Enhanced Software Implementation of a Chaos-Based Stream Cipher

Guillaume Gautier*, Safwan El Assad†, Olivier Deforges*,
Sylvain Guilley‡, Adrien Facon‡, Wassim Hamidouche*
* Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, F-35000 Rennes, France
Email: guillaume.gautier@insa-rennes.fr, olivier.deforges@insa-rennes.fr, wassim.hamidouche@insa-rennes.fr
†Polytech Nantes, CNRS, IETR - UMR 6164, F-44000 Nantes, France
Email: safwan.elassad@univ-nantes.fr
‡ Secure-IC SAS, F-35510 Cesson-Sévigné, France
Email: sylvain.guilley@secure-ic.com, adrien.facon@secure-ic.com

***Abstract*—Cipher algorithms have been created a long time ago to protect sensitive information. With the evolution of technology, particularly the increase of computational power, the multiplication of devices, the interconnection of those devices, ciphers need to be created and/or enhanced to match challenges brought by this new environment. In general, chaos-based stream ciphers have three shortcomings: their implementation is not constant-time, they have weak keys, and are not portable. We show in this paper how to overcome those three limitations in the case of our stream cipher. The stream cipher performance including statistical analysis and computational performance are carried out and compared to state-of-the-art algorithms: Advanced Encryption Standard (AES)-CounTeR (CTR), HC-128 and Rabbit.**

Keywords*—*Chaos-based stream ciphers; Constant time; Statistical analysis; Computational performance.

I. INTRODUCTION

The need of encryption methods has nearly always existed to protect sensitive information. The number of connected devices is constantly and rapidly increasing. Those devices are communicating between each other through multiple channels exchanging information, such as sensor readings or orders to control other devices. In this context, the protection of sensitive data exchanged over networks is necessary. For this purpose, a secure cryptography that can be embedded into as many devices and architectures is, now more than ever, required. This means that algorithms are required to have the lowest complexity, and implementations have to reduce the energy consumption, the code size and the Random-Access Memory (RAM) without compromising security.

Stream ciphers are commonly used to encrypt data in real time applications like, for example, in selective video encryption [1][2]. It consists in performing an eXclusive OR (XOR) operation between a plain text and the output of a deterministic random generator. In the literature, multiple stream ciphers exist, the eSTREAM project was promoting the design of efficient and compact stream cipher such as HC-128 [3] or Rabbit [4], but according to [5], eSTREAM ciphers are not all secured.

The chaos theory is used in cryptography for its natural property of deterministic randomness. Indeed, chaos-based ciphers generally use chaotic maps for their combination of security and relatively low complexity.

This paper shows the different enhancements, in terms of both secure and embedded implementation, of the chaos-based stream cipher designed in [6][7] and implemented in [8][9].

The main contributions of this paper are the following.

- Remove the vulnerabilities to time-attack analysis, consisting in analysing execution time of secret-dependent operations in order to retrieve the secret key for example, a constant-time implementation is proposed.
- Propose a fixed-point implementation whereas the original stream cipher [8][9] uses a floating-point number representation [10] to widen the range of architectures able to embed the stream cipher.
- A new solution is proposed to correct the minor vulnerability inherent to the reduction operation.

The rest of this paper is organized as follows. In Section II, a functional presentation of the stream cipher and the associated generator is introduced. Then, Section III presents the enhancements brought to the previous implementation along the expected results. The stream cipher performance (statistical analysis and computational performance) are carried out and compared to AES-CTR, HC-128 and Rabbit algorithms in Section IV. Finally, Section V concludes this paper.

II. ORIGINAL CHAOS-BASED STREAM CIPHER

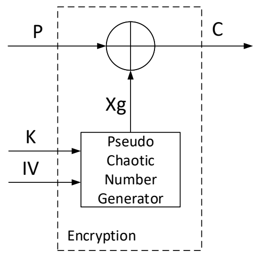
A. The Stream Cipher

The original stream cipher, based on a Pseudo-Chaotic Number Generator (PCNG), has been implemented in C [8][9]. As illustrated in Figure 1a, in order to obtain a ciphered text (C), the plain text (P) is encrypted using a XOR operation between the plain text and the PCNG output (X_g). The PCNG is initialized with a secret key (K) of length between 200 and 456 bits, depending on the number of internal delays, and a 64-bit-long Initial Vector (IV).

