symmetry

MDPI

Article

Security Analysis of the Symmetric Cryptosystem TinyJambu

Amparo Fúster-Sabater * and M. E. Pazo-Robles

Institute of Physical and Information Technologies (ITEFI), Consejo Superior de Investigaciones Científicas (CSIC), Serrano 144, 28006 Madrid, Spain; eugepazorobles@gmail.com
* Correspondence: amparo.fuster@csic.es

Abstract: Symmetric cryptography provides the best examples of cryptosystems to be applied in lightweight environments (e.g., IoT). A representative example is the cryptosystem TinyJambu, one of the ten finalists in the NIST Lightweight Cryptography Standardization Project. It is an authentication encryption with associated data scheme that is extremely lightweight and fast. In this work, we analyze the security of TinyJambu from two distinct and non-symmetric points of view: (1) the improvement of the best cryptanalytical attack found in the literature and (2) a randomness analysis of the generated sequences. Concerning item (1), we launched a differential forgery attack with probability $2^{-65.9487}$, which was improved considerably compared with previous numerical results. Concerning item (2), we analyzed the degree of randomness of the TinyJambu keystream sequences with a complete and powerful battery of statistical tests. This non-symmetric study shows the weakness of TinyJambu against cryptanalytic attacks as well as the strength of TinyJambu against statistical analysis.

Keywords: symmetric cryptography; TinyJambu; differential cryptanalysis; randomness; IoT

Citation: Fúster-Sabater, A.; Pazo-Robles, M.E. Security Analysis of the Symmetric Cryptosystem TinyJambu. *Symmetry* **2024**, *16*, 440. https://doi.org/10.3390/sym16040440

Academic Editors: Lorentz Jäntschi and Jie Yang

Received: 20 February 2024
Revised: 22 March 2024
Accepted: 2 April 2024
Published: 5 April 2024

1. Introduction

Nowadays, the Internet of Things (IoT) is an essential component in both information technologies and computer science. In our digital society, IoT technology is more and more deployed to connect diverse devices of daily use. All these connections need security, which is the use of cryptographic primitives. At the same time, the devices to be interconnected exhibit very different features: some of them are equipped with powerful processors that are able to support any cryptographic algorithm (public or private cryptography), while many others include just low-power microcontrollers; consequently, they are unable to support most of the current cryptosystems. Symmetric cryptography provides the only type of algorithm that can be applied in the frame of lightweight cryptography. In fact, this cryptography tackles the problem of protecting all these interconnected devices, considering their specific characteristics. Lightweight cryptography does not mean less secure cryptography. Indeed, it tries to keep the same level of security as that of traditional algorithms but under much more severe conditions.

In August 2018, "the National Institute of Standards and Technology (NIST) initiated a process to solicit and standardize lightweight cryptography algorithms to be deployed in constrained environments where the current NIST standards were not yet suitable"; see reference [1]. Given this call to action, the TinyJambu algorithm was submitted and was one of the ten finalists [2]. Moreover, due to its lightweight design and very simple components, it was the fastest among all of candidates submitted to this selection process.

Within symmetric cryptography, stream ciphers unify, in a single scheme, the characteristics of speed and simplicity. In conventional cryptography, a stream cipher design generates, from a short and truly random key, a long and pseudorandom binary sequence, the so-called keystream sequence. In a symmetric cryptosystem, in emission, a bitwise XOR operation between the original message (plaintext) and the keystream sequence is performed to generate the ciphertext. In reception, a bitwise XOR operation between the

ciphertext and the same keystream sequence is performed to recover the original message. A stream cipher is just an approximation of a one-time pad encryption [3] (p. 36), the only unconditionally secure cryptosystem. In fact, quantum cryptography and quantum key distribution aim to securely implement the one-time pad procedure, making use of quantum mechanics laws [4,5]. Currently, there are experimental applications of quantum key distribution, although they are not yet technically suitable for deployment as a standard component in lightweight environments. In cryptographic terms, stream cipher security lies in the quality of the keystream sequences or, equivalently, in the randomness degree of such sequences. In fact, keystream sequences should meet the following requirements: (a) the generated sequence must not be distinguishable from a truly random sequence; (b) the sequence must be unpredictable; (c) the period of the keystream sequence must be extremely large; (d) the key space of the sequence generator must be large enough to prevent a brute force attack; and (e) the design of the keystream generator should be resistant to any cryptanalytic attack reported in the literature. In brief, we studied the security of this cryptosystem from two different points of view: the improvement of the best cryptanalytical attack and a randomness analysis of the keystream sequence.

In this work, we analyzed the symmetric cryptosystem TinyJambu as a stream cipher algorithm, paying particular attention to above requirements (a), (b) and (e). In fact, we studied conditions (a) and (b), that is, the randomness and unpredictability of a TinyJambu bit sequence, by applying a number of empirical tests to determine its cryptographic suitability. In addition, we improved a forgery attack, for condition (e), which is the best cryptanalytical attack currently reported against the cryptosystem TinyJambu. The remaining conditions are included in the parameters and specifications of the algorithm itself, given by the designers.

In the original TinyJambu proposal [6], the authors analyzed the success probability of a forgery attack against a nonce introduction and quantified such a probability as 2^{-80}, where 80 was the number of active *AND* gates in a particular differential trail. Since the probability of 2^{-80} is extremely low, the designers claimed immunity for TinyJambu against this kind of cryptanalysis. Later, in reference [7], the authors developed a more refined model of differential trail search and evidenced the correlation among active *AND* gates. According to these additional features, they could compute more precisely the probability of forgery attack and provided a new probability of value $2^{-70.64}$.

In this work, we used the same search model developed in [7] and sought a maximum number of different trails with a minimum number of uncorrelated *AND* gates. The final result is the computation of a success probability of value $2^{-65.948}$, which is improved from previous numerical results. To compute such a probability, we adapted the programs provided in [8] and carried out a more exhaustive search of differential trails of type 3 on a keyed permutation of 384 rounds.

To obtain these numerical results, we used a desktop PC with a 13th Gen Intel (R) Core (TM) i9-13900K with 3.00 GHz, RAM of 128 GB with 24 cores and a Microsoft Windows 11 Pro operating system. Note that our computational resources were quite general. We also used Python version 3.11 64-bit and Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (win64) [9].

In the TinyJambu updated design [10], the number of rounds in the nonce introduction was increased to 640. The new design wins in security but loses in performance. At any rate, our results show a reduced security margin compared to that given by the designers (first version of TinyJambu with 384 rounds), although their impact was less for the 640-round updated version. In addition, we introduced a new strategy to simplify the trail search computation when the 640-round problem was tackled.

Concerning the randomness study, we analyzed in detail what would be the TinyJambu keystream sequence by concatenating the stream of bits that the algorithm uses in the phases of encryption/decryption. The generation of these sequences is symmetric in both emission and reception. Later, a wide and powerful battery of statistical tests (graphical tests, Diehard battery of tests or FIPS Test 140-2 from the NIST) were applied

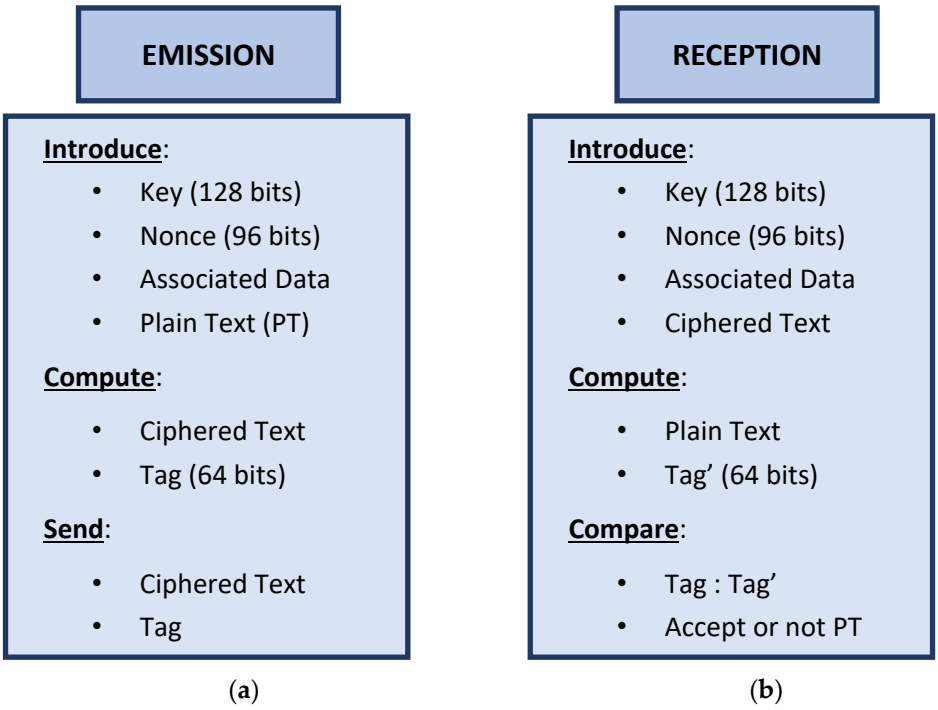
to these keystream sequences. To our knowledge, this is the first time that such a deep statistical analysis has been performed over TinyJambu sequences as most attacks against this cryptosystem have been conducted in the cryptanalyses of block ciphers [11,12].

The rest of this paper is organized as follows. In Section 2, we describe the TinyJambu candidate of the NIST Lightweight Cryptography Standardization Project. Next, in Section 3, we consider the security analysis found in the literature about this lightweight cryptosystem. In Section 4, we reduce the security margin of TinyJambu by improving the search for optimal differential trails and, consequently, the final differential probability in the nonce introduction phase. The 640-round problem is also considered. Next, we analyze the keystream sequence of this cryptosystem in terms of randomness. Finally, the conclusions and future work in Section 6 conclude the paper.

2. The TinyJambu Cryptosystem: An AEAD Scheme

The NIST Lightweight Cryptography Standardization Project solicited cryptographic proposals with double functionality: authentication and encryption both unified in the same algorithm (AE scheme). Moreover, such algorithms should incorporate the management of associated data that are additional non-confidential information such as headers, IP directions, routing protocol, etc., used in the communication process. In brief, an AEAD scheme (authentication and encryption with associated data scheme) was required for all candidates in the NIST call [13]. At the same time, one part of the information should be ciphered (plain text or confidential message), and another, part non-ciphered (nonce and associated data), but all data must be authenticated by means of a tag constructed for verification purposes. The required minimum sizes of the algorithm parameters were as follows: a key (k) of 128 bits, a nonce (N) of 96 bits and a tag (T) of 64 bits, while the length of the plain text (M) and associated data (AD) was variable. Notice that only (k) and (M) must be kept secret; the other parameters are public.

