Homework 4 - Elliptic Curves and Symmetric Key Cryptography

Cryptography and Security 2019

- 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 all `Q` values specified in the questions below. You can download your **personal** files from the following link: `https://lasec.epfl.ch/courses/cs19/hw4/index.php`
- You will find an example parameter and answer file on the moodle. You can use this parameters' file to test your code and also ensure that the types of `Q` values you provided match what is expected. **Please do not put any comment or strange character or any new line** in the .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. Notebook files are allowed, but we prefer if you export your code as a text file with a sage/python script.
- 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/corrections 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.
- The homework is due on Moodle on **Saturday the 30th of November** at 22h00.

Exercise 1 DDH for Elliptic Curves

In this question you are asked to solve the decisional Diffie-Hellman on an elliptic curve.

In your parameters file you can find `Q1_q` the size of the finite field your elliptic curve is defined over, `Q1_f` the polynomial used to define the field, `Q1_E` $= [a, b]$, the coefficients of the curve $y^2 = x^3 + ax + b$, followed by a list `Q1_challenge` of tuples $(g, A = g^a, B = g^b, C)$.

You are asked to compute the list `Q1_response` $= [a_1, a_2, \ldots, a_k]$ such that a_i is 1 if the i^{th} challenge tuple is a correct Diffie-Hellman tuple, and 0 otherwise.
hint: The given curve is supersingular.

Exercise 2 Bat-Stream

Our favorite superhero, Batman, is back; this time with a stream cipher! After reading many papers about stream ciphers with short internal states, Batman decides to create his own cipher; The Bat-Stream!

He decides to use the idea of using linear registers to create his bit-stream. The cipher is explained in Figure 1. The end of a message is determined with a "#" symbol, and the last block is padded by trailing zeros, so that the bit-length of the whole message would be a multiple of N.

Algorithm ENCRYPT($Stream, K, A, N$)
  $B \leftarrow \perp$
  $counter \leftarrow 0$
  $S \leftarrow K$
  $C \leftarrow \perp$
  while $True$ **do**
    if $counter \mod N/8 = 0$ AND $|B| \neq 0$ **then**
      $C \leftarrow C||\text{ENC\_BLOCK}(B, S)$
      $S \leftarrow S \times A^T$
      $B \leftarrow \perp$
    end if
    $m \leftarrow Stream.get\_byte()$ ▷ Get one byte from the stream
    $B \leftarrow B||m$
    $counter \leftarrow counter + 1$
    if $m =$ "#" **then**
      $B \leftarrow B||0 \ldots 0$ ▷ Make the block of size $\lfloor N/8 \rfloor$ bytes by adding zeros
      $C \leftarrow C||\text{ENC\_BLOCK}(B, S)$
      return C
    end if
  end while
end Algorithm

Algorithm ENC_BLOCK(B, S)
  $b_0 b_1 \ldots b_{N-1} \leftarrow B$ ▷ b_0 is MSB and b_{N-1} is LSB of B
  $s_0 s_1 \ldots s_{N-1} \leftarrow S$ ▷ s_0 is MSB and s_{N-1} is LSB of S
  return $(b_i \oplus s_i)_{i \in \{0, \ldots N-1\}}$
end Algorithm

Figure 1: The encryption algorithm

In your parameters file, you are given the block size `Q2_N` two ciphertexts `Q2_c1` and `Q2_c2`, and also the invertible matrix `Q2_A`. These ciphertexts belong to two plaintexts which were encrypted right after each other, meaning that the key-stream used to encrypt the second one, is the continuation of the bit-stream used to encrypt the first one.

You are asked to find the plaintexts corresponding to these ciphertexts and write it down in your solutions file. The plaintexts only consist of printable characters.

hint: You know what every message that batman sends ends with? You know! Because "I am batman!".

Exercise 3 Modes of operation with counter

After the lecture about the mode of operation, the crypto-apprentice realized that any mode of operation might be secure with a counter-based initial vector. So, he decided to implement a system which keeps changing the mode of operations for each message. Since CBC mode cannot take incomplete block as input, the crypto-apprentice decided to accept only complete blocks for any mode. That means that the message must be a multiple of the block size. The exact implementation can be found in Figure 2.