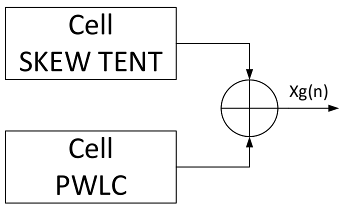
B. Pseudo-Chaotic Number Generator (PCNG)

The PCNG uses a couple of chaotic maps, the skew tent and the PieceWise Linear Chaotic (PWLC) map, to produce N-bit samples, with $N = 32$, at each instant n. The two maps are encapsulated in two different cells and the output cells ($X_s(n)$ and $X_p(n)$) are paired using a XOR operation as illustrated in Figure 1b.

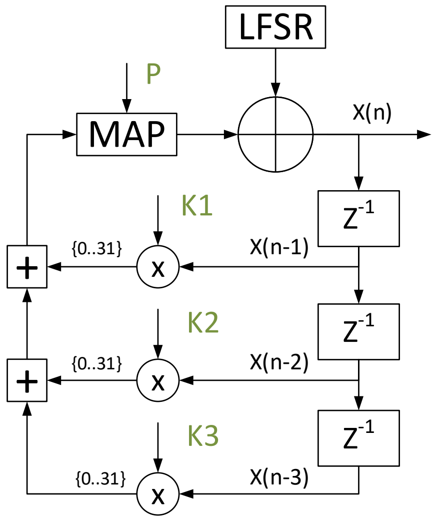
Figure 1c shows the block diagram of a cell where $X(n)$ can be $X_s(n)$ for Skew Tent map, or $X_p(n)$ for PWLC map,



(a) Stream cipher block diagram



(b) Association of the two PCNG cells



(c) Generic block diagram of a cell

Figure 1. Block diagram of the chaos-based stream cipher

P can be P_s or P_p and $K1,\ K2,\ K3$ can be $K1_s,\ K2_s,\ K3_s$ or $K1_p,\ K2_p,\ K3_p$, respectively.

Each cell in Figure 1c is composed of its own chaotic map. One cell is using the Skew Tent map defined by (1) and the other is using the PWLC map defined by (2).

$$X_s(n) = STmap(X_s(n-1), P_s) = \begin{cases} \left\lfloor 2^N \times \frac{X_s}{P_s} \right\rfloor & \text{if } 0 < X_s < P_s \\ \left\lfloor 2^N \times \frac{2^N - X_s}{2^N - P_s} \right\rfloor & \text{if } P_s < X_s < 2^N \\ 2^N - 1 & \text{otherwise} \end{cases} \tag{1}$$

$$X_p(n) = PLWCmap(X_p(n-1), P_p) = \begin{cases} \left\lfloor 2^N \times \frac{X_p}{P_p} \right\rfloor & \text{if } 0 < X_p < P_p \\ \left\lfloor 2^N \times \frac{X_p - P}{2^{N-1} - P_p} \right\rfloor & \text{if } P_p < X_p < 2^{N-1} \\ \left\lfloor 2^N \times \frac{2^N - P_p - X_p}{2^{N-1} - P_p} \right\rfloor & \text{if } 2^{N-1} < X_p < 2^N - P_p \\ \left\lfloor 2^N \times \frac{2^N - X_p}{P_p} \right\rfloor & \text{if } 2^N - P_p < X_p < 2^N \\ 2^N - 1 & \text{otherwise} \end{cases} \tag{2}$$

The map outputs are periodically perturbed using a Linear Feedback Shift Register (LFSR) [9] and are encapsulated inside an Infinite Impulse Response (IIR) filter with a variable order (1 to 3) (see Figure 1c). Increasing the filters' order will improve the statistical performance significantly.

In the cell Skew Tent, the parameter $P_s \in]0, 2^{32}[$ and the coefficients of the IIR filter $K1_s,\ K2_s,\ K3_s \in]0, 2^{32}[$ are part of the secret key, for PWLC, the parameter $P_p \in]0, 2^{31}[$ and $K1_p,\ K2_p,\ K3_p \in]0, 2^{32}[$, respectively.