Figure 1 shows an exchange of information between a sender and receiver in the NIST operation mode.



(**a**) (**b**)

Figure 1. Operation mode required by the NIST: (**a**) steps performed in emission; (**b**) steps performed in reception.

In emission, the sender makes use of the key, nonce, associated data and plain text to compute the ciphered text and tag. Later, they send to the receiver both computed items. In reception, the receiver makes use of the same key, nonce, associated data and the received ciphered text to compute the plain text and their own tag (denoted by Tag'). Then, they compare the tag received with the computed Tag'. If both coincide, then the plain text is accepted; otherwise, it is rejected.

2.1. A Detailed Description of the TinyJambu Family

TinyJambu is one of the 10 finalists in the search for a standard algorithm in the Lightweight Cryptography Project. At the same time, TinyJambu is part of a family of lightweight authenticated encryption algorithms that supports three variants related to three different key sizes: 128 bits, 192 bits and 256 bits. The remaining parameters such as the nonce (public number of the message), a tag computed for verification purposes and a state (a 128-bit register that is the core of the cryptosystem) have exactly the same number of bits in all the members of the family.

In Table 1, the parameters of the three members of the TinyJambu family are listed.

Table 1. Parameters of the three members of the TinyJambu family.

Name	Key	Nonce	Tag	State
TinyJambu-128	128 bits	96 bits	64 bits	128 bits
TinyJambu-196	196 bits	96 bits	64 bits	128 bits
TinyJambu-256	256 bits	96 bits	64 bits	128 bits

TinyJambu mode is mainly based on a keyed permutation that provides authenticated encryption/decryption. Indeed, it uses a secret key permutation in the form of a Nonlinear Feedback Shift Register (*NLFSR*); see Figure 2. Such an *NLFSR* is made up of a 128-bit register and a feedback function; no key schedule algorithm is implemented. In turn, the register is made up of 128 interconnected memory cells simultaneously controlled by a unique clock. At each clock pulse, the content of each cell is shifted to the next cell on the right. For cell 127, new content is generated by means of the feedback function.

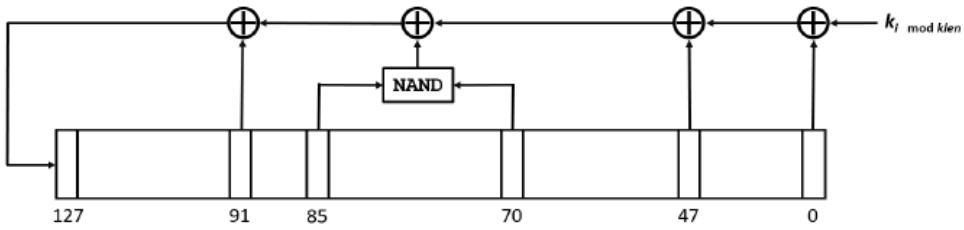


Figure 2. The 128-bit nonlinear feedback shift register in TinyJambu.

In the following, the following notation will be employed:

- S is the 128-bit state of the permutation, and S_i is the i-th bit of the state ($i = 0, \ldots, 127$), numbered from right to left.
- P_n is the 128-bit permutation with n rounds (that is, n shifts of the *NLFSR*).
- k_i ($i = 0, \ldots, 127$) is the i-th bit of the key (k), N_i ($i = 0, \ldots, 95$) is the i-th bit of the nonce (N), and T_i ($i = 0, \ldots, 63$) is the i-th bit of the tag (T).
- AD_i is the i-th bit of the associated data (AD), and M_i and C_i are the i-th bits of the plain text (M) and ciphered text (C), respectively.
- *FrameBits* is a three-bit function that takes different values depending on the application phase. More precisely, *FrameBits* = 1 for nonce introduction, *FrameBits* = 3 for associated data introduction, *FrameBits* = 5 for encryption/decryption, and *FrameBits* = 7 for the finalization process.

- For the $a, b \in \mathbb{F}_2 = \{0,1\}$ elements of the binary finite field, $\bar{a}$, $a \oplus b$, and $a{\cdot}b$ denote the bit-wise *NOT*, bit-wise exclusive *OR* and bit-wise *AND* logic operations, respectively.

In Figure 2, it can be noticed that *feedback* is the bit-wise *XOR* operation among three state bits (S_0, S_{47}, S_{91}), k_i is the i-th bit of the key, and the output of the *NAND* gate is represented by $\overline{S_{70}{\cdot}S_{85}}$. The unique nonlinear component in the feedback function is the logical *NAND* gate.

The permutation P_n is always the same but with a different number of rounds n depending on the phase in which it is applied. In the i-th round of the permutation, the 128-bit *NLFSR* is used to update the state as follows:

StateUpdate (S, k, i):
$feedback = S_0 \oplus S_{47} \oplus \overline{S_{70}{\cdot}S_{85}} \oplus S_{91} \oplus k_i$
for j from 0 to 126: $S_j = S_{j+1}$
$S_{127} = feedback$
end

For instance, P_{1024} means that the state of the permutation is updated using the function StateUpdate ( ) 1024 times. The subscripts of the key k are always computed modulo $|k|$; that is, $ki = ki \bmod |k|$. Remark that for permutation P_{1024}, the key is repeated eight times while for P_{384}, the key is repeated three times.

Due to the fact that this structure is a classical sponge for the 128-bit state and keyed permutation [6], TinyJambu is an extremely fast cryptosystem. Indeed, it was the fastest among all the candidates submitted to the NIST call [2]. According to the designers, the tapping bit positions were chosen in such a way that during the algorithm execution, 32 rounds of the permutation can be computed in parallel on 32-bit CPUs.

2.2. Operational Mode for TinyJambu

In the way it operates, we might consider TinyJambu a sequential mode of encryption/decryption similar to a stream cipher mode because the sponge structure allows it. The whole operational mode is as follows:

1. *Key Setup*: introduction of the key k by means of the permutation P_{1024}.
   Set the 128-bit state S as 0.
   Update the state using P_{1024}.
2. *Nonce setup*: introduction of the nonce N in three blocks of 32 bits with *FrameBits* = 1.
   for i from 0 to 2:
   $S_{\{36\ldots38\}} = S_{\{36\ldots38\}} \oplus FrameBits_{\{0\ldots2\}}$
   Update the state using P_{384}
   $S_{\{96\ldots127\}} = S_{\{96\ldots127\}} \oplus N_{\{32i\ldots32i+31\}}$
   end for
3. *Processing the blocks of associated data*: introduction of the different 32-bit blocks of associated data (*adlen* is the length in bits of *AD*). Here, *FrameBits* = 3.
   for I from 0 to $\lfloor adlen/32 \rfloor - 1$:
   $S_{\{36\ldots38\}} = S_{\{36\ldots38\}} \oplus FrameBits_{\{0\ldots2\}}$
   Update the state using P_{384}
   $S_{\{96\ldots127\}} = S_{\{96\ldots127\}} \oplus AD_{\{32i\ldots32i+31\}}$
   end for
4. *Encryption:* use *FrameBits* = 5, and the state is updated by means of the permutation P_{1024}. Next, sequentially introduce 32-bit blocks M_i of plain text to obtain 32-bit blocks C_i of ciphered text (*mlen* is the length in bits of M)
   for i from 0 to $\lfloor mlen/32 \rfloor - 1$:
   $S_{\{36\ldots38\}} = S_{\{36\ldots38\}} \oplus FrameBits_{\{0\ldots2\}}$
   Update the state using P_{1024}
   $S_{\{96\ldots127\}} = S_{\{96\ldots127\}} \oplus M_{\{32i\ldots32i+31\}}$
   $C_{\{32i\ldots32i+31\}} = S_{\{64\ldots95\}} \oplus M_{\{32i\ldots32i+31\}}$

end for

5. *Finalization:* this phase carries out the generation of the *tag* and uses *FrameBits* = 7. Notice that two different permutations P_{1024} and P_{384} are employed depending on which bits of the *tag* are being computed.
   $S_{\{36\ldots38\}} = S_{\{36\ldots38\}} \oplus FrameBits_{\{0\ldots2\}}$
   Update the state using P_{1024}
   $T_{\{0\ldots31\}} = S_{\{64\ldots95\}}$
   $S_{\{36\ldots38\}} = S_{\{36\ldots38\}} \oplus FrameBits_{\{0\ldots2\}}$
   Update the state using P_{384}
   $T_{\{32\ldots63\}} = S_{\{64\ldots95\}}$
6. *Operations in reception:* the receiver performs the following operations:
   Decryption: this is analogous to *Encryption* but substitutes $M_{\{32i\ldots32i+31\}}$ with $C_{\{32i\ldots32i+31\}}$ in order to recover the original 32-bit blocks M_i of plain text (M).
   Generation of T′: the receptor tag (T') is generated in the same way as the sender tag (T). Then, both tags are used for verification purposes.
   Verification: If $T' = T$, then accept the plain text (M); otherwise, reject it.

After the description of the TinyJambu operational mode, we proceed with the security analysis of this cryptosystem.

3. Security of TinyJambu against Cryptanalytic Attacks

The first security evaluation of TinyJambu was performed by the own designers in [6], the document submitted at the NIST Lightweight Cryptography Standardization Project. Later, a new security evaluation of TinyJambu with additional features was reported in [7]. The authors introduced a method for finding differential trails by means of Mixed Integer Linear Programming (MILP). Examples of the MILP application in cryptanalysis can be easily found in the literature [14–16]. Now, we analyze such an evaluation in detail.

