear function. For the LFSR itself, the apprentice picked a connection polynomial of degree 256 that is irreducible in $\mathbb{Z}_2$, namely $P_2(x) = x^{256} + x^{10} + x^5 + x^2 + 1$.

To encrypt a plaintext M with a key K, and an initialization vector IV, the apprentice first applies AES to K, using the IV as the AES key to compute a derived key K', and appends the IV to K' to obtain the initial state. This is then iteratively updated with the LFSR, and the least significant bit is always used to encrypt one bit of the message. His design, called “Unbelievable New Streamcipher with AES-then-Feedback Encryption” (UNSAFE) is described in Figure 2.

The crypto-apprentice used UNSAFE to exchange messages with a friend. Again, he wanted to apply the bit-oriented streamcipher to ASCII strings, so he used the same encoding as in Exercise 1, and represented binary strings as hexadecimal strings for storage and sending.

You have intercepted C_{21}, a 256-bit fragment of a longer ciphertext, for which you also know the corresponding fragment of plaintext P_{21} and IV_{21} that was used for encryption. However, you do not know how long was the original message, and what was the position of the fragment in it. You have also intercepted a complete ciphertext C_{22}, which encrypts an (encoded) ASCII string that contains an English phrase Q_2, and the IV IV_{22} that was used to encrypt it. Recover Q_2 and write it in your answer file. Note that we expect a decoded English phrase, and nothing else.

Exercise 3 Randomized Vernam

Feeling nostalgic, the apprentice cryptographer felt like trying to improve the Vernam cipher once again. This time he knew that the secret key should be as long as the plaintext, but exchanging a secret key for every query still felt like too much hassle. He thus decided to get inspired by modes of operation and selecting a random IV for every query and deriving a “new key” from a master-key should do the trick. Rebelling against the Kerkhoff principles, he did not share his design with you.

You have discovered that again, the apprentice has left running a web interface that allows you to submit plaintexts and obtain the corresponding ciphertexts. You can have a blackbox access to the encryption mechanism on the php page `http://lasec.epfl.ch/courses/cs16/hw4/query.php` where you have to provide the SCIPER and the plaintext you want to encrypt. We recommend you to use this page for your queries.

Another alternative is to connect to the server lasecpc28.epfl.ch on port 5555 using Sage or ncat (see instructions). The query should have the following format: SCIPER followed by the plaintext you want to encrypt. For instance

```
123456 my message
```

Our apprentice used the following encoding scheme for the standard 26 lowercase letter alphabet plus the space symbol, the . symbol and the , symbol. The letter 'a' is mapped to 0, 'b' to 1, ..., 'z' to 25 and ' ' to 26 and '.' to 27 and ',' to 28.

You have intercepted a ciphertext C_3 that encrypts a secret encoded phrase Q_3. Recover Q_3 and write it in your answer file. **Explain in your source code which queries did you use, and why.**

Hint: Remark that we work in a field.

Exercise 4 ElGamal in Elliptic Curve

This time, we consider ElGamal cryptosystem over an elliptic curve.

- *Key Generation:* First, we pick a prime p_4. We work over a field $\mathbf{Z}_{p_4}$ and use parameters a_4 and b_4 to define the curve $y^2 = x^3 + a_4x + b_4$. We pick a generator P_4 from the curve which has prime order n_4. In the end, we pick the secret key $d_4 \in \mathbf{Z}_{n_4}$ and compute the public key $Y_4 = d_4P_4$.
- *Encryption:* Given a point Q_4 from the curve, we pick a random $r_4 \in \mathbf{Z}_{n_4}$ and compute the ciphertext (U_4, V_4) where $U_4 = r_4P_4$ and $V_4 = Q_4 + r_4Y_4$.

In this exercise, we encrypted a secret message Q_4 which is a point on the curve.

In your paramater file, you have an ElGamal ciphertext (U_4, V_4) encrypted by the public-key Y_4, the secret key d_4 and public parameters a_4, b_4, p_4, n_4 and P. Decrypt the ciphertext (U_4, V_4) to obtain Q_4. Note that in this case we don't have an English phrase, but a point on the curve. Please provide the coordinates of the point without using any point compression.