C. Secret key and IV set-up

The first iteration is computed according to the following equations:

$$\begin{gathered} Xin_s = \left(MSB(IV) + \sum_{i=1}^{nbDelay} Xi_s \times Ki_s \right) mod\ 2^N \\ X_s(0) = STmap\left[Xin_s, P_s\right] \end{gathered} \tag{3}$$

$$\begin{gathered} Xin_p = \left(LSB(IV) + \sum_{i=1}^{nbDelay} Xi_p \times Ki_p \right) mod\ 2^N \\ X_p(0) = PLWCmap\left[Xin_p, P_p\right] \end{gathered} \tag{4}$$

where the values Xi_s and Xi_p are parameters of the key.

As shown in Equations (3) and (4), the 32 Most Significant Bits (MSB) of the IV are fed to the Skew Tent map and respectively the 32 Less Significant Bits (LSB) to the PWLC map.

III. Enhanced Software Implementation

A. Constant-Time Implementation

The implementation introduced in [8][9] showed secret-dependent timings. Indeed, the implementation profiling shows that the maps' computation is not constant since a branching is used to compare elements of the secret key and the complexity of each branch is different, resulting in different execution times. Branching is done by comparing X_s to P_s or X_p to P_p, as shown in Figure 2 for PWLC map given as an example.

```
Require: $X_p \in ]0; 2^{32}[$ and $P_p \in ]0; 2^{31}[$
  if $0 < X_p < P_p$ then
    $X_p \leftarrow X_p \times ratio3$
  else if $(P_p < X_p < M_2)$ then
    $X_p \leftarrow (X_p - P_p) \times ratio4$
  else if $M_2 < X_p < (M_1 - P_p)$ then
    $X_p \leftarrow (M_1 - P_p - X_p) \times ratio4$
  else if $(M_1 - P_p) < X_p < M_1$ then
    $X_p \leftarrow (M_1 - X_p) \times ratio3$
  else
    $X_p \leftarrow M_1 - 1$
  end if
  return $X_p$
where $M_1 = 2^{32}$, $M_2 = 2^{31}$ and ratios are defined in (5).
```

Figure 2. Calculate $X_p(n) = PLWCmap(X_p(n-1), P_p)$

Having secret-dependent timings is a vulnerability that an attacker can exploit to retrieve elements of the secret key. To overcome this problem, the proposed solution is detailed, as pseudo-code, in Figure 3. In order to achieve the same computational time and complexity for each sample, the maps compute, first, all the flags B_1 to B_5 used to determine which case should be selected. Then, the maps compute all the cases and masks them to select the correct output value. Similar modifications are applied to $STmap()$.

```
Require: $X_p \in ]0; 2^{32}[$ and $P_p \in [0; 2^{31}[$
  $B_1 \leftarrow 0 < X_p < P_p$
  $B_2 \leftarrow P_p < X_p < M_2$
  $B_3 \leftarrow M_2 < X_p < (M_1 - P_p)$
  $B_4 \leftarrow (M1 - P_p) < X_p < M_1$
  $B_5 \leftarrow (B_1 + B_2 + B_3 + B_4) = 0$
  $X_1 \leftarrow (X_p \times ratio3)\&\text{mask}(B_1);$
  $X_2 \leftarrow ((X_p - P_p) \times ratio4))\&\text{mask}(B_2)$
  $X_3 \leftarrow ((M_1 - P_p - X_p) \times ratio4))\&\text{mask}(B_3)$
  $X_4 \leftarrow ((M_1 - X_p) \times ratio3))\&\text{mask}(B_4)$
  return $(X_1 + X_2 + X_3 + X_4 + ((M_1 - 1)\&\text{mask}(B_5))$
where $M_1 = 2^{32}$, $M_2 = 2^{31}$, ratios are defined in (5) and mask($B_X$) returns 0xFFFFFFFF if $B_X = 1$, otherwise 0.
```

Figure 3. Calculate $X_p(n) = PLWCmap(X_p(n-1), P_p)$

B. Fixed-Point Implementation

In the C implementation of [8][9], the maps were computed using double-precision floating-point number representation [10], which cannot always be computed on embedded systems. The other drawback is the computational power required to perform such operation.

