Practical Related-Key Forgery Attacks on the Full TinyJAMBU-192/256

Orr Dunkelman, Eran Lambooij*, Shibam Ghosh

Department of Computer Science, University Of Haifa, Haifa, Israel
*eran@hideinplainsight.io

Abstract. TinyJAMBU is one of the finalists in the NIST lightweight cryptography competition. It has undergone extensive analysis in the recent years as both the keyed permutation as well as the mode are new designs. In this paper we present a related-key forgery attack on the updated TinyJAMBU scheme with 256- and 192-bit keys. We introduce a high probability related-key differential attack were the differences are only introduced into the key state. Therefore, the characteristic is applicable to the TinyJAMBU mode and can be used to mount a forgery attack. The time and data complexity of the forgery are 2^{32} using 2^{10} related-keys for the 256-bit key version, and 2^{42} using 2^{12} related-keys for the 192-bit key version.
For the 128-bit key we construct a related-key differential characteristic on the full keyed permutation of TinyJAMBU with a probability of 2^{-16}. We extend the related-key differential characteristics on TinyJAMBU to practical time key recovery attacks that extract the full key from the keyed permutation with a time and data complexity of 2^{23}, 2^{20}, and 2^{18} for respectively the 128-, 192-, and 256-bit key variants.
All characteristics are experimentally verified and we provide key nonce pairs that produce the same tag to show the feasibility of the forgery attack.

1 Introduction

In recent years it has been understood that, although confidentiality is an important feature of cryptographic primitives, authenticity of the data is, an often overlooked, but equally valuable property. Therefore, when NIST launched the lightweight cryptography competition [5] to find a new lightweight primitive. They included authenticity as one of the primary design goals.

When looking at authenticated encryption, the scheme is considered to be broken when either the confidentiality or the authenticity of the ciphertext are diminished. This both improves and worsens the job of a cryptanalyst, as now, for a full break of the scheme two things must happen in parallel: First the cryptographer has to find a favorable property in the primitive; Second, it must be possible to observe this property through the (often restrictive) mode.

In this paper we look at the authenticated encryption scheme TinyJAMBU [6,7], which is based on the sponge duplex construction [1,2] and a lightweight keyed

permutation. The keyed permutation used in TinyJAMBU has a 128-bit state and a 128-, 192-, or 256-bit key. The permutation is an NLFSR, with a NAND gate as the non-linear component. Each round the feedback bit is XOR-ed with the next key bit using a cyclic key schedule.

In the TinyJAMBU mode two versions of the keyed permutation are used. We denote these as: $\mathcal{P}^a$, and $\mathcal{P}^b$. $\mathcal{P}^a$ is used in phases where no output is observed, and consists of 640 rounds for all the key sizes. $\mathcal{P}^b$ is used in the first initialization step and when part of the state can be observed. It consists of 1024, 1152, and 1280 rounds for respectively the 128-, 192-, and 256-bit key variants. We note that the designers of TinyJAMBU have recently changed the number of rounds of $\mathcal{P}^a$ from 384 to 640 [7] to counter a differential attack by Saha *et al.* [3]. Therefore, most of the results in the literature are on 384 rounds.

We first look at the current results on TinyJAMBU before we turn our attention to our contributions. In their specification of TinyJAMBU [6,7] the designers provide a rigorous security analysis of the design against various attacks. Using MILP modelling they prove the probability of the best differential for 384 rounds of the permutation to be less then 2^{-78}. In [3], Saha *et al.* look at the differential characteristics through the TinyJAMBU permutation improving the probability of the characteristics using (first order) correlated NAND gates. The authors propose a differential characteristic through 338 and 384 rounds of the permutation with a probability of respectively $2^{-62.68}$ and $2^{-70.68}$.

Most attacks consider the case where the attacker only has access to 32 in- and output bits as is dictated by the mode. When there are no constraints on the bits that the attacker can access; Saha *et al.* report a characteristic through 384 rounds of the keyed permutation with probability 2^{-19} [3]. To mitigate these attacks, the designers of TinyJAMBU increased the number of rounds from 384 to 640 in the second version [7] of the specification.