Exercise 5 A new Feistel Scheme

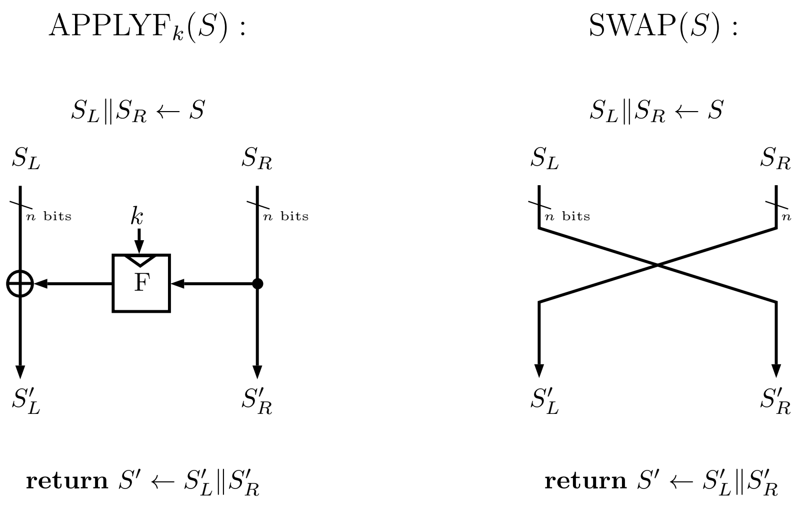
APPLYF$_k(S)$:
$S_L \| S_R \leftarrow S$
S_L
S_R
n bits
k
F
n bits
S'_L
S'_R
return $S' \leftarrow S'_L \| S'_R$
SWAP(S) :
$S_L \| S_R \leftarrow S$
S_L
S_R
n bits
n bits
S'_L
S'_R
return $S' \leftarrow S'_L \| S'_R$

Figure 3: The APPLYF and SWAP operations used in a round of a Feistel scheme.

After he gave it a try with stream ciphers and modes of operation, our crypto apprentice decided to design a block cipher as well. Analysing the slides of the crypto course, he saw that DES is based on Feistel scheme of 16 rounds. He saw DES is broken in the case when one is able to access many messages encrypted with the same key. He decided he can do a simpler but still secure cipher based on Feistel where he is encrypting few messages, each with a new key.

When looking on how to simplify the block cipher, he decided to cut down the number of rounds from 16 to 6 (which still seems like a secure choice). While analysing one round of Feistel he saw you do 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 3). Not understanding what is the role of each step he decided to cut down the number of swaps (swaps do not depend on the secret key, can they possibly have any role in the design's security?) and designed a variant of Feistel with 6 rounds that are:

```
1: Algorithm FEISTEL 2($k_1, k_2, k_3, M$)
2:     R1: $S_1 = \text{APPLYF}_{k_1}(M)$
3:     R2: $S_2 = \text{SWAP}(\text{APPLYF}_{k_1}(S_1))$
4:     R3: $S_3 = \text{APPLYF}_{k_2}(S_2)$
5:     R4: $S_4 = \text{APPLYF}_{k_2}(S_3)$
6:     R5: $S_5 = \text{SWAP}(\text{APPLYF}_{k_3}(S_4))$
7:     R6: $C = \text{APPLYF}_{k_3}(S_5)$
8:     return $C$
9: end Algorithm
```

The function F is an AES encryption and the three keys are of 128 bits each. As the first two keys have in total 256 bits and they are chosen at random, this seems sufficient to our apprentice. Thus, two secure keys provide a strong security. Working on a 8-bit microprocessor, he decides to choose a random number between 0 and 255, represent it in binary on 8 bits and concatenate it 16 times in order to construct the third key of 128 bits.

He decided to challenge you and encrypted an English message (using the encoding from Exercise 1). In your parameter file you will find the ciphertext C_5. Decrypt the ciphertext and write the answer Q_5. Note that we expect to receive an ASCII string (and not a hexa or binary string).