The software pre-calculates ratios for each maps. These ratios depend on the parameters P_s and P_p contained in the secret key and are defined as follows:

$$\begin{aligned} ratio1 &= \tfrac{2^{32}}{P_s}; \quad ratio2 = \tfrac{2^{32}}{2^{32}-P_s}; \\ ratio3 &= \tfrac{2^{32}}{P_p}; \quad ratio4 = \tfrac{2^{32}}{2^{31}-P_p} \end{aligned} \tag{5}$$

To match the double-precision floating-point standard [10] previously used, the 12.52 format is taken as the fixed point representation.

Due to the precision required to perform this computation of the maps, using the fixed-point ratio, at least 96-bit number is required. The computation consists in adding/subtracting 32-bit input, multiply it by the 64-bit ratio and then shift the result by 52 to obtain the result on 32 bits. The targeted platform (i.e., x86-64 Central Processing Unit (CPU)) computation is done on 128-bit words. The implementation of the fixed point ratios is described in Figures 4 and 5. Figure 4 shows how the pre-calculation of the ratio is performed and Figure 5 presents the implementation of the PWLC map, the same thinking is applied to the Skew Tent map.

```
  $ratio1 \leftarrow (M_1 << 52)/P_s$
  $ratio2 \leftarrow (M_1 << 52)/(M_1 - P_s)$
  $ratio3 \leftarrow (M_1 << 52)/P_p$
  $ratio4 \leftarrow (M_1 << 52)/(M_2 - P_p)$
where $M_1 = 2^{32}$ and $M_2 = 2^{31}$.
```

Figure 4. Computation of the ration using a fixed-point representation 12.52

```
Require: $X_p \in ]0; 2^{32}[$ and $P_p \in [0; 2^{31}[$
  $B_1 \leftarrow 0 < X_p < P_p$
  $B_2 \leftarrow P_p < X_p < M_2$
  $B_3 \leftarrow M_2 < X_p < (M_1 - P_p)$
  $B_4 \leftarrow (M1 - P_p) < X_p < M_1$
  $B_5 \leftarrow (B_1 + B_2 + B_3 + B_4) = 0$
  $X_1 \leftarrow ((X_p \times ratio3) >> 52)\&\text{mask}(B_1);$
  $X_2 \leftarrow (((X_p - P_p) \times ratio4) >> 52)\&\text{mask}(B_2)$
  $X_3 \leftarrow (((M_1 - P_p - X_p) \times ratio4) >> 52)\&\text{mask}(B_3)$
  $X_4 \leftarrow (((M_1 - X_p) \times ratio3) >> 52)\&\text{mask}(B_4)$
  return $(X_1 + X_2 + X_3 + X_4 + ((M_1 - 1)\&\text{mask}(B_5))$
where $M_1 = 2^{32}$, $M_2 = 2^{31}$, ratios are defined in (5) and mask($B_X$) returns 0xFFFFFFFF if $B_X = 1$, otherwise 0.
```

Figure 5. Calculate $X_p(n) = PLWCmap(X_p(n-1), P_p)$ using a fixed-point representation 12.52

C. Uniqueness of reduced products

Uniqueness of reduced products inside the IIR filter is primary. Indeed, the filter initialization being based on the secret key, filter output needs to be different for each keys, otherwise the generated sequence is the same. Two solutions are possible, the key space can be reduced to remove the weak keys or, as proposed below, to shift the result before the reduction to N bits, where N is the internal resolution of the chaotic maps, here $N = 32$.

