TP – Birthday attack against CBC

We want to implement the birthday attack against an encryption scheme built using the CBC mode of operation. We use block cipher(s) from the family SPECK[2], and more precisely we will use the three variants SPECK 32/64, SPECK 48/96 and SPECK 64/128 which have block size n and key size $2n$, for $n = 32$, 48 and 64 respectively.

General instructions.

- All needed files are found in the tarball: https://membres-ljk.imag.fr/Bruno.Grenet/IntroCrypto/25/cbc.tar.bz2. You must not modify these files!
- For you own implementations, **do not use any external library beyond the C standard library** (`stdxxx.h` and a few others such as `math.h`, `string.h`, ...).[3]
- Beyond the explicitly requested functions, you can (and are encouraged to) write any other function you need. In particular, it is good practice to avoid huge monolithic functions and to comment your code.
- Your code is due for March 28., 2025. *Details for the submission of your code will be given later.*

Contents of the tarball.

- `rand.h` and `rand.c` implement the `xoshiro256**` pseudo-random number generator[4] to be used in your program: the function `void random_bytes(uint8_t* c, size_t len)` fills `len` bytes of `c` with pseudo-random values.
- `speck.h` and `speck.c` implement the three variants of SPECK presented above, with two functions:

```
void speck_enc(const byte k[2*NBYTES], const byte m[NBYTES], byte c[NBYTES])
void speck_dec(const byte k[2*NBYTES], byte m[NBYTES], const byte c[NBYTES])
```

  - They take as inputs an array of $2n/8$ bytes for the key, and two arrays of $n/8$ bytes for the message block and the ciphertext block, for $n = 32$, 48 or 64.
  - The value `NBYTES` is defined to equal $n/8$ (depends on the variant of SPECK used), and `byte` is defined to be `uint8_t`.
  - The function `speck_enc` fills `c` with the encryption of `m`, and the function `speck_dec` fills `m` with the decryption of `c`.
- `test_speck.c` is a test file, that can serve as an example to write your own test files: It creates a random key and a random message block, encrypt the message block, and then decrypts it.

Exercise 1. *Warm up*

1. Download and extract the tarball.
2. Compile and execute `test_speck.c`, with a command-line such as

```
gcc rand.c speck.c test_speck.c -o test_speck -DBLOCKSIZE=<n>
```

   where `<n>` is 32, 48 or 64 depending on the variant of SPECK to use. Try the three variants.
   You can use the compiler you prefer!

[1] The original version is due to Pierre Karpman.
[2] Described in https://eprint.iacr.org/2013/404.
[3] Full list on Wikipedia: https://en.wikipedia.org/wiki/C_standard_library.
[4] Described in https://arxiv.org/abs/1805.01407.

Exercise 2. *CBC mode of operation*

In this part, write the implementations in `cbc.h` and `cbc.c`, and the tests in `test_cbc.c`.

1. Implement a function

   ```
   void cbc_enc(const byte k[2*NBYTES], const byte* m, byte* c, size_t mlen)
   ```

   that takes as inputs the $2n$-bit key, a message m with its byte-length `mlen`, and fills c with the encryption of m using the CBC mode of operation used with the block cipher SPECK.

   - You can assume that `mlen` is a multiple of `NBYTES`: no padding is needed to encrypt m.
   - Be careful that the message is represented as an array of bytes, that must be grouped in packets of `NBYTES` to feed the block cipher.
   - The encryption must be non deterministic: check it!

2. Implement a function

   ```
   void cbc_dec(const byte k[2*NBYTES], byte* m, const byte* c, size_t mlen)
   ```

   that takes as inputs the $2n$-bit key, a ciphertext c and the byte-length `mlen` of the corresponding message, and fills m with the decryption of c using the CBC mode of operation used with the block cipher SPECK.

   - What is the size of the ciphertext, compared to the size of the message?
   - Test that the decryption of a ciphertext correctly returns the original message.
   - Test with the three variants of SPECK.

Exercise 3. *The birthday attack*

In this part, write the implementations in `attack.h` and `attack.c`, and the tests in `test_attack.c`.

1. Remember what the birthday attack for CBC mode is, with the different parameters, and *write down the details on a scrap paper!*
2. Implement a function

   ```
   size_t challenge(byte** m, byte** c)
   ```

   that creates a (random) message m of byte-length `mlen`, computes the corresponding ciphertext with a (random) key, and returns the value `mlen`.

   - You have to determine what is the correct value for `mlen`.
   - The function takes care of allocating the necessary memory.

3. Implement the attack for your implementation of CBC, as a function

   ```
   void attack(const byte* c, size_t clen, size_t collision[2], byte xor[NBYTES])
   ```

   that takes as inputs a ciphertext c of byte-length `clen` corresponding to some (unknown) message m, and outputs a pair `collision` of indices i and j, and a block `xor` such that $m_i \oplus m_j =$ `xor`. The indices indicate the block number (between 0 and `clen`/`NBYTES`). If no collision is found, return $i = j = 0$ and `xor` $= 0^n$.

   - For what size of the ciphertext can you expect the attack to be successful?
   - Think about the data structures you need, and be careful with the space consumption.
   - For the compilation, you may want to use some optimization flag `-Olevel`.

4. Test your attack with SPECK 32/64: this should not take much more than one second.
5. Test your attack with SPECK 48/128: this should not take much more than a few seconds.
6. *(bonus)* Test your attack with SPECK 64/128: this may require to redesign the attack, and take some initiative to deal with the space usage.