Homework sheet 3, due 05 December 2024 at 13:30

Make sure to justify your answers in detail and to give clear arguments. Document all steps, in particular of algorithms; it is not sufficient to state the correct result without the explanation. If the problem statement asks for usage of a particular algorithm other solutions will not be accepted even if they give the correct result.
Submit your homework by encrypted and signed email to all seven TAs (not Tanja). Send *one* single email to all TAs togeher, do not send individual emails to them; also cc your teammates. Do not forget to attach your public key and the public key of anybody you put in cc.
Note: email clients do support support multiple recipients in one encrypted email.

1. This exercise is about LFSRs. You know that A and B use an LFSR with state size 6. You observe a piece of ciphertext

   `10100 10010 00101 11010`

   and know that start of the message piece that is encrypted is `tue` and that the following encoding was used:

   ```
   a -> 00000
   b -> 00001
   c -> 00010
   ...
   z -> 11001
   0 -> 11010
   ...
   5 -> 11111
   ```

   The ciphertext is the bitwise xor of the message with the output stream of the LFSR. You don't get to see the IV or any earlier piece of ciphertext.

   (a) Compute the first 15 bits of the LFSR output used to encrypt the message $\boxed{\text{2 point}}$

   (b) Compute the state of the LFSR just before this piece of stream was generated and determine the feedback coefficients of the LFSR. $\boxed{\text{6 point}}$

(c) Compute the next character in the plaintext after `tue`. $\boxed{\text{2 point}}$

Note that the encoding used above is the binary representation of the integer associated to the letter

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Q	R	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

2. The cipher you studied on exercise sheet 3 is RC4.

   In RC4 we need to swap two states. This is easiest to do using an extra variable, i.e., we copy $S[i]$ to `dummy`, copy $S[j]$ to $S[i]$ and finally copy `dummy` to $S[j]$. To save on storage space one might have the idea to implement the swap in the following three steps:

   (a) $S[i] \leftarrow S[i]$ xor $S[j]$
   (b) $S[j] \leftarrow S[i]$ xor $S[j]$
   (c) $S[i] \leftarrow S[i]$ xor $S[j]$

   Explain first why this usually computes the correct $S[i]$ and $S[j]$. Now assume that this piece of code does the swap in the second part of the code (after the key setup). Explain why this can go wrong and state (with explanation) the expected number of steps until this goes wrong for the first time. Explain what happens long term with this implementation.
   Note that there are multiple possibilities of what happens. I don't expect a full analysis. $\boxed{\text{5 points}}$

3. This exercise expects you to brute force RC4 at “export-cipher” strength (40 bit = 5 byte keys). Through some side-channel information you learn that the first byte `key`[0] = 80. Find a key that could have produced the following as the first 10 bytes of the keystream

   130, 189, 254, 192, 238, 132, 216, 132, 82, 173.

   Note: this should be a feasible computation on a Laptop, but don’t start working on this on Wed evening. If it feels like more computation, chances are that something is wrong in your approach or that you’re using a very slow implementation. $\boxed{\text{5 points}}$