Algorithm AES-CTR$^*(N, K, M)$
  $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
  for $i = 0$ **to** $m - 1$ **do**
    $V \leftarrow \mathsf{LitEnd}((N - i) \bmod 2^{64}) \| \mathsf{BigEnd}((N + i) \bmod 2^{64})$
    $T_i \leftarrow \mathsf{AES}(K, V)$
    $C_i \leftarrow M_i \oplus T_i$
  end for
  return $C_0 \| C_1 \| \ldots \| C_{m-1}$
end Algorithm

Algorithm AES-CBC$^*(N, K, M)$
  $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
  $C_{-1} \leftarrow \mathsf{LitEnd}(N) \| \mathsf{BigEnd}(N)$
  for $i = 0$ **to** $m - 1$ **do**
    $C_i \leftarrow \mathsf{AES}(K, C_{i-1} \oplus M_i)$
  end for
  return $C_0 \| C_1 \| \ldots \| C_{m-1}$
end Algorithm

Algorithm AES-CFB$^*(N, K, M)$
  $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
  $C_{-1} \leftarrow \mathsf{LitEnd}(N) \| \mathsf{BigEnd}(N)$
  for $i = 0$ **to** $m - 1$ **do**
    $C_i \leftarrow \mathsf{AES}(K, C_{i-1}) \oplus M_i$
  end for
  return $C_0 \| C_1 \| \ldots \| C_{m-1}$
end Algorithm

Algorithm ENC(N, K, M)
  if $N \equiv 0 \pmod 4$ **then**
    $C \leftarrow$ AES-CTR$^*(N, K, M)$
  else if $N \equiv 1 \pmod 4$ **then**
    $C \leftarrow$ AES-OFB$^*(N, K, M)$
  else if $N \equiv 2 \pmod 4$ **then**
    $C \leftarrow$ AES-CFB$^*(N, K, M)$
  else if $N \equiv 3 \pmod 4$ **then**
    $C \leftarrow$ AES-CBC$^*(N, K, M)$
  end if
  $N' \leftarrow (N + 1) \bmod 2^{64}$
  return N', C
end Algorithm

Algorithm AES-OFB$^*(N, K, M)$
  $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$
  $T \leftarrow \mathsf{LitEnd}(N) \| \mathsf{BigEnd}(N)$
  for $i = 0$ **to** $m - 1$ **do**
    $T \leftarrow \mathsf{AES}(K, T)$
    $C_i \leftarrow M_i \oplus T$
  end for
  return $C_0 \| C_1 \| \ldots \| C_{m-1}$
end Algorithm

Figure 2: AES-CTR*, AES-CBC*, AES-CFB* (left), ENC and AES-OFB* (right). $|M|$ denotes the bit length of the message M, $M_0, M_1, \ldots, M_{m-1} \xleftarrow{128} M$ denotes the split of a message M into m blocks of 128 bits. The function $\mathsf{BigEnd}(n)$ denotes the 8 byte representation of the 64 bit integer n in big-endian (e.g. $\mathsf{BigEnd}(81985529216486895) =$ `0x0123456789ABCDEF`), and the function $\mathsf{LitEnd}(n)$ denotes the 8 byte representation of the 64 bit integer n in little-endian (e.g. $\mathsf{LitEnd}(81985529216486895) =$ `0xEFCDAB8967452301`). Note that N is a 64 bit integer and $\mathsf{AES}(K, X)$ means an encryption of a message X by using AES and the key K.

In order to not manually store the secret key K and the counter N, the apprentice tried to implement it as an encryption server. It permanently stores the secret key K and the initial counter N, and when one sends a message M to the encryption server, the encryption server takes K and N from its storage and calls ENC(N, K, M). Then, the server returns the ciphertext C and store the new counter N' in its storage.

When he finished the implementation, the apprentice manually exchanged the secret key K and the initial value of the counter N with a friend. After exchanging some messages with his friend, the apprentice was wondering if it is really secure. Since you found problems from his previous encryptions, he decided to ask you.

The apprentice first picked a random byte string M and a random counter N', and computed C which is the encryption of M with the above encryption by using the same key K and the counter N', i.e. $\_, C = \text{ENC}(N', K, M)$. He then gave you C, N' and the access to

the encryption server while asking if you can decrypt it. Now, it's time to decrypt it.

By querying your SCIPER and a plaintext encoded in base64 with the following \n to the encryption server `lasecpc25.epfl.ch` on port 6666, you can get the encryption of this plaintext, also encoded in base64. 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 encryption of the ASCII string "Red Fox!" with following 24 zero bytes.