In the updated specification the designers of TinyJAMBU improved the differential characteristics through the keyed permutation in the case that the attacker can only affect 32 of the in- and output bits. They found differential characteristics with probabilities 2^{-41}, 2^{-64}, and 2^{-88} covering respectively 384, 512, and 640 rounds.

Due to the cyclic nature of the permutation slide attacks are a natural direction of analysis. Sibleyras, *et al.* [4], discuss full round slide attacks on TinyJAMBU in the single key setting. The designers of TinyJAMBU already mention in the specification that there exists a simple related-key slide attack. Due to the frame bits used in the mode the slide attacks cannot be used to attack the full scheme. The complexities of these attacks are beyond the birthday bound.

1.1 Our Contributions

We propose an iterative related-key differential characteristic that can be applied to the keyed permutation inside the mode. The attack works for the 256-bit as well as the 192-bit key variants of the permutation. Combining this characteristic with the proper nonce differences we can reach a zero difference in the state just before the first message addition with a probability of 2^{-32} and 2^{-42} for

respectively the 256- and 192-bit key variants. We note that the designers of TinyJAMBU do not claim security in the related key model.

We extend this attack to a nonce respecting forgery attack which has a data and time complexity of 2^{32} and 2^{42} for respectively the 256- and 192-bit variants of TinyJAMBU. The results are summarised in Table 1 and we provide inputs that lead to a forgery for both versions in Table 3.

The 128-bit key variant of TinyJAMBU cannot be attacked using the same characteristic. But, we show that if we allow for differences to be inserted into state bits, we can get a similar iterative characteristic for the 128-bit variant with a probability of 2^{-2} per 128 rounds.

We note that our results are also applicable to the first version to the TinyJAMBU scheme.

Table 1: A summary of distinguishers on TinyJAMBU. The results only include distinguishers where the attacker has access to 32 in-, and output bits.

Key size	#Rounds	Complexity (Data)	Setting	Type	Source
128	1024	2^{16}	Related-key CP	Differential	Section 3.3
192	1152	2^{12}	Related-key CP	Differential	Section 3.3
256	1280	2^{10}	Related-key CP	Differential	Section 3.3
any	338	$2^{62.68}$	CP	Differential	[3]
any	384	$2^{70.68}$	CP	Differential	[3]
any	384	2^{41}	CP	Differential	[7]
any	512	2^{64}	CP	Differential	[7]
any	512	2^{64}	KP	Linear	[7]
any	640	2^{88}	CP	Differential	[7]
128	∞	2^{64}	KP	Slide	[4]
192	∞	2^{65}	Adaptive CP	Slide	[4]
256	∞	$2^{67.5}$	Adaptive CP	Slide	[4]

CP = Chosen Plaintext
KP = Known Plaintext

1.2 Paper Structure

We give a short overview of the mode and the keyed permutation used in TinyJAMBU in Section 2. Next we describe the related key differential characteristic and the resulting forgery on the full mode in Section 3 and Section 4. We conclude the paper in Section 5.

2 The Specification of TinyJAMBU

TinyJAMBU is one of the finalists of the NIST lightweight competition. The design principle of TinyJAMBU is based on the sponge duplex mode using a keyed permutation. The keyed permutation is derived from a NLFSR with a 128-bit state using a NAND gate as the non-linear operation the key bits are added in a cyclic fashion.

In the upcoming sections we give a quick overview of the important parts of TinyJAMBU with respect to understanding the attack. For a complete specification of TinyJAMBU we refer the reader to the full specification of TinyJAMBU [7].

2.1 The keyed permutation

The TinyJAMBU keyed permutation used has a 128-bit state and is defined for 128-, 192-, or 256-bit keys. Given an n-bit register $x \in \{0,1\}^n$ we define x_i to denote the i-th bit of the register. Here x_0 is the least significant bit, and x_{n-1} is the most significant bit of the register. Using this notation we can write the n-bit register x as $(x_{n-1}, x_{n-2}, \ldots, x_0)$. The size of a register x is denoted as $|x|$ and the concatenation of two registers x, y is denoted as $x\|y$.