Let $q = P(C = C')$ be the probability of having $C = C'$ with $C = A \times B$, $C' = A' \times B'$ and $A,\ A',\ B,\ B'$ being four distinct unsigned integers defined on N bits. Equation (6) presents the probability of having q in different cases. In our case, the generator is included in the second case, i.e., $q \neq 0$. The proposed solution aims to minimize the probability q.

$$\begin{cases} q = 0 & \text{if } C \text{ or } C' \text{ is defined on 2N-bits} \\ q \neq 0 & \text{if } C \text{ and } C' \text{ are defined on } M,\ M' \text{ bits,} \\ & \text{with } M,\ M' < 2N \end{cases} \tag{6}$$

Let $\epsilon(j)$ be equal to $1 << j$ with $\{j \in \mathbb{N} \mid j < N\}$ and let i be an integer in $[0; N-1]$ where i number of right shifts executed before the reduction to N bits. In the worst case, i.e., $A' = A,\ B' = B \oplus \epsilon(j)$ or $A' = A \oplus \epsilon(j),\ B' = B$, the probability q is equal to:

$$\begin{aligned} q_N(i) = &\ P((A \times B) >> i = (A \times (B \oplus \epsilon(N-1))) >> i) \\ &+ P((A \times B) >> i = (A \times (B \oplus \epsilon(0))) >> i) \\ = &\ 2^{-(i+1)} + 2^{-(N-i)} \end{aligned}$$

Figure 6 shows the value of q_N depending on the value i for $N = 32$. Minimum of q_{32} is obtained for $i = 15$ and $i = 16$. In the rest of the paper, we consider the value $i = 16$. The new generic block diagram of a cell using shifting is presented in Figure 7.

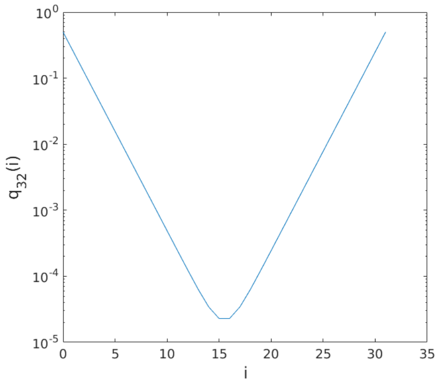


Figure 6. Probability $q_N(i)$ of having $A \times B = A' \times B'$ being four distinct unsigned integers defined on N bits, for $N = 32$

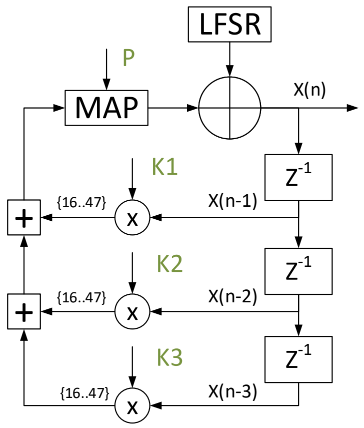


Figure 7. New generic block diagram of a cell using shifts

IV. Results and discussions

In this section, multiple versions of the cipher are implemented and tested.

- V0: this version corresponds to the initial version presented in [8][9].
- Shifting: this version is V0 that includes the enhancement presented in Section III-C.
- Fixed-Point: this version is V0 that includes the enhancement presented in Section III-B.
- Shifting + Fixed-Point: this version is the combination of the two previous versions.
- Constant time (CT): this version is the Shifting + Fixed-Point version with constant-time implementation presented in Section III-A.

A. Statistical Tests

To ensure the robustness of the enhanced implementations against statistical attacks, we perform the following statistical tests. The statistical tests are only run on the constant-time version. Similar results are obtained for all considered versions.

1) NIST Statistical Tests Suite (STS) SP 800-22: National Institute of Standards and Technology (NIST) STS [11] the popular test suite for investigating the randomness of binary data is applied. The suite contains 188 tests and sub-tests that assess the randomness of arbitrarily long binary sequences. These tests focus on a variety of different types of non-randomness that could exist in a sequence.

To perform the different tests, 100 sequences of 31250 32-bit samples (i.e., 1 million bits per sequence) are generated using 100 different secret keys. All 188 tests and sub-tests of the suite are run. For each test, a set of 100 P_{value} is produced and a sequence passes a test whenever the $P_{value} \geq \alpha = 0.01$, where α is the level of significance of the test. A value of α = 0.01 means that 1% of the 100 sequences are expected to fail. The proportion of sequences passing a test is equal to the number of $P_{value} \geq \alpha$ divided by 100.

