5 February 2019

Personal Report **Instructions**
Course **Cryptography** 2018/2019
Prof. Massimiliano Sala

You were divided into the following ten teams.

1. ABEL: Flamini, Ibrisevic, Zanetti. **Project** Camellia.
2. BOREL: Foracchia, Sartori, Suleman. **Project** Camellia.
3. CANTOR: Piasenti, Tizianel, Pellegrino. **Project** Kasumi.
4. DIRICHLET: Pavone, Schicchi, Fidao. **Project** Kasumi.
5. EULER: Bracco, Turetta, Grisafi, Pellizzari. **Project** Present and Square Attack.
6. FIBONACCI: Mazzone, Carbonari, Irfani. **Project** Present and Square Attack.
7. GAUSS: Trentin, Dal Bianco, Di Liello **Project** AES and Square Attack.
8. HILBERT: Vicino, Stumpf, Ferronato. **Project** AES and Square Attack.
9. ITO: Tognolini, Elia, Del Sale. **Project** Twofish.
10. JACOBI: Cristiano, Piconese, Giambi. **Project** Twofish

All teams submitted a cipher (encryption and decryption), but since Present and AES are significantly easier than Camellia, Kasumi and Twofish, the teams 5-8 had also to implement the Square attack: teams 5-6 on BunnyTN and teams 7-8 on AES itself. We gave a working implementation of BunnyTN (since the square attack does not work for Present).

All teams **succeeded** in implementing their assigned job before the deadline 21/12/2018.

Starting from the (working) version of the cipher (encryption and decryption) submitted as a team job, each member of a team **may** derive her personal version, in which she can include improvements or additional features, **limited to** the encryption/decryption algorithm. It is also **possible** for the student to keep the submitted (team) version. We discourage personal implementation of the attack (and of BunnyTN, which was only an auxiliary cipher).

On this (personal or team) version of the cipher implementation, we **require** that each member of the team writes a personal report, in which:

1. She describes briefly the cipher
2. She describes briefly the attack
3. She comments on the implementation choices, for example:
    a. "In this function we used a look-up table because.."
    b. "Here we represented all bit-vectors as sequences, since.."
    c. "We avoided public variables because .. / We used a lot of public variables because .."

A report coming without a personal implementation **can** reach at **most** the mark 10/12.

Reports that aim at getting the maximum mark 12/12 **must** also:

1. Present a personal implementation
2. Discuss comparisons between the team's implementation choices and the student's implementation choices
3. Include timing comparisons showing the actual improvement achieved by the student's version

Report Format

There is no special format for the report, provided it does not exceed 15 pages in a 12-points pdf file. Pictures and screenshots can be added, as well as direct reference to the implementation (for example, "In function update() we defined variable X to denote ..").

A report describing a **personal** implementation must arrive via email **together** with a text file, containing the new MAGMA implementation of the cipher (we will test AGAIN the implementation with NEW test vectors).

SNIPPETS FROM OLD REPORTS

SNIPPET 1

****

`Measuring time`

Now we'll compare all CryptonoteAdGen versions. I measured the time for Team1 and my implementation (with and without precomputations) with MAGMA commands Cputime().
I averaged on 10 running of 2048 test vectors, for any implementation, and dividing the time between CryponoteKeyGen, Keccak and Base58. The following table summarizes the results on my laptop (XX2 is the version with precomputations).
Time (in seconds)

…….

…….

From the table we obtain that XX1 implementation improves Team1 CryptonoteGenKey by 56.8% and Keccak by 3.20%. XX2 further improves XX1 CryptoGenKey by 85.0% storing a relatively small amount of base point multiples. In total Team1 is improved by 30.8% from XX1 and by 72.0% from XX2.

****

SNIPPET2

****

Changes to the group implementation

The goals of my implementation were: to avoid as much as possible conversion between Sequences and Matrices; to eliminate redundant code; to reduce the use of memory space, allocating fewer objects. I provide examples for the three. First, the conversion from Matrix to Sequence and back again is expensive in terms of time and space, and it is in most cases useless. Compare AddRoundConstant in group 3:

```
function AddRoundConstantP(S,r);
A:=[Eltseq(S[j]): j in [1..8]];
A[1][1]:=BitwiseXor(A[1][1],r);
S:=Matrix(Integers(),A);
return S;
end function;
```

and the corresponding code in personal:

```
function AddRoundConstant(S, r, b);
if b eq 8 then r := BitwiseXor(r,0xff); end if;
S[b][1] := BitwiseXor(S[b][1],r);
return S;
end function;
```

The same operation - adding a constant byte to a specific entry of a Matrix - can be easily done without the use of Sequences (the meaning of b is explained below).
For the second goal, I noticed that the …….

****

SNIPPET 3

****

Comments on the team implementation

The public key PbKey is obtained from the compression of the point P = k B, obtained previously. Suppose the point is expressed in affine coordinates P = (x, y), to compress, we read the y-coordinate as a little-endian sequence of 32-bytes. The encoding of the point is obtained by copying the least significant bit of the x-coordinate to the most significant bit of the final byte of the y-coordinate.

```
function compress(P)
```

```
x := P[1];
y := P[2];
y := IntegerToSequence(y,256);
x := IntegerToSequence(x,2);
if (#x eq 0) then
x := [0];
end if;
y cat:= ZeroSequence(Integers(),32 - #y);
y[32]:=y[32]+(x[1]*(2^7));
return y;
end function;
```

This encoding allows to express a point through the y-coordinate and the sign of the x-coordinate. Through the curve equation it is possible to go back to the affine coordinates of the point.

****

SNIPPET 4

****

Comments on the team implementation

The inputs are first transformed into integers, using the function StringToInteger(), and then in sequences, using the function Intseq(). The rest of the algorithm works with sequences. When the function Intseq() is used, the length of the resultant sequence could not be the required one. To solve these problems, we designed the function RLength() that takes as input a sequence and an integer n and returns a sequence padded with zeros to the right in order to reach the desired length n. Moreover, the sequences in MAGMA have the less significant bit on the right and the most significant bit on the left. Therefore, we used the function Reverse() to reverse the sequence. These two operations could be inverted, but in this case we should have added the required zeros to the left. We preferred this choice because in the algorithm we had to use the function RLength() very often and padding to the right allows to save some reverse operations.

****