We define the permutation $\mathcal{P}$ on the state $v \in \{0,1\}^{128}$, given a key $k \in \{0,1\}^{\kappa}$ (where $\kappa \in [128, 192, 256]$), and round i as:

$$\mathcal{P}_{(k,i)}(v) = (v_{91} \oplus \overline{v_{85} v_{70}} \oplus v_{47} \oplus v_0 \oplus k_{(i \mod |k|)}, v_{127}, \ldots, v_1)$$

Now we define the r-round permutation $\mathcal{P}$ for some key k as:

$$\mathcal{P}^r = \mathcal{P}_{(k,r-1)} \circ \mathcal{P}_{(k,r-2)} \circ \ldots \circ \mathcal{P}_{(k,0)}$$

The round function $\mathcal{P}$ of the TinyJAMBU permutation is depicted in Figure 1.

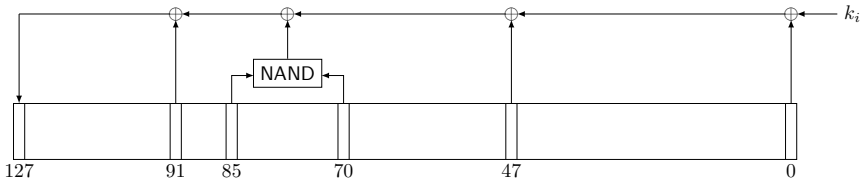

Fig. 1: Round function of the TinyJAMBU permutation.

2.2 The TinyJAMBU mode

The TinyJAMBU mode uses the permutation described in Section 2.1 in a duplex construction with a 32-bit message injection part, a 32-bit squeezing part. The mode consists of four separate phases: the key initialization, the associated data,

the encryption, and the finalization part. After every permutation call a constant depending on the current phase const_ℓ is added to the state. One thing to note is that unlike most duplex constructions the squeezing and injection of data occurs in different parts of the state.

TinyJAMBU uses keyed permutations using the same key k, but a different number of rounds in different phases of the encryption. We denote them as $\mathcal{P}^a$ and $\mathcal{P}^b$ where the values of a and b for the different key sizes are given in Table 2.

Table 2: Parameters for the different variants of TinyJAMBU.

Variant	#Rounds $\mathcal{P}^a$	#Rounds $\mathcal{P}^b$	State	Key	Nonce	Tag
TinyJAMBU-128	640	1024	128	128	96	64
TinyJAMBU-192	640	1152	128	192	96	64
TinyJAMBU-256	640	1280	128	256	96	64

Initialization. In the initialization the key k is mixed into the state $s \in \{0,1\}^{128}$ by applying $\mathcal{P}^b$ to the initial state $s = (0, 0, ..., 0)$. After that, in the nonce setup phase, a 96-bits nonce N is split up into three 32-bit nonce parts $N_0 \| N_1 \| N_2$ and for each part of the nonce the state is updated with $\mathcal{P}^a$ after which the nonce is added to the most significant bits of the state. A depiction of the initialization is given in Figure 2.

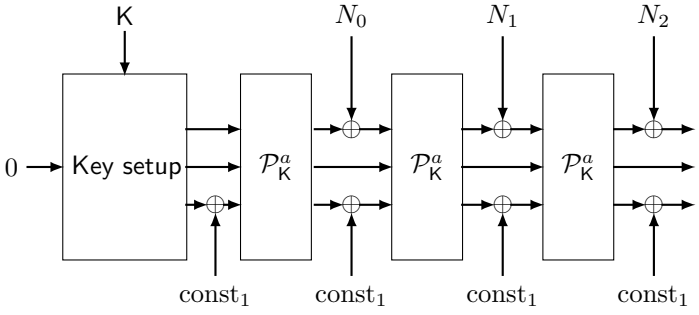

Fig. 2: TinyJAMBU initialization and nonce addition.

Associated Data Processing. During the Associated data processing the associated data is added to the state. The associated data is divided into 32-bit

blocks. For each block the state is updated with $\mathcal{P}^a$, after which the associated data block is XORed into the state. We depict the associated data processing in Figure 3.