Table I presents the NIST STS's results of the constant-time version. The P_{values} of all the tests are strictly over 0.01, meaning that the cipher passed all the tests. Passing this test is necessary, but not sufficient to affirm that generated sequences are random.

TABLE I. NIST STS RESULTS OF THE 3-DELAY CONSTANT-TIME STREAM CIPHER

Tests	P Value	Proportion of passed keys(%)
Frequency	0.51412	100.00
LinearComplexity	0.51412	99.00
LongestRun	0.16261	100.00
OverlappingTemplate	0.92408	98.00
RandomExcursions	0.21822	99.58
Rank	0.94631	100.00
BlockFrequency	0.00463	99.00
NonOverlappingTemplate	0.51879	98.96
ApproximateEntropy	0.22482	99.00
CumulativeSums	0.89412	100.00
Serial	0.21070	99.50
Universal	0.19169	99.00
Runs	0.07572	98.00
FFT	0.17187	98.00
RandomExcursionsVariant	0.40488	98.40

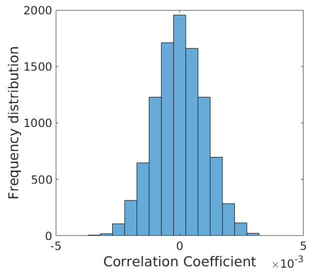
2) Correlation - Hamming Distance (HD): These tests show the non-similarity of two generated streams from two different keys.

The correlation coefficient is computed using the binary representation of the sequences where $1 \rightarrow 1$ and $0 \rightarrow -1$. The expected value of the correlation coefficient ρ_{ij}, for two completely random sequences, should be equal to 0.

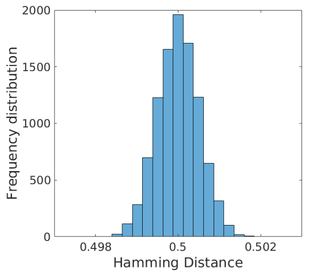
Figure 8a shows the obtained correlation coefficients between two-by-two different sequences. As we can see, all correlation coefficients are centred around 0 and maximum and minimum values are bounded by $3,94 \times 10^{-3}$, result expected for non-correlated sequences.

The average HD is defined in (7), where S_x is the generated sequence of size L, x is the index of a key inside an array of 100 random keys. The expected value, for two completely random sequences, should be equal to $\frac{1}{2}$.

$$HD_{ij} = \left\{ \frac{1}{L} \times \sum_{k=1}^{L} S_i(k) \oplus S_j(k) \quad \text{if } i \neq j \right. \tag{7}$$



(a)



(b)

Figure 8. Frequency distribution of the correlation coefficients (a) and hamming distances (b) of the 3-delay stream cipher

Figure 8b shows HDs centred around $\frac{1}{2}$, and maximum deviation is bound by $1,97 \times 10^{-3}$, meaning there is equal chance to generate a 0 or 1.

3) Histogram distribution: The aim of this test is to determine if the histogram distribution is uniform. To assert that, the χ^2 test is used. If a generated sequence verifies (8), the key associated passes the χ^2 test.

$$\sum_{i=0}^{C} \frac{(V_{observed}(i) - V_{expected})^2}{V_{expected}} < V_{critical} \tag{8}$$

This test is run on our algorithms, and some reference algorithms. The test conditions are the following.

- The test is run independently over 1000 randomly generated keys, and IVs.
- Samples are unsigned 32-bit integers.
- 10^8 samples are generated per sequence.
- $C = 1000$ classes are used.
- $V_{expected} = \frac{10^8}{C} = 10^5$
- $V_{critical}$ is computed using the inverse of the chi-square cumulative distribution function as defined in [12][13]. For this paper, $V_{critical} = 1073.6$.

Table II shows the percentage of keys passing the χ^2 test with a set of 1000 random keys and different algorithms.

The performance of literature algorithms is close to 95%. The initial version is only presenting 88,1% passing keys, but 94.6% keys for the enhanced version pass the test and is close to standard algorithms.