3.1. The Concept of Active AND Gates

Differential cryptanalysis has been applied to the nonlinear part of the cryptosystem. Since the *NAND* logic operation is the only nonlinear component of the feedback function, it is the natural target for launching a differential attack [14]. The key idea of this cryptanalysis is to introduce two slightly different initial states in two parallel versions of TinyJambu and analyze the evolution of these differences along the encryption process. Indeed, we denote with S and S^* two different 128-bit states of TinyJambu. As before, S_i and S_i^* correspond to the i-th bits of both states ($i = 0, \ldots, 127$), respectively. Thus, the initial differential between states S and S^* is defined as

$$\Delta S_i = S_i \oplus S_i^* \text{ for } (i = 0,\ 1,\ \ldots, 127). \tag{1}$$

After l rounds of the keyed permutation P_l, the state bits S_i and S_i^* have been simultaneously shifted as described in Section 2. That is, for state S, the successive feedback results satisfy the following recurrence relationship:

$$S_{128+j} = S_{0+j} \oplus S_{47+j} \oplus \overline{S_{70+j} \cdot S_{85+j}} \oplus S_{91+j} \oplus k_j \quad (j = 0,\ 1, \ldots, l), \tag{2}$$

and similarly for state S^*,

$$S^*_{128+j} = S^*_{0+j} \oplus S^*_{47+j} \oplus \overline{S^*_{70+j} \cdot S^*_{85+j}} \oplus S^*_{91+j} \oplus k_j \quad (j = 0,\ 1, \ldots, l). \tag{3}$$

Summing up Equations (2) and (3), we obtain a differential that also satisfies its own recurrence relationship and gives rise to a differential trail of the form

$$\Delta S_{128+j} = \Delta S_{0+j} \oplus \Delta S_{47+j} \oplus \Delta\left(\overline{S_{70+j} \cdot S_{85+j}}\right) \oplus \Delta S_{91+j} \quad (j = 0,\ 1, \ldots, l). \tag{4}$$

Notice that in Equation (4), the only nonlinear term is the differential of the *NAND* logic operation notated:

$$\Delta\left(\overline{S_{70+j}\cdot S_{85+j}}\right) = \left(\overline{S_{70+j}\cdot S_{85+j}}\right) \oplus \left(\overline{S^*_{70+j}\cdot S^*_{85+j}}\right) \quad (j = 0,\ 1, \ldots, l). \tag{5}$$

This differential propagates along the keyed permutation P_l and tells us when a difference between the nonlinear term of both versions takes place.

Then, two different items can be pointed out:

1. According to Equation (4), it can be noticed that in the differential trail, the terms k_j of the key have been cancelled as they are just additive elements.
2. Since the complementation of a logic product is defined as $\overline{a\cdot b} = a\cdot b \oplus 1$, we can omit the constant 1 that appears in both summands of Equation (5) and replace the *NAND* gate for an *AND* gate. In this way, we obtain a new and more simplified equation in terms of an *AND* gate:

$$\Delta\left(S_{70+j}\cdot S_{85+j}\right) = \left(S_{70+j}\cdot S_{85+j}\right) \oplus \left(S^*_{70+j}\cdot S^*_{85+j}\right) \ (j = 0,\ 1, \ldots, l). \tag{6}$$

In the following and for the sake of simplicity, we will just use the *AND* logic operation. Now, we define a new concept related to such an operation.

An *active AND gate* is defined as a differential of value 1. That is,

$$\Delta(a\cdot b) = 1 \tag{7}$$

where $a,\ b \in \mathbb{F}_2 = \{0,1\}$. In TinyJambu, an active *AND* gate, notated as $\Delta\left(S_{70+j}\cdot S_{85+j}\right) = 1$, determines the difference of the nonlinear component between the two parallel versions of the generator, which allows us to perform this differential cryptanalysis.

In addition, we denote with X the number of active *AND* gates through the permutation P_l. Since a trail with score X can be satisfied with probability $p = 2^{-X}$ (see [7], Section 3), the challenge is to look for differential trails, as those defined in Equation (4), that minimize the number X of active *AND* gates after l rounds. In TinyJambu, this kind of attack is launched in the initialization phase (either in the nonce or associated data setup) as both initializations make use of the keyed permutation P_l with $l = 384$, which is the minimum number of rounds in the cryptosystem design. As long as the probability p satisfies the inequality $p \geq 2^{-64}$ (see [7], Section 4.2), a differential attack breaks the 64-bit security claimed by the designers in [6].

3.2. The Concept of Correlated AND Gates

As stated in the original document of TinyJambu [6], the designers looked for differential trails with a minimum number of active *AND* gates. Nevertheless, they considered that all the *AND* gates are independent. By contrast, in [7], the authors analyzed the possible correlations of the *AND* gates that modify the number of active *AND* gates along the keyed permutation P_l.

Indeed, according to Figure 2, the separation between cells entering the *NAND* gate is 15. This means that if a bit a in cell 85 enters the gate at a particular round, then 15 rounds later, the same bit will enter the same gate in cell 70. More precisely, according to the TinyJambu recurrence relationship, bit S_{85} will be an input of the *NAND* gate with bit S_{70}, but after 15 rounds, S_{85} will also be an input of the gate with bit S_{100}. In reference [7], the authors studied the possible correlations between the logic products $\left(S_{85+j}\cdot S_{70+j}\right)$ and $\left(S_{100+j}\cdot S_{85+j}\right)$ when the bits of the register shift along the successive rounds.

Let $\Delta\left(S_{85+j}\cdot S_{70+j}\right)$ and $\Delta\left(S_{100+j}\cdot S_{85+j}\right)$ denote the output difference of the *AND* gate for inputs $\left(S_{85+j}, S_{70+j}\right)$ and $\left(S_{100+j}, S_{85+j}\right)$, respectively. In [7] (pp. 161–162), it was

proved that if $(\Delta S_{100+j},\ \Delta S_{85+j},\ \Delta S_{70+j}) = (1,\ 0,\ 1)$ and $S_{85+j} = 1$ for any j through the permutation P_l , then

$$\Delta(S_{85+j} \cdot S_{70+j}) = \Delta(S_{100+j} \cdot S_{85+j}) = 1. \tag{8}$$

That is, $\Delta(S_{85+j} \cdot S_{70+j}) = 1$ is an active *AND* gate, but 15 rounds later, $\Delta(S_{100+j} \cdot S_{85+j}) = 1$ is an active *AND* gate too. Both differentials will be active *AND* gates or not depending on the value of S_{85+j}. In fact, if $S_{85+j} = 1$, then Equation (8) holds, and there are two *AND* gates. On the other hand, if $S_{85+j} = 0$, then $\Delta(S_{85+j} \cdot S_{70+j}) = \Delta(S_{100+j} \cdot S_{85+j}) = 0$, and there are no active *AND* gates. Both cases happen with probability 2^{-1}, as it depends on a single bit. Under these conditions, both active *AND* gates exist or not jointly, and both gates can be counted as a single active *AND* gate. This is a different and more favorable way of counting the number of active *AND*s. Thus, in this more refined model proposed in [7], the number X of active gates is reduced through l rounds, and consequently, the success probability $p = 2^{-X}$ of a differential attack is incremented.

3.3. Differential Probabilities

In reference [6], the designers searched for differential trails under four different constraints related to the bit positions S_i and S_i^* of the states before and after the permutation P_l . The four types of differences are enumerated as follows:

Type 1 differences: Input differences only exist at the 32 most significant bits (MSBs) of the input differential, that is, $\Delta S_{\{127\ldots96\}}$, with no constraints on the output differential.

Type 2 differences: No constraints on the input differential while output differences only exist at the 32 MSBs of the output differential, that is, $\Delta S_{\{511\ldots480\}}$.

Type 3 differences: Both input and output differences jointly exist at the 32 MSBs in input and output differentials, that is, in $\Delta S_{\{127\ldots96\}}$ and $\Delta S_{\{511\ldots480\}}$, respectively.

Type 4 differences: No constraints at all.

Concerning TinyJambu, type 3 is the most interesting scenario since, as described in item 2 of Section 2.2, the cryptosystem introduces a 96-bit nonce to three blocks of 32 bits each. Thus, in type 3, the input and output differences of an optimal trail can be injected as if they were the first and second 32-bit nonce blocks, respectively. Later, taking the previous output difference as the new input difference, we can compute a new optimal trail and its output difference can be injected as the third 32-bit nonce block. The type 3 differential trails allow this manipulation in the nonce introduction. Thus, the differential cryptanalysis exploits the introduction of the three nonce blocks in the frame of a nonce forgery attack. In brief, in this nonce setup phase, we have a keyed permutation with 384 rounds and an introduction of just three 32-bit blocks. The associated data introduction also allows the exploitation of this kind of forgery attack, but in such a case, the number of 32-bit blocks to be introduced may be larger.

According to [6], the maximum probability found by the designers for a differential trail of type 3 was $p = 2^{-80}$, which is enough to guarantee 64-bit security. Nevertheless, in [7] (p. 166), the authors, making use of their refined model with correlated *AND* gates, found a better differential trail, and its evolution is presented in Table 2 in hexadecimal format. It is a differential trail type 3 with l = 384 rounds. Moreover, along this trail, they counted X = 88 active *AND* gates with 14 correlated gates. Therefore, their success probability is $p = 2^{-88+14} = 2^{-74}$. Later, they computed the differential probability by identifying a cluster of trails with the same input and output differentials but with different values of X. The number of trails associated with each probability (#Trails) is depicted in Table 3.

By summing up the probability of each trail multiplied by the number of trails found, they computed the following differential probability:

$$p = 1{\cdot}2^{-74} + 5{\cdot}2^{-75} + 9{\cdot}2^{-76} + 14{\cdot}2^{-77} + 20{\cdot}2^{-78} + 24{\cdot}2^{-79} + 30{\cdot}2^{-80}.$$

Therefore, the final differential probability given in [7] is

$$p = 2^{-70.68}, \tag{9}$$

which is higher than the first probability $p = 2^{-80}$ obtained by the designers and referenced in [6].

Table 2. A differential trail type 3 with probability 2^{-74} for 384 rounds.

Input: $\Delta S_{127\ldots0}$	01004800	00000000	00000000	00000000
$\Delta S_{255\ldots128}$	81044c80	24080304	d9200000	22090000
$\Delta S_{383\ldots256}$	81004082	00010200	83000010	26090240
Output: $\Delta S_{511\ldots384}$	81004082	00000000	00000000	00000000

Table 3. Differential trails with the same input/output differential masks defined in Table 2.