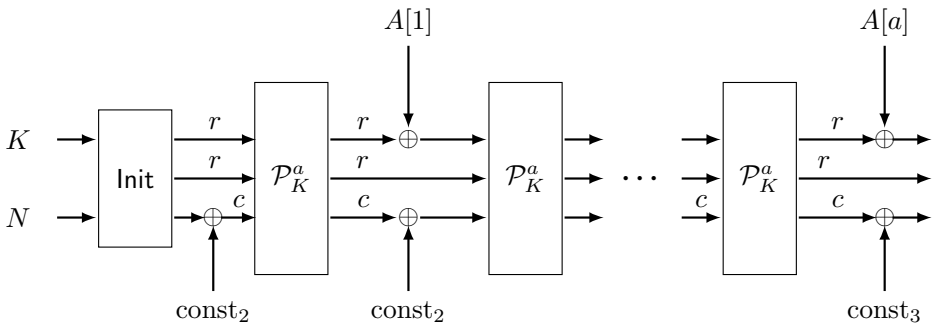

Fig. 3: Processing the associated data.

Encryption. During the encryption stage a key stream is generated to encrypt a message into a ciphertext. The plaintext is divided into 32-bit blocks. For each block, the state is updated with $\mathcal{P}^b$, after which the plaintext block is XORed into the most significant part of the state. Finally, we obtain the 32-bit ciphertext block by XORing bits $95 \dots 64$ of the state with the plaintext block. Note that, the plaintext and nonce are added to the 32 most significant bits of the state which are $127 \dots 96$. The key stream used for encryption, is obtained from bits $95 \dots 64$.

Finalization. After encrypting the plaintext the 64-bit authentication tag $T_0 \| T_1$ is generated in two steps. First, to generate T_0, we apply $\mathcal{P}^b$ and extract bits $95 \dots 64$. Then we apply $\mathcal{P}^a$ after which we extract the same 32 bits of the state to get T_1. We depict the finalization and encryption process in Figure 4.

3 Related Key Differential Characteristics on $\mathcal{P}$

We describe a high probability related-key differential characteristic on the full-round permutation $\mathcal{P}^{1024}$ with 256- and 192-bit keys. This characteristic uses the simplicity of the key schedule of the keyed permutation to introduce and cancel differences in the state. Moreover, at all times there is at most one active bit in the state, which leads to a high probability characteristic through the full permutation $\mathcal{P}^{1280}$ with a probability of 2^{-10} for the 256-bit key variant. For the 192-bit key variant the characteristic has a probability of 2^{-12} through $\mathcal{P}^{1152}$. The probability for the characteristic through $\mathcal{P}^{640}$ is only 2^{-4} for the 256-bit

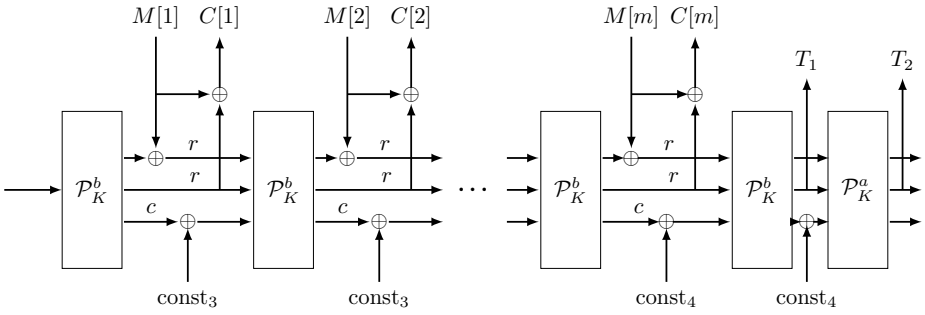

Fig. 4: TinyJAMBU encryption and finalization.

variant and 2^{-6} for th 192-bit variant. These characteristics allow an attacker to reach a zero difference state just before the message addition with a probability of 2^{-32} for the 256-bit key case and 2^{-42} for the 192-bit key case.

3.1 The 256-bit key variant

We start with a difference in key bit k_0 which is inserted into the state in the first round. The propagation of the difference through the linear taps is cancelled by the following key differences: k_{37}, k_{81}, k_{128}. After 128 rounds the difference is cancelled and the state difference is 0 for the next 127 rounds with probability $(\frac{1}{2})^2$. This is because when the introduced state difference reaches a NAND gate it produces a zero difference with probability $\frac{1}{2}$ and the difference enters the NAND gate twice before getting cancelled. At round 256 we return to the original configuration of state and key differences.