TABLE II. HISTOGRAM PERFORMANCE

Algorithm	Key passing χ^2 test
V0 - 3 delays	88.1%
Constant Time - 3 delays	94.6%
AES	94.9%
HC-128	95.4%
Rabbit	95.5%

B. *constant time measurement*

To check if the algorithmic meets with the constant time requirement, the Kalray Multi-Purpose Processing Array (MPPA®) manycore architecture [14] and a x86 platform are used.

a) On Kalray MPPA® processor: the MPPA® architecture is designed to achieve high energy efficiency, and deterministic response times for compute-intensive embedded applications.

The MPPA® processor , code-named Bostan, integrates 256 Very Long Instruction Word (VLIW) application cores and 32 VLIW management cores (288 cores in total) which can operate from 400 MHz to 600 MHz on a single chip and delivers more than 691.2 Giga FLOPS single-precision for a typical power consumption of 12 W. The 288 cores of the MPPA® processor are grouped in 16 Compute Clusters (CC) and implement two Input/Output Subsystems (IO) to communicate with the external world through high-speed interfaces via the PCIe Gen3 and Ethernet 10 Gbits/s.

MPPA® platforms integrate a register that counts the number of CPU cycles elapsed since the start of the machine. Indeed, it allows to measure a precise complexity of any algorithm ran on this architecture. To measure this complexity, a simple difference of two register readings, one before starting the encryption and one after, is performed.

The number of cycles measured is normalized to have the Number of Cycles per Byte (NCpB) (Equation (9)).

$$NCpB = \frac{C}{M \times K} \tag{9}$$

where C is the number of cycles elapsed since the start of the encryption, K is the number of keys used and M is the size, in bytes, of the message.

b) On INTEL® x86 processor: similar measurement method exists for INTEL® x86 processor, using Time Stamp Counter (TSC) register [15], but is not as precise. The reading of the TSC register returns the number of ticks elapsed since the start of the machine. The Number of Ticks per Byte (NTpB) is the unit used to compare the two implementations and is defined in (10).

$$NTpB = \frac{T}{M \times K} \tag{10}$$

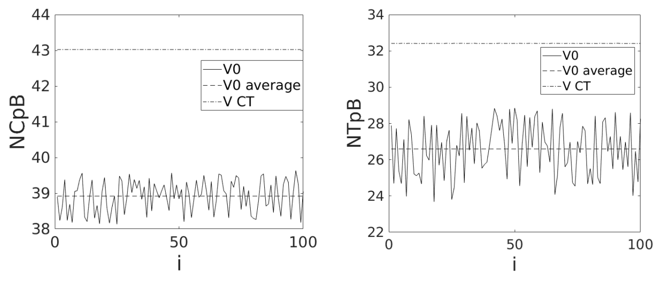
where T is the number of ticks elapsed since the start of the encryption, measured using TSC register, K is the number of keys used and M is the size, in bytes, of the message.

c) Results and discussions: to check the time stability of the constant-time version, 100 encryptions of a same 125000-byte-long message using 100 random keys are started on two different architectures, MPPA® processor (Figure 9a) and on x86 processor (Figure 9b).

As illustrated by Figures 9a and 9b, the number of cycles/ticks necessary to encrypt a byte in the initial version clearly depends on the key used, no matter which architecture is used. Oppositely, in the constant-time implementation, the NCpB/NTpB is constant, consequently removes the vulnerabilities to timing attacks.

C. *Time performances*

Time measurements are done on an Intel Core i7-6700 CPU @3.40GHz. The test environment is set as follows:



(a) on MPPA® Processor (b) on x86 Processor

Figure 9. NCpB/NTpB = f(Key[i]) of the initial version(V0) and the constant time(CT) version

- CPU frequencies are fixed at 3.00 GHz.
- Hyper-Threading is disabled.
- Pre-fetching is disabled.
- Process is assigned to a core using *taskset* command.

The function *gettimeofday()* is used to measure the time elapsed between the beginning and the end of the encryption. The message to encrypt is 125000 bytes long.

The metric used in this paper is defined as follows.

$$NCpB = \frac{F \times t}{M \times K} \tag{11}$$

where t is the time measured, K is the number of keys used, M is the size, in bytes, of the message and F is the frequency of the CPU. In this paper:

- $F = 3.00$ GHz.
- $M = 125000$ Bytes.
- $K = 100$ Keys.

Table III presents timing performance for different implementations of our cipher and some standard encryption methods. As shown in Table III, the constant-time version is a bit slower than other versions, but close to AES-CTR. HC-128 and Rabbit present better performance, however, these algorithms manifest some weaknesses against some attacks such as injection and side-channel attacks mentioned in [5].

V. Conclusion and Future Work

In this paper, we have proposed different enhancements for the original stream cipher implementation. The problem generated by product reductions is resolved by the patch presented in Section III-C. To secure the cipher against time attacks, one type of side-channel attack, we realized a constant-time implementation including all achieved enhancements.

The next step of this work would be to perform algebraic, side-channel and injection attacks for the initial and the constant-time versions to demonstrate the robustness of the cipher and its implementations. Then, a measurement of the energy consumption, the code size and the RAM needed for the cipher execution should be done to determine if the cipher can be categorized as lightweight.

Also, the initial and the constant-time versions will be implemented on embedded FPGA platform.

TABLE III. TIMING OF THE DIFFERENT CIPHER VERSIONS COMPARED TO STANDARD CIPHERS

Cipher version	delay	NCpB
V0	**1**	20.86
	2	22.11
	3	22.59
Shifting	**1**	20.82
	2	22.54
	3	23.43
fixed point	**1**	21.21
	2	22.24
	3	22.72
shifting + fixed-point	**1**	21.68
	2	22.79
	3	23.65
Constant-Time	**1**	24.46
	2	26.04
	3	27.06
HC-128		2.35
Rabbit		5.82
AES CTR		24.38

References

[1] S. Lian, J. Sun, J. Wang, and Z. Wang, "A chaotic stream cipher and the usage in video protection," *Chaos, Solitons and Fractals*, vol. 34, no. 3, pp. 851 – 859, 2007.

[2] W. Hamidouche, M. Farajallah, N. Sidaty, S. E. Assad, and O. Deforges, "Real-time selective video encryption based on the chaos system in scalable hevc extension," *Signal Processing: Image Communication*, vol. 58, pp. 73 – 86, 2017.

[3] H. Wu, "New stream cipher designs," M. Robshaw and O. Billet, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. The Stream Cipher HC-128, pp. 39–47.

[4] M. Boesgaard, M. Vesterager, and E. Zenner, *The Rabbit Stream Cipher*. Springer Berlin Heidelberg, 2008, pp. 69–83.

[5] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and Y. Papaefstathiou, "A survey of lightweight stream ciphers for embedded systems," *Security and Communication Networks*, vol. 9, no. 10, pp. 1226–1246, dec 2015.

[6] S. El Assad, H. Noura, and I. Taralova, "Design and analyses of efficient chaotic generators for crypto-systems," vol. 0, pp. 3–12, 10 2008.

[7] S. El Assad and H. Noura, "Generator of chaotic sequences and corresponding generating system," Patent WO2 011 121 218, Oct., 2011, extension internationale Brevets France n° FR20100059361 et FR20100052288. WO2011121218 (A1) 6/10/2011 CN103124955 (A) 29/05/2013 JP2013524271 (A) 17/06/2013 US2013170641 (A1) 3/07/2013.

[8] A. Arlicot, "Sequences Generator Based on Chaotic Maps," Université de Nantes, Tech. Rep., February 2014.

[9] M. A. Taha, S. E. Assad, A. Queudet, and O. Deforges, "Design and efficient implementation of a chaos-based stream cipher," *International Journal of Internet Technology and Secured Transactions*, vol. 7, no. 2, p. 89, 2017.

[10] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[11] L. E. Bassham et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," National Institute of Standards and Technology(NIST), Tech. Rep., 2010.

[12] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Government Printing Office, 1964, vol. 55.

[13] E. Kreyszig, *"Introductory Mathematical Statistics"*. John Wiley, 1970.

[14] B. D. de Dinechin, "Kalray MPPA®: Massively parallel processor array: Revisiting DSP acceleration with the Kalray MPPA Manycore processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE*. IEEE, 2015, pp. 1–27.

[15] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, M-Z," Tech. Rep., 2016.