Probability	2^{-74}	2^{-75}	2^{-76}	2^{-77}	2^{-78}	2^{-79}	2^{-80}
#Trails	1	5	9	14	20	24	30

4. Reducing the Security Margin of TinyJambu against Differential Cryptanalysis

In view of the previous results, in this section, we introduce new numerical values that we obtained after a more detailed and complete search of differential trails for 384 rounds. Next, a strategy sketch used to improve the search of trails for a number of rounds greater than 384 is also described.

4.1. Security Margin with 384 Rounds

Our goal was to look for differential type 3 trails with a number of uncorrelated *AND* gates X < 74, as 74 is the lowest value reported by the authors in [7]. To fulfill this task, we used the Gurobi Optimizer [9] and the refined model developed in [7].

First of all, we looked for the optimal number of active *AND* gates X through a type 3 trail with l = 384 rounds in the TinyJambu feedback function. According to the Gurobi Optimizer, the optimal value of uncorrelated *AND* gates in this scenario is X = 71. More precisely, the optimizer provided us with a differential trail of type 3 with 84 active *AND* gates and 13 correlated gates. Thus, this differential trail propagates with probability $p = 2^{-84+13} = 2^{-71}$, as described in Table 4.

Table 4. A differential trail type 3 with probability 2^{-71} for 384 rounds.

Input: $\Delta S_{127\ldots0}$	048a2000	00000000	00000000	00000000
$\Delta S_{255\ldots128}$	44800001	12000986	48800020	91440000
$\Delta S_{383\ldots256}$	40800441	00008100	0880000c	12009120
Output: $\Delta S_{511\ldots384}$	40800441	00000000	00000000	00000000

The first row of Table 4, including $\Delta S_{\{127\ldots0\}}$, corresponds to the initial differential, while the following rows, including $\Delta S_{\{255\ldots128\}}$, $\Delta S_{\{383\ldots256\}}$ and $\Delta S_{\{511\ldots384\}}$, correspond to the differentials after 128, 256 and 384 rounds, respectively. Notice that in type 3 differences, the input/output differentials are just the values in $\Delta S_{\{127\ldots96\}}$ / $\Delta S_{\{511\ldots480\}}$, respectively, as the remaining differences in both $\Delta S_{\{95\ldots0\}}$ and $\Delta S_{\{479\ldots384\}}$ equal 0. Table 4 as a whole shows the differential propagation through the permutation P_{384} with information about the intermediate differences.

Once the minimum value of uncorrelated active *AND* gates was obtained, we honed in on a long interval of possible solutions provided by the Gurobi. Proceeding in this way, we could identify a cluster of multiple trails with the same input $\Delta S_{\{127\ldots0\}}$ and output

$\Delta S_{\{511...384\}}$ differential masks, which appear in Table 4. All these trails start and end at the same differentials, but the intermediate differences and the number X of *AND* gates may be different. The distribution of such trails (#Trails) is depicted in Table 5, where the number X of active *AND* gates ranges in the interval X ϵ [71, ..., 77]. As before, by summing up all of these probabilities, we obtained a final differential probability of value

$$p = 9{\cdot}2^{-71} + 24{\cdot}2^{-72} + 27{\cdot}2^{-73} + 28{\cdot}2^{-74} + 18{\cdot}2^{-75} + 14{\cdot}2^{-76} + 22{\cdot}2^{-77}.$$

$$p = (33.15625)\, 2^{-71}.$$

Thus, our final differential probability is

$$p = 2^{-65.9487}. \tag{10}$$

According to Equation (10), it can be noticed that the new differential probability is a good approach to the inequality $p \geq 2^{-64}$ that breaks the 64-bit security.

Table 5. Differential trails with the same input/output differential masks defined in Table 4.

Probability	2^{-71}	2^{-72}	2^{-73}	2^{-74}	2^{-75}	2^{-76}	2^{-77}
#Trails	9	24	27	28	18	14	22

Table 5 is a much improved version of the first numerical values obtained by the authors of this work in [17], as the number of differential trails has significantly increased. In addition, the interval of the values of X is also greater than that in [17].

Table 6 shows a comparison between the results of Tables 3 and 5. It can be noticed that our best probability is $2^{-71,}$ while that of Saha et al. is just 2^{-74}. Moreover, in this work, the number of trails associated with each probability is greater than the number of trails found in [7] for the same probabilities.

Table 6. Comparison between results given in reference [7] and those provided in this work.

Saha et al. [7]							
Probability	2^{-74}	2^{-75}	2^{-76}	2^{-77}	2^{-78}	2^{-79}	2^{-80}
#Trails	1	5	9	14	20	24	30
This work							
Probability	2^{-71}	2^{-72}	2^{-73}	2^{-74}	2^{-75}	2^{-76}	2^{-77}
#Trails	9	24	27	28	18	14	22

Considering the scores of the previous differential trails X = 80 for the TinyJambu designers, X = 74 for the authors of reference [7] and X = 71 for our optimal trail, we can compare the success probabilities $P_{succ.}$ for a differential attack in all these cases,

$$P_{succ.} = \frac{1}{2^{71}} > \frac{1}{2^{74}} > \frac{1}{2^{80}}\,, \tag{11}$$

as well as the differential probabilities $P_{dif.}$ in all these cases,

$$P_{dif.} = \frac{1}{2^{65.9487}} > \frac{1}{2^{70.68}} > \frac{1}{2^{80}}. \tag{12}$$

Our final differential probability is much greater than the probability of value $p = 2^{-80}$ obtained by the designers and rather greater than the probability $p = 2^{-70.68}$ obtained by the authors of [7]. The new differential probability dramatically increments the success probability for a cryptanalytical attack.

In this subsection, we provided Tables 4 and 5 with their corresponding numerical results. We can say that we found much more differential trials with higher success probabilities for a differential cryptanalysis of type 3 than those ones obtained by previous authors. These numerical results are the best we can obtain at this point with our computational resources, that is, with the PC for which its specifications are described in Section 1. Better computational facilities would allow us to find more differential trails with the optimal value X = 71 and, consequently, to increase our final differential probability.

4.2. Differential Trail Search with 640 Rounds

Since the updated version of TinyJambu [10] increases the number of rounds in the nonce introduction to 640, we can analyze the effect of this increment on the differential trail search. First of all, it must be noticed that with a single PC, it is unfeasible to address the search of differentials when the keyed permutation is P_{640} .

At any rate, we can sketch a strategy that allows us to partially tackle this computation. Indeed, we observed in Tables 2 and 4 (as well as in many other differentials obtained in this work) that the 32 most significant bits in the two last rows of the tables coincide. That is, $\Delta S_{383\ldots352} = \Delta S_{511\ldots480}$ in all the examples. The justification for such a coincidence is as follows.

For instance, in Table 4, the row before the last is

$\Delta S_{383\ldots256}$	40800441	00008100	0880000c	12009120

Then, after 96 rounds the differences would be

$\Delta S_{479\ldots352}$	00000000	00000000	00000000	40800441

Remark that $\Delta S_{\{479\ldots384\}} = 0$ means that

$$\Delta S_i = S_i \oplus S_i^* = 0 \quad \text{for}(i = 384, \ldots, 479), \tag{13}$$

so that the state bits S_i and S_i^* are equal in the range $(i = 384, \ldots, 479)$.

Next, according to the TinyJambu recurrence relationship, the new state bits S_i and S_i^* computed in the following 32 rounds would be as follows:

$$S_{480+j} = S_{352+j} \oplus S_{399+j} \oplus \left(\overline{S_{422+j} \cdot S_{437+j}}\right) \oplus S_{443+j} \; (j = 0,\ 1, \ldots, 31)$$

$$S^*_{480+j} = S^*_{352+j} \oplus S^*_{399+j} \oplus \left(\overline{S^*_{422+j} \cdot S^*_{437+j}}\right) \oplus S^*_{443+j} \quad (j = 0,\ 1, \ldots, 31).$$

Therefore, the differential can be written as

$$\Delta S_{480+j} = \Delta S_{352+j} \oplus \Delta S_{399+j} \oplus \Delta\left(\overline{S_{422+j} \cdot S_{437+j}}\right) \oplus \Delta S_{443+j} \quad (j = 0,\ 1, \ldots, 31).$$

Thus, according to Equation (13), the following equalities hold:

$$\Delta S_{399+j} = S_{399+j} \oplus S^*_{399+j} = 0 \quad (j = 0,\ 1, \ldots, 31).$$

$$\Delta S_{443+j} = S_{443+j} \oplus S^*_{443+j} = 0 \quad (j = 0,\ 1, \ldots, 31).$$

$$\Delta\left(\overline{S_{422+j} \cdot S^*_{437+j}}\right) = \left(\overline{S_{422+j} \cdot S_{437+j}}\right) \oplus \left(\overline{S^*_{422+j} \cdot S^*_{437+j}}\right) = 0 \quad (j = 0,\ 1, \ldots, 31).$$

Since,

$$S_{422+j} = S^*_{422+j} \quad (j = 0,\ 1, \ldots, 31).$$

$$S_{437+j} = S^*_{437+j} \quad (j = 0,\ 1, \ldots, 31).$$

Consequently,

$$\Delta S_{480+j} = \Delta S_{352+j} \oplus 0 \oplus 0 \oplus 0,$$

and the 32 most significant bits in the two last rows of the tables coincide; that is,

$$\Delta S_{\{383\ldots352\}} = \Delta S_{\{511\ldots480\}}$$

The reasoning is general for any differential trail with a number of rounds l = 384, 512, 640, etc., or for any multiple of 128, provided that the output differential is type 3; that is, not all of the 32 most significant bits are 0, but the remaining bits equal 0.

This coincidence can be successfully exploited to tackle the trail search along 640 rounds. In fact, the computational problem can be split into two different search processes:

1. Take an input differential of type 3.
2. Search for a differential trail along 512 rounds with a number X of *AND* gates as low as possible. Upon obtaining $\Delta S_{\{639\ldots512\}}$, take the 32 most significant bits $\Delta S_{\{639\ldots608\}}$.
3. Search for a differential trail along 128 rounds with $\Delta S_{\{639\ldots512\}}$ as the input differential and $\Delta S_{\{767\ldots640\}} = \Delta S_{\{639\ldots608\}}$ 00000000 00000000 00000000 as the output differential.
4. Obtain a differential trail of type 3 with 640 rounds.

The obtained trail is probably not the optimal trail, but it exhibits a suitable number of *AND* gates, and it was obtained with low computational resources.