The above key difference gives a zero state difference after $\mathcal{P}^{1280}$ with probability 2^{-10}. However, if we use the same key differences for $\mathcal{P}^{640}$ there is a difference in the least significant bit of the state. The problem here is that we cannot cancel this difference using the nonce, since the nonce is added to the most significant bits of the state. The solution to this problem is to shift the key difference by 127 bits. Which leads to the following sets of possible differences in key bits:

$$(k_{164-t}, k_{208-t}, k_{255-t}) \quad \text{for } 0 \leq t < 32$$

and a difference in the following nonce bits (where the nonce is denoted by N):

$$(N_{95-t}, N_{63-t}, N_{31-t})$$

This puts this one bit difference in the output in the most significant bit after $\mathcal{P}^{640}$, while still keeping the zero difference after $\mathcal{P}^{1280}$. One added benefit is that for $\mathcal{P}^{640}$ the characteristic passes through two less non-linear taps, reducing the probability of the characteristic from 2^{-6} to 2^{-4}.

Due to this characteristic we get, with probability $2^{-10-3\cdot4-10} = 2^{-32}$, a zero difference before the message addition. This is important since this is the

first point in which we can observe (part of) the state difference through the ciphertext.

3.2 The 192-bit key variant

The characteristic for the 192-bit variant is nearly the same as the characteristic for the 256-bit variant. The only difference is that instead of offsetting the characteristic by 127 positions to get the 1 difference after $\mathcal{P}^{640}$ in the most significant bit. We offset the difference by 63, which produces the same behaviour. This gives us the following sets of possible differences in the key bits:

$$(k_{100-t}, k_{144-t}, k_{191-t}) \quad \text{for } 0 \leq t < 32$$

and a difference in the following nonce bits (where the nonce is denoted by N):

$$(N_{95-t}, N_{63-t}, N_{31-t})$$

The main difference with the 256-bit variant is that the probability is slightly lower. This is due to the fact that instead of 2 non-linear taps per 256 rounds of the cipher we get 2 non-linear taps per 192 rounds of the cipher. This characteristic has a probability of 2^{-6} and 2^{-12} for respectively $\mathcal{P}^{640}$, and $\mathcal{P}^{1152}$. Producing a 1 bit in the most significant bit of the state after $\mathcal{P}^{640}$ and a zero difference after $\mathcal{P}^{1152}$.

3.3 Other Related Key Characteristics on $\mathcal{P}^{1024}$

The characteristics in Section 3 do not work for the 128-bit key variant as it requires cancelling the state difference with the key difference after 129 rounds. However, if we allow the attacker to insert a difference into the state as well as the key. Or in other words, if we analyse the permutation as a stand alone primitive, we can use the same idea to construct a differential characteristic for the 128-bit key variant.

We insert a difference in the state on bits v_{127}, and a difference in the key state in k_{36} and k_{80}. This difference is chosen such that the difference does not diffuse through the linear taps of the NLSFR. Since there is one active bit the bit reaches a NAND gate twice every 128 rounds. This leads to a probability of 2^{-2} per 128 rounds and a probability of 2^{-16} for $\mathcal{P}^{1024}$.

We can improve the probability of the distinguisher by a factor of 2^{-2} by moving the difference bit in the state to behind the second non-linear tap, i.e., v_{69}.

Naturally, this characteristic can be extended to the 192- and 256-bit key cases.

3.4 Key recovery attack

In the case that we allow for the attack to observer the full input and output states we can do a key recovery attack on the keyed permutation using the

characteristics discussed in Section 3.3. We use the observation that, if the output difference of the NAND is zero while one of the input differences is one, the values of the non difference bits of the NAND are 0. Thus, observing the output difference immediately leaks the value of one bit of the state in 16 rounds of the cipher. By looking at the first two state bits that are leaked we can recover two of the key bits.