In your parameter file, you can find the counter N' in `Q3_Np`, and the ciphertext C encoded in base64 in `Q3_C`. Recover M and write $\mathsf{base64}(M)$ under `Q3_M` in your answer file as a ASCII string.

Exercise 4 Nibblet Blockcipher

Our apprentice was responsible with designing a symmetric cipher for memory and energy constrained devices. For the particular application he had in mind, only small set of synchronized devices needed to share a key and this key would be generated and hard coded into the device during manufacturing time. To thwart possible key leakage problems, he decided to revisit the design of DES and invent a cipher that does not use an explicit key. As the block size of the cipher, he decided that 64 bits would be adequate.

His design is actually 9-round Feistel network complemented with initial and final bit permutation layers $\mathsf{P_{in}}, \mathsf{P_{out}}$ as seen in Figure 3. Layers $\mathsf{P_{in}}$ and $\mathsf{P_{out}}$ move each bit to a new position, i.e. application of $\mathsf{P_{in}}$ moves the bit at position i to position $\mathsf{P_{in}}(i)$ at the output. Meanwhile, each round function RF_i for $i = 0, \ldots, 8$ is chosen differently to increase the effective key-length, through sampling S-boxes randomly for each set of synchronized devices at manufacturing time.

The internal structure of the round function RF_i is further illustrated in Figure 3. 32-bit input L is first split into a sequence of nibbles (i.e. 4-bit strings) $B_0||B_1||\ldots||B_7$. Each nibble B_j is then passed through 4-bit S-box $S_{i,j}$, i.e. both input and output are 4-bit. Finally, the nibble permutation layer P_i is applied, where each nibble B_j is moved to its new nibble position $\mathsf{P}_i(j)$, i.e. combined with S-box acting as $C_{\mathsf{P}_i(j)} = \mathsf{S}_{i,j}(B_j)$. The outputs is the sequence $C_0||C_1||\ldots||C_7$. The permutations P_i for each round is chosen rather simply, by tossing a coin and picking either of the following fixed permutations $\mathcal{P}_0$ or $\mathcal{P}_1$:

$$\mathcal{P}_0(i) = i + 2 \bmod 8, \qquad \mathcal{P}_1(i) = i - 2 \bmod 8$$

The generation Gen, round function RF_i, and encryption Enc are given as algorithms in Figure 3.

In this question, your goal is to implement this blockcipher with respect to S-box configuration (`Q4_s` as list), initial and final permutation layers (`Q4_pin`, `Q4_pout`), and nibble permutations (`Q4_p` as double list) given in your parameter file. Each permutation p is encoded as a list of elements in sage as $[p(0), p(1), \ldots, p(n-1)]$.

Your task is to encrypt the given message (`Q4_m` encoded with ASCII) using the CBC mode using the 64-bit IV is given in your parameter file as `Q4_IV`.

$\mathsf{Gen}()$:

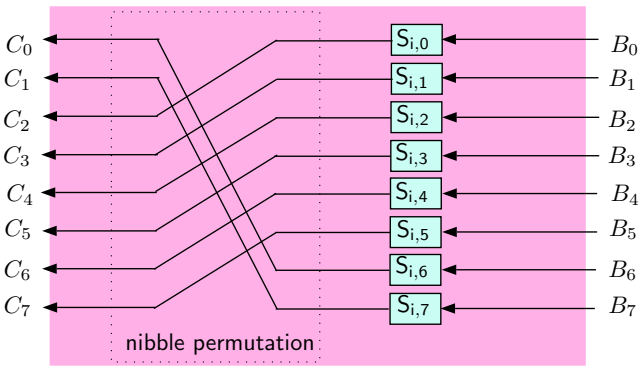
1: $\mathsf{P_{in}}, \mathsf{P_{out}} \leftarrow$ random 64-bit shuffling
2: **for** $i = 0$ to 8 **do**
3: **for** $j = 0$ to 7 **do**
4: $\mathsf{S}_{i,j} \leftarrow$ random 4-bit S-box
5: **end for**
6: $\mathsf{P}_i \leftarrow$ sample either $\mathcal{P}_0$ or $\mathcal{P}_1$
7: encode $(\mathsf{S}_{i,j}, \mathsf{P}_i, \mathsf{P_{in}}, \mathsf{P_{out}})$ as the key
8: **end for**

$\mathsf{RF}_i(R)$:

1: $b_0||b_1||\ldots||b_{31} \leftarrow R$
2: **for** $j = 0$ to 7 **do**
3: $B_j \leftarrow b_{4j}||b_{4j+1}||b_{4j+2}||b_{4j+3}$
4: $B_j \leftarrow \mathsf{S}_{i,j}(B_j)$
5: $C_{\mathsf{P}_i(j)} \leftarrow B_j$
6: $c_{4j}||c_{4j+1}||c_{4j+2}||c_{4j+3} \leftarrow C_j$
7: **end for**
8: $Z \leftarrow c_0||c_1||\ldots||c_{31}$
9: **return** Z

$\mathsf{Enc}(X)$:

1: $x_0||x_1||\ldots||x_{63} \leftarrow X$
2: **for** $u = 0$ to 63 **do**
3: $y_{\mathsf{P_{in}}(u)} \leftarrow x_u$
4: **end for**
5: $L_0 \leftarrow y_0||y_1||\ldots||y_{31}$
6: $R_0 \leftarrow y_{32}||\ldots||y_{63}$
7: **for** $i = 0$ to 8 **do**
8: $L_{i+1}||R_{i+1} \leftarrow R_i||(L_i \oplus \mathsf{RF}_i(R_i))$
9: **end for**
10: $y_0||y_1||\ldots||y_{63} \leftarrow R_9||L_9$
11: **for** $u = 0$ to 63 **do**
12: $z_{\mathsf{P_{out}}(u)} \leftarrow y_u$
13: **end for**
14: $Z \leftarrow z_0||z_1||\ldots||z_{63}$
15: **return** Z



RF_0

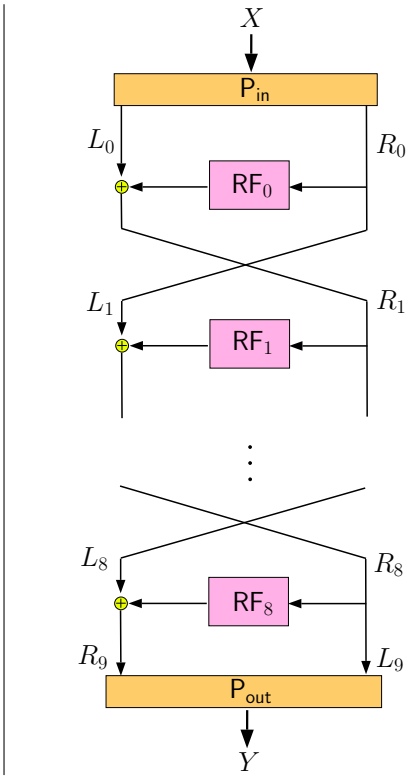


Figure 3: Note that RF_i and Enc algorithms takes the S-box and permutation definitions $(\mathsf{S}_{i,j}, \mathsf{P}_i, \mathsf{P_{in}}, \mathsf{P_{out}})$ as the implicit input. The drawing above depicts the 9-round Feistel network and an example choice for a round function RF_0 where the nibble permutation is sampled as $\mathcal{P}_0$.

Exercise 5 Nibblet Oracle

(We use the same Nibblet design from the previous question.)

The apprentice runs the Gen algorithm in Figure 3 and sets up an oracle that encrypts given multiple block messages with the ECB mode. The S-box and permutations are fixed for this oracle, but this configuration is hidden from you.

He uses the ECB mode to create a challenge for you. You are given a ciphertext Z that consists of ℓ blocks $Z_0||Z_1||\ldots||Z_{\ell-1}$, and asks you to recover the plaintext $X = X_0||\ldots||X_{\ell-1}$ such that $\mathsf{Enc}(X_i) = Z_i$ for all $i \in \{0, 1, \ldots, \ell - 1\}$. Z is given as a string of hexadecimal characters in `Q5_ct`, and you are similarly expected to return a string of ASCII characters in `Q5_m`, i.e. randomly chosen few sentences from an acclaimed literature piece.

The oracle is located at `lasecpc25.epfl.ch` and port `5559`. We provide you an example sage function on moodle that takes your sciper number, and a string of 16ℓ hexadecimal characters, i.e. ℓ blocks, and retrieves the ciphertext from the oracle.

Hint1: If you flip a single bit in your plaintext, how many and which bits of the ciphertext are also flipped?

Hint2: You can test your final answer with the oracle.