5. Statistical Analysis of Randomness

Traditionally, all the cryptanalytical attacks launched against TinyJambu have been conducted in block cipher cryptanalysis, (e.g., differential and linear cryptanalyses). Nevertheless, the cryptosystem TinyJambu can be considered a stream cipher cryptosystem in that both encryption and decryption are concerned. This is the reason why an analysis of the sequence generated using TinyJambu (the so-called keystream sequence) deserves a section in this work. To our knowledge, a deep study of such a sequence cannot be found in the literature. The analysis of this sequence is a way of evaluating TinyJambu's strength.

The encryption/decryption procedure in TinyJambu is a typical stream cipher procedure that is carried out as follows:

- The encryption is just the bit-wise XOR operation between each 32-bit block M_i of plain text and the contents of stages $S_{\{64\ldots95\}}$ in the NLFSR.
- The decryption is just the bit-wise XOR operation between each 32-bit block C_i of ciphered text and the same contents $S_{\{64\ldots95\}}$ in the NLFSR stages.

Notice that TinyJambu is not a keystream generator in a classical sense [3] (pp. 29–30); that is, it does not generate a long binary sequence for cryptographic purposes. Nevertheless, we can concatenate the contents of stages $S_{\{64\ldots95\}}$ used in the successive M_i encryptions and generate the corresponding keystream sequence. It is over such a concatenated sequence that we applied the different statistical tests.

As the initialization of TinyJambu depends on several parameters (key, nonce, associated data, plain text), for our randomness analysis, we generated multiple keystream sequences originating from different choices of random keys, nonces and AD, as well as distinct types of plain texts (e.g., text files, images, videos, etc.). The lengths of our sequences were 2^{23} bits. Moreover, we made use of three kinds of tests: graphical tests [18,19], the Diehard battery of tests [20,21] and the family of statistical tests FIPS 140-2 developed by the NIST [22]. For more details concerning the meaning and assessment of such tests, see references [18,19]. In the following subsections, we show the obtained results.

5.1. Graphical Tests

In this subsection, we discuss the application of the main graphical tests developed in reference [18]. These tests were implemented in Matlab 9.1 and executed on the PC described in Section 1. Most of the graphical tests group the bits in octets (groups of eight bits), except for the linear complexity test, which analyzes the sequence bit after bit, and the chaos game, which considers the bits in groups of two elements. Now, we detail the applied tests.

1. Return map

This test is a measure of the sequence entropy, which would detect any useful information about the parameters used in the design of the sequence generator. The result of this test should be a distribution of points with no trends, no lines, no figures, etc., that is, without any specific pattern. Basically, the return map is a graph of the sequence terms x_t as a function of the previous terms x_{t-1}. Figure 3 shows the return map of a keystream sequence generated using TinyJambu. In fact, it exhibits a cloud of disordered points spread out all over the rectangle without any particular distribution. In brief, it does not provide any information for a possible cryptanalysis, and consequently, from the point of view of this graphical test, the sequence is suitable for cryptographic application.

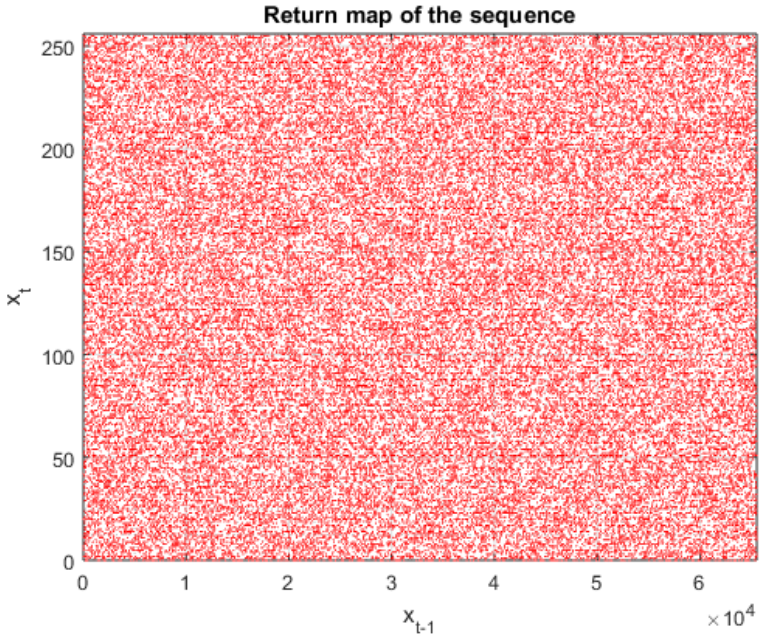


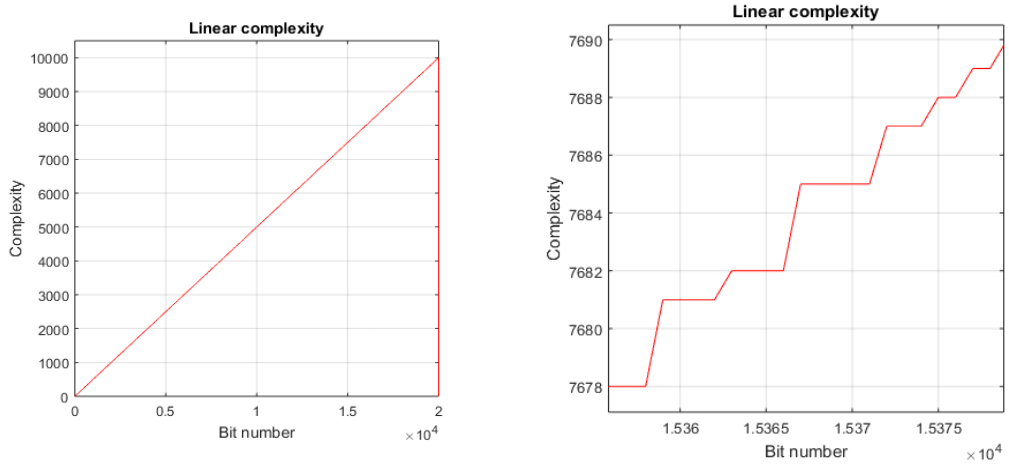
Figure 3. Example of return map for a TinyJambu keystream sequence: a set of disordered points that do not provide any information about the parameters of the generator.

In [18] (Section 5, Figure 3a,b), several examples of return maps obtained from quadratic generators reveal clearly geometric structures and a lack of randomness.

2. Linear Complexity

Traditionally, linear complexity (LC) is a measure of the unpredictability of a sequence, which is a powerful metric for cryptographic sequences. LC is defined as the length of the shortest Linear Feedback Shift Register (LFSR) able to generate such a sequence [23] (pp. 27–29). The algorithm of Berlekamp–Massey [24] computes the value of this parameter. A typical linear complexity profile is a staircase graph that closely follows, but irregularly, the $1/2$ line (that is, the straight line with a slope of $1/2$), exhibiting average step lengths and step heights of values 2 and 4, respectively. More details concerning LC can be found in [25,26].

For cryptographic purposes, the complexity must be as large as possible. In practice, for a sequence of period T, its LC must satisfy $LC \cong T/2$. In Figure 4, we can see (a) the complexity profile for the first 20,000 bits of a TinyJambu sequence and (b) a zoom-in of the linear complexity profile. After the analysis of 20,000 bits, the value of LC should be approximately LC = 10,000, as can be observed in Figure 4a.



(**a**) (**b**)

Figure 4. Linear complexity profile of a TinyJambu keystream sequence. (**a**) Linear complexity profile of the first 20,000 bits as a staircase graph over the 1/2 line; (**b**) A zoom-in of the staircase profile.

3. Shannon Entropy and Min-Entropy

The entropy of a sequence is a measure of the amount of information provided by a process with a result that is the sequence, or equivalently, it is a measure of the uncertainty of a random variable X. Such a measure is expressed in bits; see [20]. Shannon entropy is defined as the average probability of all the values that the random variable can take. More precisely, let X be a random variable that takes the values x_1, x_2, x_3, . . ., x_n. Thus, Shannon entropy is defined as

$$H(X) = -\sum_{i=1}^{n} Pr(x_i) \cdot log_2(Pr(x_i)).$$

If the process is the generation of a sequence of integers modulo m, perfectly random with $m = 2^n$, then its entropy is equal to n. In our case, the entropy of a random sequence must be close to n = 8 bits per octet (byte). The min-entropy is the measure of the probability of the more frequent occurrence value of the random variable. It is a measure recommended by the NIST SP 800–90B standard for True Random Number Generators.

In order to determine whether the TinyJambu generator of keystream sequences is suitable for these values of entropy, we can say that for a sequence of 2^{20} bytes, the Shannon entropy must be $\geq$7.976 bits per byte, while the min-entropy must be approximately 7.91 bits per byte. In the analysis of the TinyJambu sequences, we obtained, on average, the following values:

Shannon entropy (ideal) = 8 bits per octet.
Shannon entropy (minimum accepted) = 7.976 bits per octet.
Shannon entropy measured = 7.9986 bits per octet.
Min-entropy (ideal) = 7.91 bits per octet.
Min-entropy (minimum accepted) = 7.608 bits per octet.
Min-entropy measured = 7.8132 bits per octet.

In view of the numerical values obtained, we can say that the Shannon entropy and min-entropy tests were passed.

4. Samples in increasing order

The TinyJambu sequence is codified in groups of 8 bits. Such octets, considered numerical values in the interval [0, . . ., 255], are sorted in increasing order and represented by means of a graph. If all the 8-bit numbers were generated, then the representation would be a continuous line. At the same time, if the density were uniform (all the numerical values appear with the same frequency), then the slope line (over which the samples are represented) would be 45 degrees. Figure 5 depicts the results of this test. The samples give

rise to a continuous straight line (in red) with a 45-degree slope line, which totally covers the blue line with the same slope used as reference. In view of the results of Figure 5, the samples under the increasing order test satisfactorily passed.