Note that the two first state bits that are leaked give us the following equation:

$$v_{i+91} \oplus v_{i+85}v_{i+70} \oplus v_{i+47} \oplus v_i \oplus k_i \oplus 1 = 0$$

Thus, we can recover key bits k_0 and k_{15} in the chosen plaintext model.

By using multiple characteristics we can recover $k_0 \dots k_{37}$. The rest of the key bits can be recovered by first recovering $k_0 \dots k_{37}$ and using the recovered key bits to simplify the expressions for the subsequent bits. This allows us to recover the full key in 2^{23}, 2^{28}, and 2^{44} time and data for respectively the 128-, 256-, 192-bit primitives.

3.5 Experimental Verification

We conducted experiments to verify the existence of the related-key differential for all versions. For the 192- and 256-bit variants we verified the existence of the related-key differential through the mode after key-setup and nonce initialization step. For TinyJAMBU-192 we get zero difference after key-setup and initialization step with probability $2^{-11.8}$ and 2^{-29} respectively. For TinyJAMBU-256 we get a zero difference after the key-setup and nonce initialization step with probability $2^{-10.5}$ and $2^{-21.6}$ respectively.

We tested the probability of the differential through P_{1024} of TinyJAMBU-128 for 2^{10} random keys. For each key we have tested 2^{20} pairs of plaintexts. The experimentally verified probability of the differential is $2^{-15.5}$. The source code can be found in https://github.com/ShibamCrS/TinyAtttacksOnTinyJambu.git.

4 Forgery Attack on TinyJAMBU

Using the related key differential characteristic from Section 3 we can create a forgery attack on TinyJAMBU. We generate a key, nonce, message pair that produces the same tag for different nonces and keys. We first discuss the attack on the 256-bit key version of TinyJAMBU, but is also applicable to the 192-bit key version, although with a slightly higher complexity.

The main hurdle in creating a forgery is to find a key pair that can be used for the attack. This is due to the fact that during the key initialization the only input we have access to is the key. It is only after the key initialization, in the nonce setup, that we can insert more data. Thus only a part of the key pairs can be attacked this way. The probability that the characteristic survives through $\mathcal{P}^{1280}$ is 2^{-10}, thus only one in 2^{10} key pairs will be susceptible to a forgery

attack under a specific difference. In the case that we are interested in creating a forgery for a certain key we can try 32 key differences to increase the probability that we find a forgery under a certain key to 2^{-5}.

To find a forgery we start with 2^{10} key pairs. For each pair we encrypt the same message block with 2^{22} nonce pairs. The difference for the nonce is such that it cancels the difference after $\mathcal{P}^{640}$, i.e., we put a difference in the most significant bit of each partial nonce. We can observe the output difference through the first ciphertext. We expect to see one 'golden' key pair that survives this initial filtering step, which with high probability, follows the characteristic through the key initialization and the nonce addition.

Using this key and nonce pair we search for a forgery by changing the last nonce. Since we have to pass twice through $\mathcal{P}^{640}$ and twice through $\mathcal{P}^{1280}$ to produce both tags, the (naive) probability of finding a colliding tag is 2^{-28} after we found a golden key and nonce pair. One thing to consider is that the last finalization step is using $\mathcal{P}^{640}$ as the permutation, which as discussed in Section 3 produces a difference in the most significant bit of the state. Nevertheless, since we extract bits 64 to 95 to use as the tag, the tag difference stays zero.

The total cost of this attack is $2^{32} + 2^{28}$ data and time, where we ask for encryptions under 2^{10} related key pairs in the nonce respecting setting. If we move to the nonce misuse setting we get a forgery in $2^{32} + 2^{14}$ time since we can add the randomness in the message instead of the last nonce part. We note that finding more forgeries after finding a 'golden' key nonce pair is 2^{28} in the nonce respecting setting and 2^{14} for the nonce misuse setting.

The complexities for the forgery are low enough that we could compute a forgery on our own desktop. In Table 3 we provide key and nonce inputs that produce the same tag for both the 256- and 192-bit key cases.