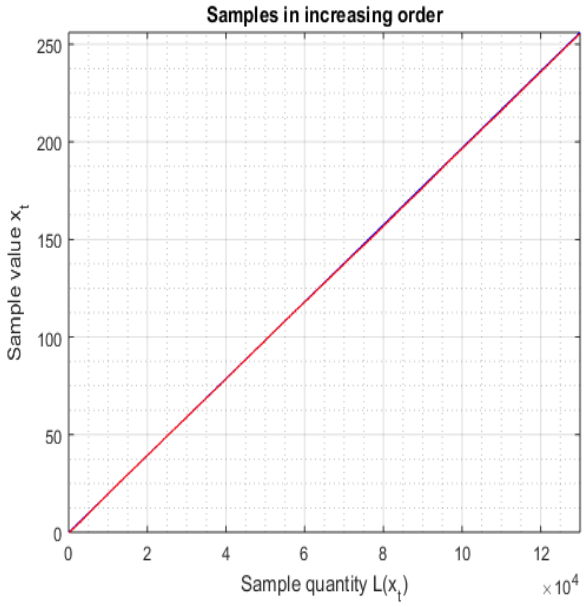


Figure 5. Samples sorted in increasing order: a representation of the 8-bit numerical samples that totally covers the 45-degree line (red line) over the reference line (blue line).

5. Chaos game

The chaos game is a mathematical technique that allows us to convert a one-dimensional sequence such as that generated using TinyJambu into a two-dimensional sequence, giving rise to a more detailed visual representation. From such a representation, we can characterize possible patters in the sequence under consideration. The outcome of a chaos game representation is called an attractor (fractal or compact set); for more details, the interested reader is referred to [18,19]. In fact, this technique can determine non-randomness in the kind of pseudorandom sequences that we are analyzing. In Figure 6, we can just see a cloud of disordered points spread out all over the square. No patterns or fractals are observed. Consequently, the chaos game test was passed.

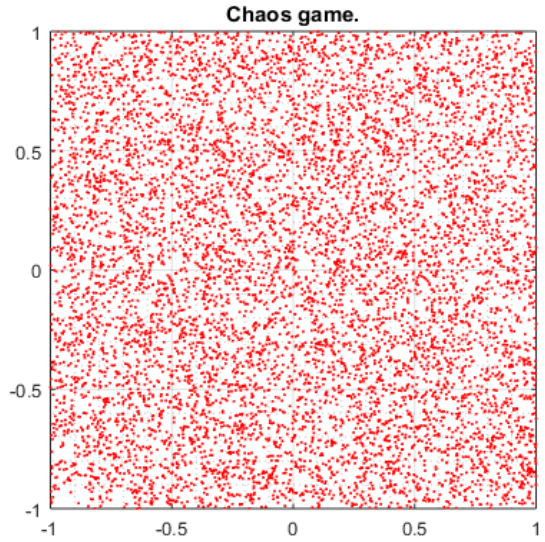


Figure 6. Example of chaos game for a TinyJambu keystream sequence. A cloud of points spread out all over the square without a particular distribution indicates good randomness.

In [18] (Section 5, Figure 7a,b), examples of chaos game representations obtained from quadratic generators show fractal structures that reveal a lack of randomness.

6. Autocorrelation

Autocorrelation (or cross-correlation) is a mathematical technique that looks for repeated samples among the different portions of a sequence when they are compared. This property is very useful for finding periodic or repetitive patterns within a signal or, in this case, within a binary sequence. The sequence is compared with a shifted version of the same sequence. The number of agreements and disagreements among bits for the pair of the original sequence and shifted version are computed. The comparison is carried out for all the possible shifts of the sequence over itself. Figure 7 represents the autocorrelation of a TinyJambu sequence after analyzing 120,000 bits. The first autocorrelation coefficient is always equal to 1, as the sequence coincides with itself (autocorrelation in phase); the remaining coefficients corresponding to the successive shifts must be as small as possible (autocorrelation out of phase). In Figure 7, it is clear that the obtained values in red are close to 0. This means that, as far as this metric is concerned, the analyzed sequence exhibits good randomness characteristics, and consequently, the autocorrelation test was passed.

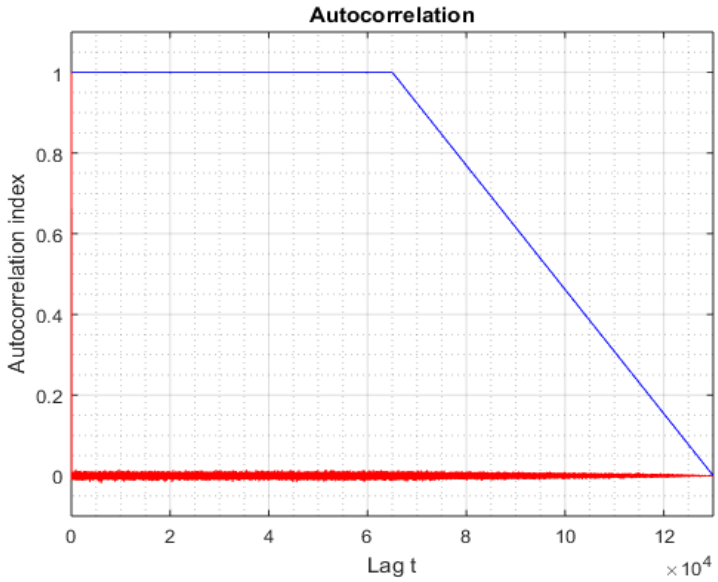


Figure 7. Example of autocorrelation for a TinyJambu keystream sequence with values of autocorrelation in red out of phase and close to 0. The blue line depicts the decrement of autocorrelation values related to mutual sequence shift.

7. Fast Fourier Transform

A Fast Fourier Transform (FFT) efficiently computes a Discrete Fourier Transform or transformation of a signal into the frequency domain. In the analysis of binary sequences, a FFT may detect repetitive patterns in the sequence under study. In fact, a random sequence must exhibit all the FFT harmonics with approximately the same amplitude, and no up/down trends should appear. Figure 8 shows the result of this test applied to a TinyJambu sequence of 120,000 bits. It can be seen that all the amplitudes are in the same range; consequently, the test was passed.

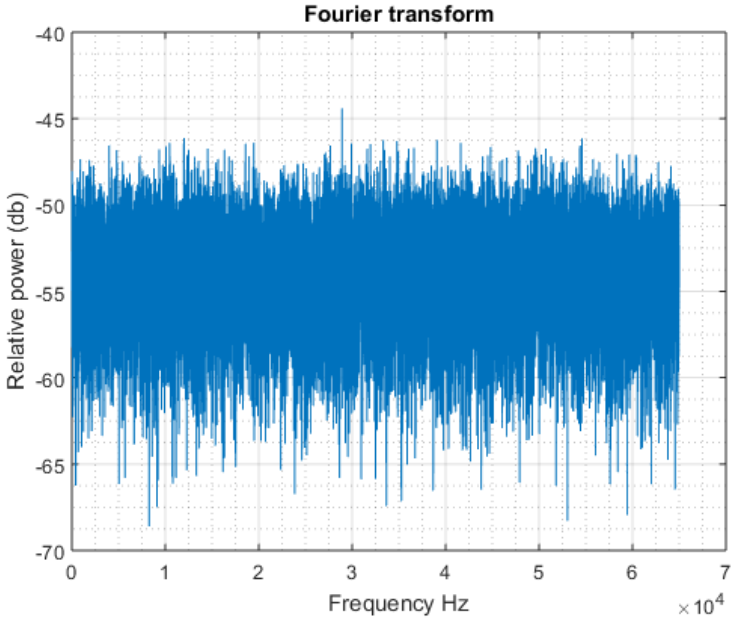


Figure 8. Fast Fourier Transform (FFT) representation for a TinyJambu sequence of 120,000 bits, where all the harmonics exhibit approximately the same amplitude.

5.2. The Diehard Battery of Pseudorandomness Tests

Random number generators only exist in nature. In practice, we have to settle with pseudorandom number generators, which are tested, and their results should not exhibit neither evidence of regularity nor evidence of a statistical distribution. At this point, we applied the Diehard battery of tests, which consists of 15 independent tests, although some of them are repeated but with distinct parameter values. Diehard tests use the chi-square goodness-to-fit technique for computing uniform p-values in the interval [0, 1); see [20,21].

In order to apply the previous tests to the TinyJambu sequences, we created a series of sequences, all of them generated using the same generator but with different choices of key, nonces, AD or plain text.

From the Diehard battery, we applied the following tests:

1. Binary Rank Test for 6 × 8 matrices;
2. 3DSPHERES Test;
3. Overlapping 5-permutation Test;
4. SQEEZE Test;
5. Parking lot Test;
6. Binary Rank Test for 31 × 31 matrices;
7. Binary Rank Test for 32 × 32 matrices;
8. Birthday spacing Test;
9. Overlapping sums Test;
10. Runs Test;
11. Craps Test;
12. Minimum distance Test.

For the sake of simplicity, we considered the previous tests among the 15 proposed in the Diehard battery. Since all of these are statistical tests, it is more accurate to determine whether their results (the p-values) fall inside a determined upper and lower bound. In fact, the p-values may fluctuate but not trespass these bounds, indicating good statistical properties or good pseudorandomness characteristics. In this work, we established our boundaries in the following margins:

$$0.03 < p\text{-values} < 0.97$$

Sequences with *p*-values greater or lower than these bounds were said to exhibit poor pseudorandomness [27,28]. At this point, we applied the Diehard battery of tests to a series of sequences generated using TinyJambu.

The following results are for tests 1–4, and they are depicted in Figure 9:

1. Binary Rank Test for 6 × 8 matrices: sequence_5 and sequence_10 showed a slightly higher *p*-value than the established upper bound.
2. 3DSPHERES Test: only sequence_11 exhibited a *p*-value lower than the bound, which means a slightly deficient pseudorandomness.
3. Overlapping 5-permutation Test: here, the *p*-values were too low for sequences 1 and 2, showing poor pseudorandomness characteristics.
4. SQEEZE TEST: all sequences passed this test satisfactorily.

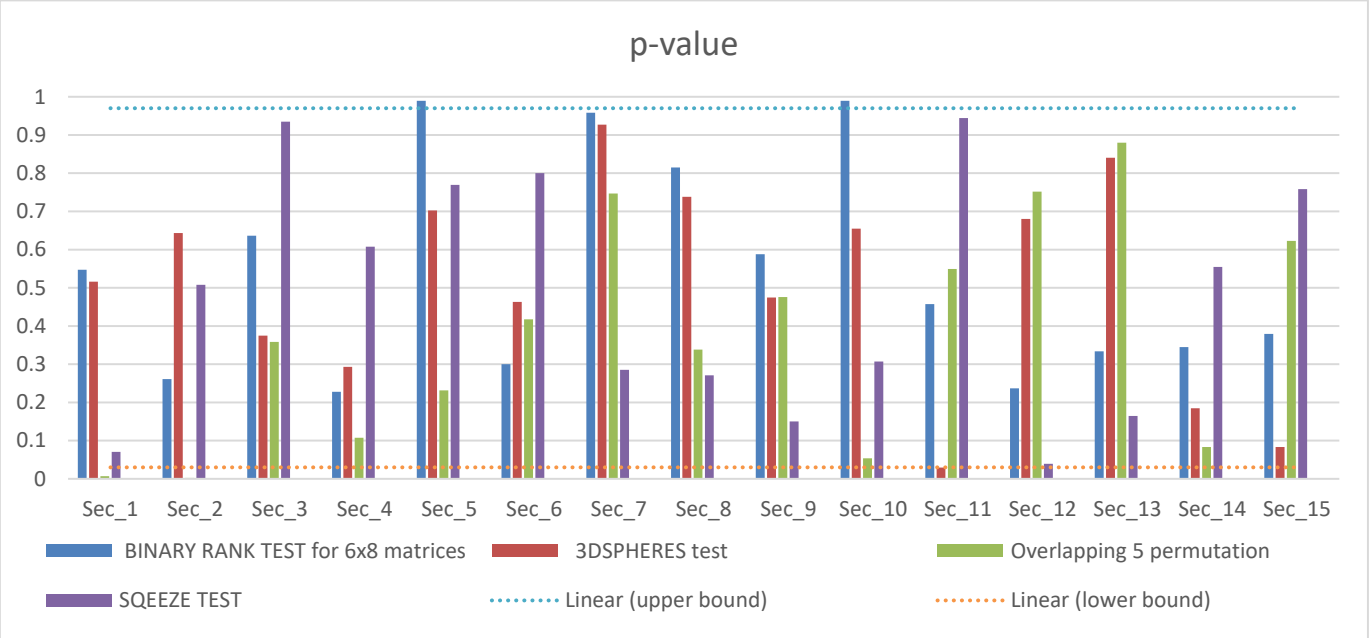


Figure 9. This figure shows the results of the first four DIEHARD tests applied to sequences_1 up to sequence_15.

Next, the following results are for tests 5–8, and they are depicted in Figure 10:

5. Parking lot Test: sequence_6 slightly surpassed the upper bound, which indicates a slight deficiency in pseudorandomness.
6. Binary Rank Test for 31 × 31 matrices: all sequences showed good pseudorandomness characteristics.
7. Binary Rank Test for 32 × 32 matrices: all sequences showed good pseudorandomness characteristics.
8. Birthday spacing Test: all sequences exhibited good pseudorandom characteristics, although sequence_6 was near the bound.

For tests 9–12, Figure 11 shows the results, and the results are as follows:

9. Overlapping sums Test: all sequences exhibited *p*-values within the boundaries.
10 Runs Test: all sequences showed good pseudorandomness characteristics.
11. Craps Test: all sequences showed good pseudorandomness characteristics.
12. Minimum distance Test: all sequences showed good pseudorandomness characteristics, except for sequence_7, which showed a poorer pseudorandomness.

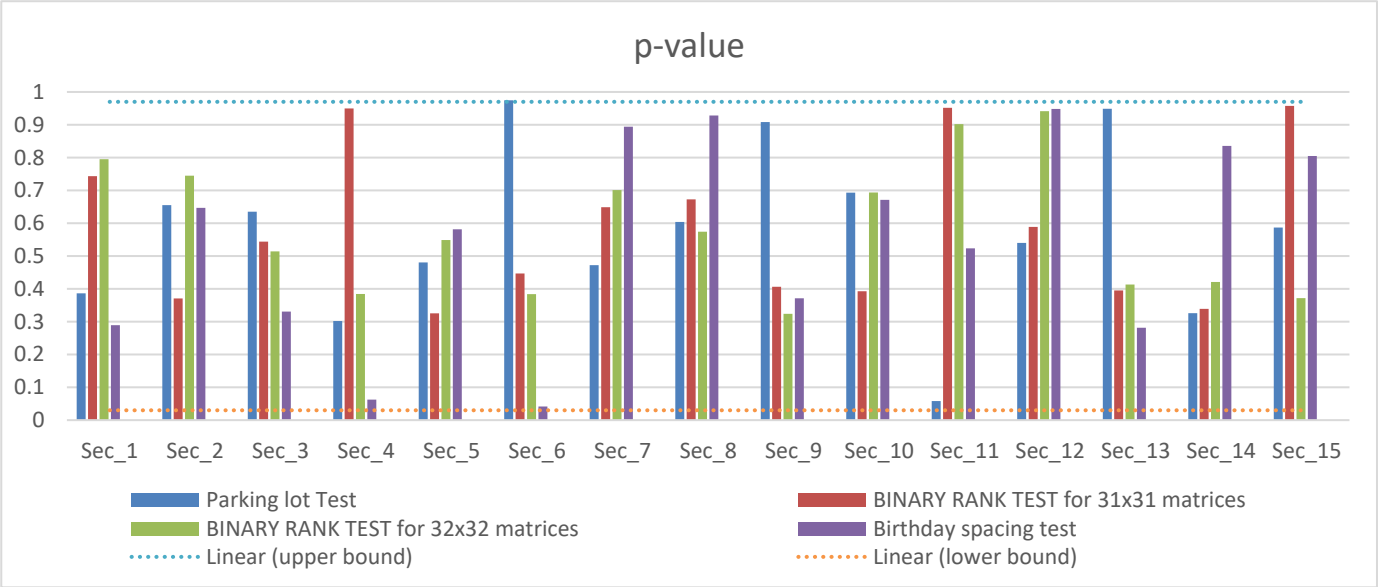


Figure 10. This figure shows the results of another four DIEHARD tests applied to sequence_1 to sequence_15.

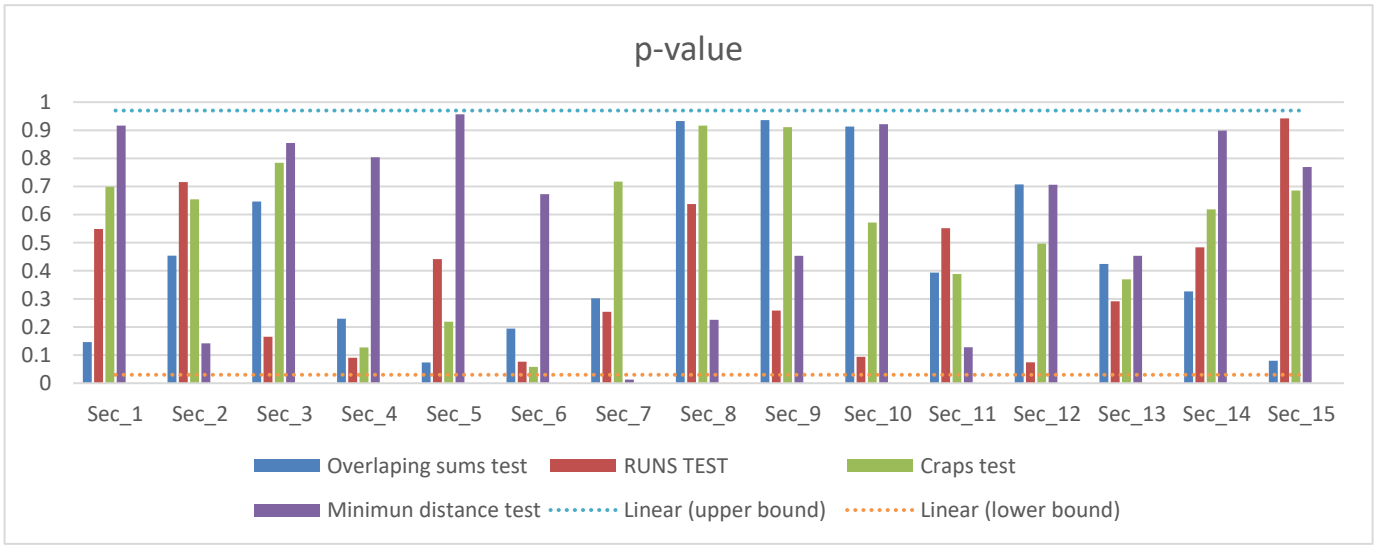


Figure 11. This figure shows the results of the last four DIEHARD tests applied to sequence_1 to sequence_15.

As we have seen from the above results, in general, the TinyJambu sequences pass the tests. Some sequences exhibit *p*-values beyond the boundaries for any test. Nevertheless, we must remember that we are applying statistical tests to detect regularities, and we could not find any at this point. Based on the results, we can say that the sequences generated using the cryptosystem TinyJambu show good pseudorandomness.

5.3. FIPS Test 140-2. Requirements of Security for Cryptographic Modules

FIPS (Federal Information Processing Standard) Publication 140-2 is a U.S. government security standard [22] used to evaluate cryptographic modules. FIPS 140-2 is made up of four statistical tests (e.g., the Monobit Test, the Poker Test, the Runs Test and the Long Runs Test). The proposed TinyJambu keystream sequences passed all the FIPS tests. The average results we obtained are as follows:

1. LONG RUNS TEST: passed. No runs of more than 25 equal and consecutive bits appeared.
2. MONOBIT TEST: passed. The test is passed if 9752 < number of ones < 10,275. The obtained result was 9941.
3. POKER TEST: passed. The test is passed if 2.16 < X < 46.17. The obtained result was 12.8192.
4. RUNS TEST: passed. The test is passed if the runs (runs of zeros in red and runs of ones in blue) of length 1–6 are within the specified interval in green in Figure 12. Notice that runs of a short length are more frequent than runs of a long length. For short runs, the number of runs of zeros (gaps) and runs of ones (blocks) must be equal (the second postulate of pseudorandomness of Golomb [23]).

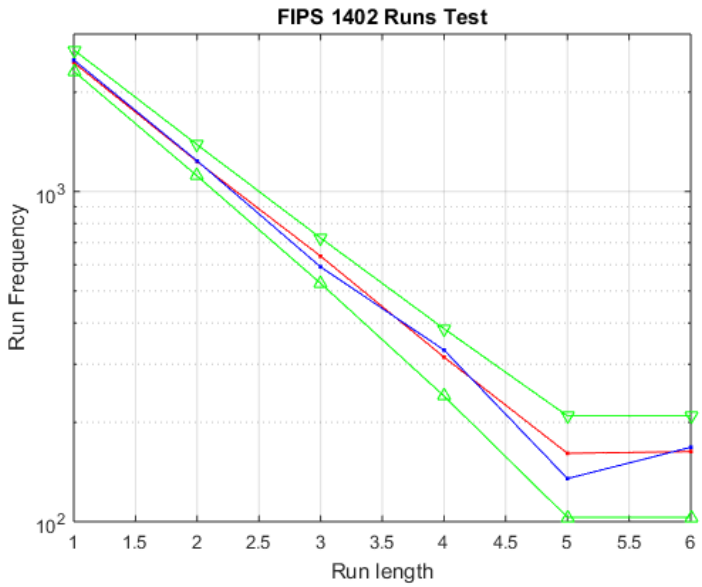


Figure 12. Example of Runs Test results for a TinyJambu keystream sequence. Notice that the test is passed by runs of zeros (red line) as well as by runs of ones (blue line) as both fall into the zone delimited by the green lines that bound the permission zone.

6. Conclusions

In this work, the security of the symmetric cryptosystem TinyJambu was analyzed from two distinct and non-symmetric points of view.

(1) We improved the best cryptanalytical attack against this cryptosystem found in the literature. It deals with a differential forgery attack against the nonce introduction. In fact, by making use of a refined model developed by Saha et al. in [8], we obtained a maximum number of different trails with a minimum number of uncorrelated *AND* gates. This search, which is much more exhaustive than previous ones, allows us to compute a differential probability of value $p = 2^{-65.9487}$, which is improved considerably compared to previous numerical results, e.g., the $p = 2^{-70.68}$ given by Saha et al. and $p = 2^{-80}$ given by the TinyJambu designers. The new probability increments, in several orders of magnitude, the success probability of a forgery attack and approximates the ideal success probability of value $p = 2^{-64}$. As a future work, we intend to execute the Python programs used in this study in a more powerful computational environment to obtain a differential probability nearer to the bound specified for a successful attack. Moreover, the strategy sketch used to obtain the differential trails when the number of rounds increases was also presented.

(2) In addition, we analyzed the degree of pseudorandomness of the sequences generated using TinyJambu. Traditionally, the cryptanalytic techniques employed against this cryptosystem were conducted under block cipher cryptanalysis. Nevertheless, in this work, TinyJambu was considered a stream cipher with keystream sequences obtained through the

concatenation of the bits used in the encryption/decryption process. Once the sequences were constructed symmetrically in emission and reception, we could study their degree of pseudorandomness. This is an original vision realized to evaluate the strength of the TinyJambu algorithm, which enlarges the possibilities of a security assessment. In this way, a complete and powerful battery of statistical tests was applied to these keystream sequences, which satisfactorily passed the empirical tests.

This double study enhances the weaknesses of TinyJambu against differential attacks as well as the strength of TinyJambu against statistical analyses. Two different aspects were analyzed, and two non-symmetric conclusions may be drawn from the outcomes of this work.

The results obtained here can be extrapolated to the updated version proposed by the designers that, in the nonce introduction, increases the keyed permutation to 640 rounds. In fact, the relationship between the number of rounds and the minimum number of uncorrelated *AND* gates and the optimization of the updated version are our priorities for future works.

Author Contributions: All the authors have equally contributed to the reported research in terms of conceptualization, methodology, software and manuscript revision. All authors have read and agreed to the published version of the manuscript.

Funding: This work is part of the R+D+i grant P2QProMeTe (PID2020-112586RB-I00), funded by MCIU/AEI/10.13039/501100011033. The authors are also supported by the University of Málaga (Spain) through "Red temática BIOMED-SEC", reference D5-2022-04.

Data Availability Statement: Our results are already depicted in the article. The links are given in the references.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. National Institute of Standards and Technology. Lightweight Cryptography (LWC) Standardization Project. 2019. Available online: https://csrc.nist.gov/projects/lightweight-cryptography (accessed on 13 February 2024).
2. NIST Lightweight Cryptography Finalists. Available online: https://csrc.nist.gov/Projects/lightweight-cryptography/finalists (accessed on 13 February 2024).
3. Paar, C.; Pelzl, J. *Understanding Cryptography*; Springer: Berlin/Heidelberg, Germany, 2010.
4. Renner, R. Security in quantum cryptography. *Rev. Mod. Phys.* **2022**, *94*, 025008. [CrossRef]
5. Yin, H.L.; Fu, Y.; Li, C.L.; Weng, C.X.; Li, B.H.; Gu, J.; Chen, Z.B. Experimental quantum secure network with digital signatures and encryption. *Natl. Sci. Rev.* **2023**, *10*, nwac228. [CrossRef] [PubMed]
6. Wu, H.; Huang, T. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms. The NIST Lightweight Cryptography (LWC) Standardization Project. 2020. Available online: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/TinyJAMBU-spec-round2.pdf (accessed on 13 February 2024).
7. Saha, D.; Sasaki, Y.; Danping, S.; Sibleyras, F.; Sun, S.; Zhang, Y. On the Security Margin of TinyJAMBU with Refined Differential and Linear Cryptanalysis. *IACR Trans. Symmetric Cryptol.* **2020**, *3*, 152–174. [CrossRef]
8. Saha, D.; Sasaki, Y.; Danping, S.; Sibleyras, F.; Sun, S.; Zhang, Y. The MILP code corresponding to the paper: On the Security Margin of TinyJAMBU with Refined Differential and Linear Cryptanalysis. *IACR Trans. Symmetric Cryptol.* **2020**, *3*, 152–174. Available online: https://github.com/c-i-p-h-e-r/refinedTrailsTinyJambu (accessed on 13 February 2024). [CrossRef]
9. Gurobi Optimizer. Available online: https://www.gurobi.com/academia/academic-program-and-licenses/ (accessed on 13 February 2024).
10. Wu, H.; Huang, T. TinyJAMBU Update, Update to the NIST Lightweight Cryptography Standardization Process. 2020. Available online: https://csrc.nist.gov/csrc/media/Projects/lightweight-cryptography/documents/finalist-round/status-updates/tinyjambu-update.pdf (accessed on 13 February 2024).
11. Teng, W.; Salam, I.; Yau, W.C.; Pieprzyk, J.; Phan, R.C. Cube attacks on round-reduced TiniJAMBU. *Sci. Rep.* **2022**, *12*, 5317. [CrossRef]
12. Naito, Y.; Matsui, M.; Sugawara, T.; Suzuki, D. SAEB: A Lightweight Block Cipher-Based AEAD Mode of Operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 192–217. [CrossRef]
13. NIST: Submission Requirements and Evaluation Criteria for the Lightweight. Cryptography Standardization Process 2018. Available online: https://csrc.nist.gov/Projects/ (accessed on 13 February 2024).

14. Mouha, N.; Wang, Q.; Gu, D.; Preneel, B. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Information Security and Cryptology, Proceedings of the Inscrypt 2011, Beijing, China, 3 December 2011*; Wu, C.K., Yung, M., Lin, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7537, pp. 57–76.
15. Sun, S.; Hu, L.; Wang, M.; Wang, P.; Qiao, K.; Ma, X.; Shi, D.; Song, L.; Fu, K. Constructing mixed-integer programming models whose feasible region is exactly the set of all valid differential characteristics of SIMON. *Cryptol. ePrint Arch.* **2015**. Available online: https://eprint.iacr.org/2015/122 (accessed on 13 February 2024).
16. Jana, A.; Rahman, M.; Saha, D. DEEPAND: In-Depth Modeling of Correlated AND Gates for NLFSR-based Lightweight Block Ciphers. *Cryptol. ePrint Arch.* **2022**. Paper 2022/1123. Available online: https://eprint.iacr.org/2022/1123 (accessed on 13 February 2024).
17. Fuster-Sabater, A.; Pazo-Robles, M.E. Reducing the Security Margin Against a Differential Attack in the TinyJambu Cryptosystem. In Proceedings of the 16th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2023), Salamanca, Spain, 5 September 2023. [CrossRef]
18. Cardell, S.D.; Requena, V.; Fuster-Sabater, A.; Orúe, A.B. Randomness Analysis for the Generalized Self-Shrinking Sequences. *Symmetry* **2019**, *11*, 1460. [CrossRef]
19. Orue Lopez, A.B. Contribution to Study of Cryptanalysis and Design of Chaotic Cryptosystems. Ph.D. Thesis, Polytechnical University of Madrid (UPM), Madrid, Spain, 2013.
20. Marsaglia, G. *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*; Florida State University: Tallahassee, FL, USA, 1995; Available online: http://www.stat.fsu.edu/pub/diehard (accessed on 13 February 2024).
21. Almaraz Luengo, E.; Roman Villaizan, J. Cryptographically Secured Pseudo-Random Number Generators: Analysis and Testing with NIST Statistical Test Suite. *Mathematics* **2023**, *11*, 4812. [CrossRef]
22. Evans, D.L.; Bond, P.; Bement, A. FIPS PUB 140-2. Security Requirements for Cryptographic Modules. In *Federal Information Processing Standards Publication 140-2*; U.S. Department of Commerce, NIST National Technical Information Service: Springfield, VA, USA, 2001.
23. Golomb, S.W. *Shift Register-Sequences*; Aegean Park Press: Laguna Hill, CA, USA, 1982.
24. Massey, J.L. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* **1969**, *15*, 122–127. Available online: https://ieeexplore.ieee.org/document/1054260 (accessed on 13 February 2024). [CrossRef]
25. Rueppel, R.A. *Analysis and Design of Stream Ciphers*; Springer: Berlin/Heidelberg, Germany, 1986.
26. Fuster-Sabater, A.; Requena, V.; Cardell, S.D. An efficient algorithm to compute the linear complexity of binary sequences. *Mathematics* **2022**, *10*, 794. [CrossRef]
27. Almaraz Luengo, E. A brief and understandable guide to pseudo-random number generators and specific models for security. *Stat. Surv.* **2022**, *16*, 137–181. [CrossRef]
28. Fen, Y.; Hao, L. Testing Randomness Using Artificial Neural Network. *IEEE Access* **2020**, *8*, 163685–163693.