4.1 TinyJAMBU-192

The forgery attack on TinyJAMBU-192 is the same as the forgery attack on TinyJAMBU-256, although the probabilities (as is discussed in Section 3) are slightly higher. The probability for the characteristic to pass through $\mathcal{P}^{640}$ and $\mathcal{P}^{1152}$ is respectively 2^{-6} and 2^{-12}. This leads to a forgery with a data and time complexity of $2^{42} + 2^{36}$ in the nonce respecting setting and a data and time complexity of $2^{42} + 2^{18}$ in the nonc misuse setting.

5 Conclusion

We discussed a full round related-key differential characteristic that can be applied to the mode. Using this characteristic we show how to create a tag forgery for TinyJAMBU-192 and TinyJAMBU-256 with a complexity of respectively 2^{32} and 2^{42} time and data using respectively 2^{10} and 2^{12} related keys. We look at the keyed permutation as a primitive and showed how to create a related-key distinguisher for the permutation with a complexity of 2^{-16} for the 128-bit key

Table 3: Key and nonce pairs that produce the same tag (and ciphertext) with the nibbles that have a difference marked. The leftmost byte is the least significant byte.

Key size	Key	Nonce	Message	Ciphertext	Tag
192	9AE19248 8B102E07 19A2492E				
	AB0F2C02 9EDB377D DF81AB70	11129DA1	C9211BA2		1734A489
	090EF19C 66F4AAEB 923635DC				1229B9F6
	9AE19248 8B102E87 19A249AE				
	AB0F2C02 8EDB377D DF81ABF0	11129DA1	C9211BA2		1734A489
	090EF09C 66F4AA6B 9236355C				1229B9F6
256	B429DBD1 14F8B269 BF8A51BD				
	7D83ABD0 3893F974 B71DC3C6				
	79626DF1 B3A3D867 8443C018	29594AD7	E015A04A		1E8CA308
	A415E2BB D5A2A68A				95CBD1F7
	B429DBD1 14F8B269 BF8A513D				
	7D83ABD0 3893F9F4 B71DC346				
	79626DF1 A3A3D867 8443C098	29594AD7	E015A04A		1E8CA308
	A415E3BB D5A2A60A				95CBD1F7

variant. We also show how to recover the key using this distinguisher in 2^{23} data and time for the 128-bit variant.

One important note to make is that TinyJAMBU-128 is the main contribution of the TinyJAMBU submission to the NIST lightweight competition. The forgery discussed in the paper does not apply to this variant. Nevertheless, as we have shown, the keyed permutation used in TinyJAMBU-128 is easily distinguished and can therefore be considered weak in the related key model. Combining this with fact that the other variants of the authenticated encryption scheme are broken, the 128-bit version should be used with care.

These attacks were mainly possible due to the fact that the key schedule of the primitive were cyclic. The protection offered against related key attacks by doing the key initialization with a constant state were not enough to protect against these attacks. To circumvent this attack the number of rounds in $\mathcal{P}^a$ and $\mathcal{P}^b$ should almost be doubled. Another easy fix for the problem would be to employ some sort of a key schedule to the permutation or to add a few additional NAND gates. We notified the designers of TinyJAMBU about the attacks which they verified. The designers suggested to use $\mathcal{P}^a$ to process the key after the initialization in the same fashion as the associated data is processed. Thus, for the 256-bit version, eight extra calls to $\mathcal{P}^a$ are made, and, in the 192-bit version, six extra calls to the permutation are made, where each of the calls processes 32 bits of the key.

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the indifferentiability of the sponge construction. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer (2008)
2. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 7118, pp. 320–337. Springer (2011)
3. Saha, D., Sasaki, Y., Shi, D., 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 (2020)
4. Sibleyras, F., Sasaki, Y., Todo, Y., Hosoyamada, A., Yasuda, K.: Birthday-bound slide attacks on TinyJAMBU's keyed permutation for all key sizes. In: Fifth Lightweight Cryptography Workshop (2022)
5. Technology, N.: Report on Lightweight Cryptography: NiSTIR 8114. CreateSpace Independent Publishing Platform (2017)
6. Wu, H., Huang, T.: TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms: Submission to NIST LwC (2019), `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf`
7. Wu, H., Huang, T.: TinyJAMBU : A family of lightweight authenticated encryption algorithms (version 2) (2021), `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf`