TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms

Designers and Submitters: Hongjun Wu and Tao Huang

Division of Mathematical Sciences
Nanyang Technological University
wuhongjun@gmail.com

29 March 2019

Contents

Chapter 1

Introduction

JAMBU is a lightweight authenticated encryption mode submitted to the CAESAR competition [4]. JAMBU is the smallest block cipher authenticated encryption mode in the CAESAR competition, and it was selected to the Third Round of the competition. JAMBU has been presented at the NIST Lightweight Cryptography Workshop 2015 [13].

In this report, we propose the TinyJAMBU mode which is a small variant of the JAMBU mode. TinyJAMBU mode is based on a keyed permutation. The state size of TinyJAMBU is only two thirds of that of JAMBU, the message block size of TinyJAMBU is half of that of JAMBU mode. When nonce is reused, TinyJAMBU provides better authentication security than JAMBU mode. The authentication security of TinyJAMBU mode is better than the Duplex mode [1] when nonce is reused (for the same permutation size and message block size).

In this report, we propose a lightweight 128-bit keyed permutation with no key schedule. The permutation is based on a 128-bit nonlinear feedback shift register. This lightweight permutation is used in the TinyJAMBU mode. The keyed permutation supports three possible key sizes: 128 bits, 192 bits, 256 bits.

For the applications in which a fixed key is embedded in the devices, and for the applications in which a secret key is stored in a device for protecting multiple messages, TinyJAMBU uses only a 128-bit register for authenticated encryption. We implemented TinyJAMBU in VHDL and synthesized it on an ASIC using 90 nm UMC technology standard cell library with a fixed 128-bit key, only 1352 Gate Equivalent (GE) are needed to achieve the full functionality of authenticated encryption and decryption. (The whole cipher is implemented, including the initialization, processing associated data, encryption/decryption, tag generation and verification. Eight rounds in parallel are computed in a clock cycle.) The throughput is about 65.1 Mbps for long associated data and 24.6 Mbps for long plaintext.

Chapter 2

TinyJAMBU Authenticated Encryption Mode

The TinyJAMBU mode is a small variant of the JAMBU mode which is a third-round candidate of the CAESAR competition. In the TinyJAMBU mode, a 128-bit keyed permutation is used, the state size is 128 its, and the message block size is 32 bits. When nonce is reused, the TinyJAMBU mode provides better authentication security than the JAMBU mode.

When nonce is reused, the TinyJAMBU mode provides better authentication security than the Duplex mode (for the same permutation size and the same message block size). The reason is that the attacker can easily set part of the state to arbitrary value when nonce is reused in the Duplex mode, while it is difficult to do that in the TinyJAMBU mode.

The TinyJAMBU mode is shown in Fig. 2.1. If the last block of the associated data (or plaintext) is not a full block , the length of the partial block (the number of bytes) is xored to the state.

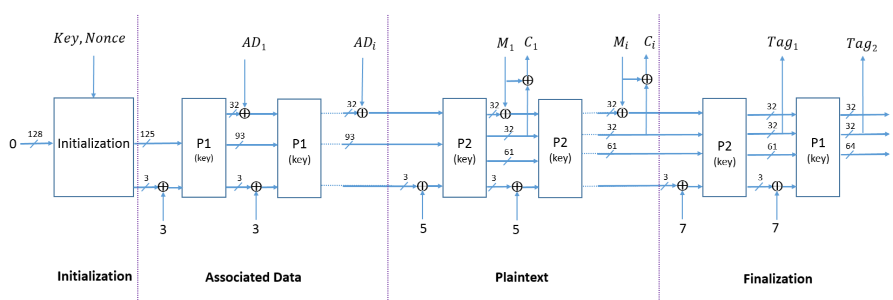

Figure 2.1: The TinyJAMBU mode for 128-bit state and keyed-permutations

Chapter 3

Specification

3.1 Recommended parameter sets

TinyJAMBU supports three key sizes: 128 bits, 192 bits and 256 bits.

- Primary member: TinyJAMBU-128
  128-bit key, 96-bit nonce, 64-bit tag, 128-bit state
- TinyJAMBU-192
  192-bit key, 96-bit nonce, 64-bit tag, 128-bit state
- TinyJAMBU-256
  256-bit key, 96-bit nonce, 64-bit tag, 128-bit state

3.2 Operations, Variables and Functions

The operations, variables and functions used in TinyJAMBU are defined below.

3.2.1 Operations

The following operations are used in the description of TinyJAMBU:

$\oplus$	:	bit-wise exclusive OR
$\&$	:	bit-wise AND
$\sim$	:	bit-wise NOT
$\|\|$	:	concatenation
$\lfloor a \rfloor$	:	floor operator, gives the integer part of a

3.2.2 Variables and Constants

The following variables and constants are used in TinyJAMBU:

$a_{\{i \cdots j\}}$	:	the word consists of $a_i \| \| a_{i+1} \| \| \cdots \| a_j$, where a_i is the ith bit of a.
AD	:	associated data, a sequence of bytes.
ad_i	:	one bit of associated data.
$adlen$	:	the length of associated data in bits.
C	:	ciphertext, a sequence of bytes
c_i	:	the ith ciphertext bit.
$FrameBits$	:	Three-bit FrameBits.
		FrameBits = 1 for nonce
		FrameBits = 3 for associated data
		FrameBits = 5 for plaintext and ciphertext
		FrameBits = 7 for finalization
$FrameBits_i$	:	The ith bit of FrameBits.
K	:	the key.
k_i	:	the ith bit of K.
$klen$	:	the key length in bits.
M	:	the plaintext, a sequence of bytes
m_i	:	the ith bit of the plaintext.
$mlen$	:	the length of the plaintext in bits.
NONCE	:	the 96-bit nonce.
nonce_i	:	the ith bit of the 96-bit nonce.
P_n	:	the 128-bit permutation with n rounds
S	:	the 128-bit state of the permutation.
s_i	:	the ith bit of the state of the permutation.
T	:	the 64-bit authentication tag.
t_i	:	the ith bit of the authentication tag.

3.2.3 The Keyed Permutation P_n

In TinyJAMBU, a 128-bit keyed permutation is used. The permutation P_n consists of n rounds. In the ith round of the permutation, a 128-bit nonlinear feedback shift register is used to update the state as follows (shown in Fig. 3.1):

StateUpdate(S, K, i):
  feedback $= s_0 \oplus s_{47} \oplus (\sim (s_{70} \& s_{85})) \oplus s_{91} \oplus k_{i \bmod klen}$
  for j from 0 to 126: $s_j = s_{j+1}$
  $s_{127} =$ feedback
end

For example, P_{384} means that the state of the permutation is updated using the function StateUpdate() for 384 times. 32 rounds of the permutation can be computed in parallel on 32-bit CPU.

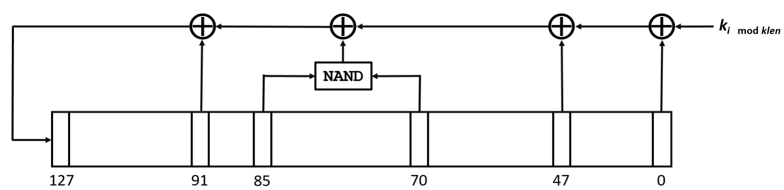

Figure 3.1: The 128-bit Nonlinear Feedback Shift Register in TinyJAMBU

3.3 TinyJAMBU-128

TinyJAMBU-128 uses a 128-bit key and a 96-bit nonce. The associated data length and the plaintext length are less than 2^{50} bytes. The authentication tag is 64-bit. The TinyJAMBU authenticated encryption mode is used in TinyJAMBU-128.

3.3.1 The initialization

In the keyed permutation of TinyJAMBU-128, the 128-bit key of TinyJAMBU-128 is used, and the $klen$ is set to 128.

The initialization of TinyJAMBU-128 consists of two stages: key setup and nonce setup.

Key Setup. The key setup is to randomize the state using the keyed permutation P_{1024}.

1. Set the 128-bit state S as 0.

2. Update the state using P_{1024}.

Nonce Setup. The nonce setup consists of three steps. In each step, the Framebits of nonce (the value is 1) are XORed with the state, then we update the state using the keyed permutation P_{384}, then 32 bits of the nonce are XORed with the state.

for i from 0 to 2:
    $s_{\{36\cdots 38\}} = s_{\{36\cdots 38\}} \oplus FrameBits_{\{0\cdots 2\}}$
    Update the state using P_{384}
    $s_{\{96\cdots 127\}} = s_{\{96\cdots 127\}} \oplus nonce_{\{32i\cdots 32i+31\}}$
end for

3.3.2 Processing the associated data

After the initialization, we process the associated data AD. In each step, the Framebits of associated data (the value is 3) are XORed with the state, then we update the state using the keyed permutation P_{384}, then 32 bits of the associated data are XORed with the state.

Processing the full blocks of associated data:

for i from 0 to $\lfloor adlen/32 \rfloor$:
    $s_{\{36\cdots 38\}} = s_{\{36\cdots 38\}} \oplus FrameBits_{\{0\cdots 2\}}$
    Update the state using P_{384}
    $s_{\{96\cdots 127\}}\} = s_{\{96\cdots 127\}} \oplus ad_{\{32i\cdots 32i+31\}}$
end for

Processing the partial block of associated data. If the last block is not a full block (it is called a partial block), the last block is XORed to the state, and the number of bytes of associated data in the partial block is XORed to the state.

if ($adlen$ mod 32) > 0:
    $s_{\{36\cdots 38\}} = s_{\{36\cdots 38\}} \oplus FrameBits_{\{0\cdots 2\}}$
    Update the state using P_{384}
    $lenp = adlen$ mod 32 /* number of bits in the partial block */
    $startp = adlen - lenp$ /* starting position of the partial block */
    $s_{\{96\cdots 96+lenp-1\}} = s_{\{96\cdots 96+lenp-1\}} \oplus ad_{\{startp\cdots adlen-1\}}$

    /* the number of bytes in the partial block is XORed to the state*/
    $s_{\{32\cdots 33\}} = s_{\{32\cdots 33\}} \oplus (lenp/8)$
end if

3.3.3 The encryption

After processing the associated data, we encrypt the plaintext M. In each step, the Framebits of plaintext (the value is 5) are XORed with the state, then we update the state using the keyed permutation P_{1024}, then 32 bits of the plaintext are XORed with the state, and we obtain 32 bits of ciphertext by XORing the plaintext with another part of the state.

Processing the full blocks of plaintext:

for i from 0 to $\lfloor mlen/32 \rfloor$:
    $s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
    Update the state using P_{1024}
    $s_{\{96\cdots127\}} = s_{\{96\cdots127\}} \oplus m_{\{32i\cdots32i+31\}}$
    $c_{\{32i\cdots32i+31\}} = s_{\{64\cdots95\}} \oplus m_{\{32i\cdots32i+31\}}$
end for

Processing the partial block of plaintext. If the last block is not a full block (it is a partial block), the last block is XORed to the state, and the number of bytes in the partial block is XORed to the state.

if $(mlen \bmod 32) > 0$:
    $s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
    Update the state using P_{1024}
    $lenp = mlen \bmod 32$ /* number of bits in partial block */
    $startp = mlen - lenp$ /* starting position of partial block */
    $s_{\{96\cdots96+lenp-1\}} = s_{\{96\cdots96+lenp-1\}} \oplus m_{\{startp\cdots mlen-1\}}$
    $c_{\{startp\cdots mlen-1\}} = s_{\{64\cdots64+lenp-1\}} \oplus m_{\{startp\cdots mlen-1\}}$
    /* the length (bytes) of the last partial block is XORed to the state*/
    $s_{\{32\cdots33\}} = s_{\{32\cdots33\}} \oplus (lenp/8)$
end if

3.3.4 The finalization

After encrypting the plaintext, we generate the 64-bit authentication tag T as follows. The Framebits of finalization (the value is 7) are XORed with the state.

$s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using P_{1024}
$t_{\{0\cdots31\}} = s_{\{64\cdots95\}}$

$s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using P_{384}
$t_{\{32\cdots63\}} = s_{\{64\cdots95\}}$

3.3.5 The decryption

In a decryption process, the initialization and processing the associate data are the same as the encryption process. After processing the associated data, we decrypt the ciphertext C. In each step, the Framebits of plaintext (the value is 5) are XORed with the state, then we update the state using the keyed permutation P_{1024}. We obtain 32 bits of plaintext by XORing the ciphertext with 32 state bits $s_{\{64\cdots95\}}$, then the plaintext is XORed with the state bits $s_{\{96\cdots127\}}$.

Processing the full blocks of ciphertext:

for i from 0 to $\lfloor mlen/32 \rfloor$:
 $s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
 Update the state using P_{1024}
 $m_{\{32i\cdots32i+31\}} = s_{\{64\cdots95\}} \oplus c_{\{32i\cdots32i+31\}}$
 $s_{\{96\cdots127\}} = s_{\{96\cdots127\}} \oplus m_{\{32i\cdots32i+31\}}$
end for

Processing the partial block of ciphertext. If the last block is not a full block (it is a partial block), the number of bytes in the partial block is XORed to the state.

if ($mlen$ mod 32) > 0:
 $s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
 Update the state using P_{1024}
 $lenp = mlen$ mod 32 /* number of bits in partial block */
 $startp = mlen - lenp$ /* starting position of partial block */
 $m_{\{startp\cdots mlen-1\}} = s_{\{64\cdots64+lenp-1\}} \oplus c_{\{startp\cdots mlen-1\}}$
 $s_{\{96\cdots96+lenp-1\}} = s_{\{96\cdots96+lenp-1\}} \oplus m_{\{startp\cdots mlen-1\}}$
 /* the length (bytes) of the last partial block is XORed to the state*/
 $s_{\{32\cdots33\}} = s_{\{32\cdots33\}} \oplus (lenp/8)$
end if

3.3.6 The verification

After decrypting the plaintext, we generate a 64-bit authentication tag T', then compare T' with the received tag T. The Framebits of finalization are of value 7.

$s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using P_{1024}
$t'_{\{0\cdots31\}} = s_{\{64\cdots95\}}$
$s_{\{36\cdots38\}} = s_{\{36\cdots38\}} \oplus FrameBits_{\{0\cdots2\}}$
Update the state using P_{384}
$t'_{\{32\cdots63\}} = s_{\{64\cdots95\}}$

$T' = t'_{\{0\cdots63\}}$. Accept the message if $T' = T$; otherwise, reject.

3.4 TinyJAMBU-192

TinyJAMBU-192 uses a 192-bit key and a 96-bit nonce. The associated data length and the plaintext length are less than 2^{50} bytes. The authentication tag is 64-bit.

The design of TinyJAMBU-192 is similar to that of TinyJAMBU-128. The differences between TinyJAMBU-192 and TinyJAMBU-128 are given below.

Keyed Permutation. In the keyed permutation of TinyJAMBU-192, a 192-bit key is used, and the *klen* is set to 192.

The keyed permutation P_{1024} in TinyJAMBU-128 is replaced with P_{1152} in TinyJAMBU-256. P_{384} in TinyJAMBU-128 is still P_{384} in TinyJAMBU-192.

3.5 TinyJAMBU-256

TinyJAMBU-256 uses a 256-bit key and a 96-bit nonce. The associated data length and the plaintext length are less than 2^{50} bytes. The authentication tag is 64-bit.

The design of TinyJAMBU-256 is similar to that of TinyJAMBU-128. The differences between TinyJAMBU-256 and TinyJAMBU-128 are given below.

Keyed Permutation. In the keyed permutation of TinyJAMBU-256, a 256-bit key is used, and the *klen* is set to 256.

The keyed permutation P_{1024} in TinyJAMBU-128 is replaced with P_{1280} in TinyJAMBU-256. P_{384} in TinyJAMBU-128 is still P_{384} in TinyJAMBU-256.

Chapter 4

Security Goals

In TinyJAMBU, each pair of key and nonce is used to protect only one message. If verification fails, the new tag and the decrypted plaintext should not be given as output.

In TinyJAMBU, the associated data plays the same role as nonce. When the same nonce but different associated data are used for a key, it is equivalent to the use of unique nonce for the key.

The nonce misuse happens when the same nonce and the same associated data are reused for a key. When nonce is misused, TinyJAMBU provides strong protection of the secret key, and provides strong authentication security, but provides weak protection of the plaintext.

4.1 Security goals with unique nonce

The security goals of TinyJAMBU for unique nonce are given in Table 4.1. We assume that each each key is used to process at most 2^{50} byes of messages (associated data, plaintext/ciphertext), and each message is at least 8 bytes. Note that the authentication security in Table 4.1 includes the integrity security of plaintext, associated data and nonce.

Table 4.1: Security Goals of TinyJAMBU with Unique Nonce

	Encryption	Authentication
TinyJAMBU-128	112-bit	64-bit
TinyJAMBU-192	168-bit	64-bit
TinyJAMBU-256	224-bit	64-bit

4.2 Security goals with misused nonce

When nonce is misused in TinyJAMBU (the same nonce and the same associated data are reused for a key), the secret key of TinyJAMBU remains strong, and the authentication of TinyJAMBU remains strong (suppose that a secret key is used to process at most 2^{50} bytes of data, and each message consists of at least 8 bytes of data).

When nonce is reused, an attacker is able to decrypt the ciphertext since the encryption of TinyJAMBU is somehow similar to the Cipher Feedback mode.

The security goals of TinyJAMBU for reused nonce are given in Table 4.2.

Table 4.2: Security Goals of TinyJAMBU with Repeated Nonce (an adversary has the maximum forgery advantage when a key was used to process adaptively chosen 2^{50} bytes of data, and each message is at least 8 bytes)

	Secret Key	Authentication	Max. Forgery Adv.
TinyJAMBU-128	112-bit	64-bit	2^{-15}
TinyJAMBU-192	168-bit	64-bit	2^{-15}
TinyJAMBU-256	224-bit	64-bit	2^{-15}

Chapter 5

Security of the TinyJAMBU Mode

In this section, we analyse the security of TinyJAMBU mode. To simplify the analysis, we consider $P_K : \{0,1\}^b \times \{0,1\}^k \to \{0,1\}^b$ as an underlying ideal keyed-permutation used in the mode, and use it to replace $P1$ and $P2$.

Our security proof is inspired by the security proof for the sponge function [8] and the SAEB mode [12].

5.1 Security Model

Let $\mathcal{E}$ be the authenticated encryption function, which takes key, nonce, associated data and plaintext as input and outputs ciphertext and authentication tag. The authenticated encryption process can be denoted as $(C, T) \longleftarrow \mathcal{E}_K(N, AD, M)$.

Similarly, the authenticated decryption function is $\mathcal{D}$, which takes key, nonce, associated data, ciphertext and authentication tag as input, and outputs plaintext when verification is successful or $\perp$ otherwise. We denote the decryption and verification process as $P/\perp \longleftarrow \mathcal{D}_K(N, AD, C, T)$.

We consider an adversary $\mathcal{A}$ that has access to $(\mathcal{E}_K,\ \mathcal{D}_K)$ and an oracle $(\$, \perp)$ where \$ takes (N, AD, M) as input and returns a random bit string with the same length as C; $\perp$ is an oracle that returns the reject symbol $\perp$ for any query. We assume that the adversary does not make any trivial authenticated encryption queries or trivial authenticated encryption queries. In addition, we assume that key is unknown to the adversary. Hence, the adversary does not have access to the underlying permutation. Let $q_{\mathcal{E}}$ and $q_{\mathcal{D}}$ denote the maximum number of queries that the adversary make for authenticated encryption and authenticated decryption respectively. Let $\sigma_{\mathcal{E}}$ and $\sigma_{\mathcal{D}}$ denote the total number of blocks involved in authenticated encryption and authenticated decryption respectively. When consider j-th authenticated encryption query, $\sigma_{\mathcal{E},j}$ is used to denote the total number of keyed permutation invocations. Similarly, $\sigma_{\mathcal{D},j}$ is the total number of keyed permutation invocations in the j-th decryption and verification query.

5.2 Privacy

In the following, we assume that the secret key is unknown to the adversary. We define the security of TinyJAMBU on privacy against nonce-respecting adversary $\mathcal{A}$ in a chosen plaintext attack setting as follows:

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{TinyJAMBU}} = |\Pr[\mathcal{A}^{\mathrm{TinyJAMBU}_{\mathcal{E}_K}} = 1] - \Pr[\mathcal{A}^{\$} = 1]|,$$

where $\mathcal{A}^{\mathrm{TinyJAMBU}_{\mathcal{E}_K}}$ refers to the case that the adversary interacts with TinyJAMBU authenticated encryption and $\mathcal{A}^{\$}$ refers to the case that the adversary interacts with a random oracle.

The following theorem gives the security bound on privacy for TinyJAMBU.

Thoerem 1. *Let $\mathcal{A}$ be a nonce-respecting adversary makes $q_{\mathcal{E}}$ encryption queries of at most $\sigma_{\mathcal{E}}$ blocks. Then*

$$\mathbf{Adv}^{priv}_{TinyJAMBU} \leq \frac{\sigma_{\mathcal{E}}^2}{2^{n+1}}.$$

Proof. We use $s := ([s]_v \parallel [s]_r \parallel [s]_c)$ to denote the internal state of the authenticated encryption scheme, where $[s]_v$ is the v-bit state segment that the message blocks inject into; $[s]_r$ is the r-bit state segment that is used as keystream, which is similar to the *rate* segment in the Duplex sponge mode; $[s]_c$ is the c-bit state segment that only frame bits are xored with, which is similar to the *capacity* segment in the Duplex sponge mode. $s_{j,k}$ is used to denote the internal state at the k-th block of j-th query, where $1 \leq k \leq \sigma_{\mathcal{E},j}$, $1 \leq j \leq q_{\mathcal{E}}$.

We start out proof by defining the following event:

`Coll`: $(s_{j,k-1} \neq s_{j',k'-1})$ and $(s_{j,k} = s_{j',k'})$ for $j, j' \in 1, ..., q_{\mathcal{E}}$, $k \in 1, ..., \sigma_{\mathcal{E},j}$ and $k' \in 1, ..., \sigma_{\mathcal{E},j'}$.

Note that for nonce-respecting adversaries, `Coll` occurs if and only if an internal state collision occurs.

In regard to the difference between $\mathrm{TinyJAMBU}_{\mathcal{E}_K}$ and \$, we have the following Lemma:

Lemma 1. *The output of $TinyJAMBU_{\mathcal{E}_K}$ and \$ are identically distributed until the event `Coll` occurs.*

Proof. From the definition of \$, the output (C, T) is a uniformly random bit string. Consider $\mathcal{E}_K$, without event `Coll`, the input of each keyed-permutation invocation is a new value. Since P_K is an ideal keyed-permutation, the output of P_K is also uniformly random, which is identical to the output of \$. □

From Lemma 1, the advantage of adversary is bounded by the probability that the event `Coll` occurs. Thus, we have

$$\mathbf{Adv}^{\text{priv}}_{\text{TinyJAMBU}} \leq \Pr[\texttt{Coll}]$$

Given $\sigma_{\mathcal{E}}$ keyed permutation invocations, the probability of event `Coll` is upper bounded by $\binom{\sigma_{\mathcal{E}}}{2} \cdot 2^{-n}$.

Thus, we have

$$\mathbf{Adv}^{\text{priv}}_{\text{TinyJAMBU}} \leq \frac{\sigma_{\mathcal{E}}^2}{2^{n+1}}.$$

□

In the TinyJAMBU authenticated cipher, we have $b = 128$, $r = v = 32$, $c = 64$ and $\sigma \leq 2^{48}$. Hence, after 2^{48} queries, the security bound for TinyJAMBU is $\mathbf{Adv}^{\text{priv}}_{\text{TinyJAMBU}} \leq 2^{96}/2^{129} \approx 2^{-33}$. It means that when nonce is not reused, the adversary has the advantage of 2^{-33} to distinguishing ciphertext from random after 2^{50} bytes of plaintext are processed.

5.3 Authenticity

For the authenticity of TinyJAMBU mode, we will prove the security for a nonce-reuse adversary $\mathcal{A}$. In the following, we assume that the secret key is unknown to the adversary.

We define the authenticity of TinyJAMBU as follows:

$$\mathbf{Adv}^{auth}_{\text{TinyJAMBU}} = \Pr[\mathcal{A}^{\text{TinyJAMBU}_{\mathcal{E}_K}, \text{TinyJAMBU}_{\mathcal{D}_K}} forges.]$$

Thoerem 2. *Let $\mathcal{A}$ be a nonce-reuse adversary makes $q_{\mathcal{E}}$ authenticated encryption queries of at most $\sigma_{\mathcal{E}}$ blocks and $q_{\mathcal{D}}$ authenticated decryption queries of at most $\sigma_{\mathcal{D}}$ blocks. Then*

$$\mathbf{Adv}^{auth}_{TinyJAMBU} \leq (\frac{e\sigma_{\mathcal{E}}}{\rho 2^r})^{\rho} \cdot \frac{2^r}{\sqrt{\rho}} + \frac{(\sigma_{\mathcal{E}} + \sigma_{\mathcal{D}})(\rho - 2)}{2^{c+v/2+1}} + \frac{q_{\mathcal{D}}}{2^t},$$

where e is the Euler's number and ρ is a positive integer constant.

Proof. We use similar notations as in the proof of Theorem 1 but with a few adjustments to include the decryption. We use $s^{\mathcal{E}}$ and $s^{\mathcal{D}}$ to denote the internal state in encryption and decryption queries respectively. So $s^{\mathcal{E}}_{j,k}$ refers to the k-th keyed-permutation invocation in the j-th encryption query.

Then we define the following events:

1. $\texttt{Coll}_{\mathcal{E}}$: $(s^{\mathcal{E}}_{j,k-1} \neq s^{\mathcal{E}}_{j',k'-1})$ and $(s^{\mathcal{E}}_{j,k} = s^{\mathcal{E}}_{j',k'})$ for $j, j' \in 1, ..., q_{\mathcal{E}}$, $k \in 1, ..., \sigma_{\mathcal{E},j}$ and $k' \in 1, ..., \sigma_{\mathcal{E},j'}$;
2. $\texttt{Coll}_{\mathcal{D}}$: $(s^{\mathcal{D}}_{j,k-1} \neq s^{\mathcal{E}}_{j',k'-1})$ and $(s^{\mathcal{D}}_{j,k} = s^{\mathcal{E}}_{j',k'})$ for $j \in 1, ..., q_{\mathcal{D}}$, $j' \in 1, ..., q_{\mathcal{E}}$, $k \in 1, ..., \sigma_{\mathcal{D},j}$ and $k' \in 1, ..., \sigma_{\mathcal{E},j'}$;

3. $\texttt{MultiColl}$: there exists ρ multi-collisions on the rate segment of state for a positive integer ρ.

The event $\texttt{Coll}_{\mathcal{E}}$ is exactly the same as $\texttt{Coll}$ in the proof of Theorem 1. The event $\texttt{Coll}_{\mathcal{D}}$ is an authenticated decryption query has a non-trivial state collision with some authenticated encryption query. Here 'trivial state collision' refers to a state collision due to common prefix blocks.

A successful forgery can either from an internal state collision or a random guess without state collision. Here we only need to consider the internal state collisions in the two collision events $\texttt{Coll}_{\mathcal{E}}$ and $\texttt{Coll}_{\mathcal{D}}$. Note that the collisions in two authenticated decryption queries will not lead to a forgery. The probability that a successful random guess without state collision is $1/2^t$. Hence, we upper bound the probability of forgeries as follows:

$$\Pr[\mathcal{A}^{\text{TinyJAMBU}_{\mathcal{E}_K},\text{TinyJAMBU}_{\mathcal{D}_K}} forges] \leq \Pr[\texttt{Coll}_{\mathcal{E}}] + \Pr[\texttt{Coll}_{\mathcal{D}}] + \frac{q_{\mathcal{D}}}{2^t}.$$

Now we compute the probability of the state collision events under the consideration of the number of multi-collisions ($\#MC$) on the rate segment of the state. As nonce reuse in our setting, it is possible to query multiple messages with the same prefix blocks. However, we consider this case as the same class of multi-collision. Only when the previous blocks are different, two blocks can be considered as different multi-collisions. We set a positive integer ρ. Let $f(t)$ be the probability that a state collision when t multi-collisions on the rate segment present. When the number of multi-collisions is less than ρ, the probability of state collision is given by:

$$\Pr[\texttt{Coll}_{<\rho}] = \sum_{t=0}^{\rho-1} \Pr[\#MC = t] \cdot f(t).$$

When the number of multi-collisions is at least ρ and of course at most σ, the probability of state collision is given by:

$$\Pr[\texttt{Coll}_{>=\rho}] = \sum_{t=\rho}^{\sigma} \Pr[\#MC = t] \cdot f(t) \leq \Pr[\#MC >= \rho].$$

With the definition of the event $\texttt{MultiColl}$, we have the following inequality:

$$\begin{aligned}\Pr[\texttt{Coll}_{\mathcal{E}}] + \Pr[\texttt{Coll}_{\mathcal{D}}] \leq \quad & \Pr[\texttt{Coll}_{\mathcal{E}} | \neg(\texttt{MultiColl})] + \\ & \Pr[\texttt{Coll}_{\mathcal{D}} | \neg(\texttt{MultiColl})] + \\ & \Pr[\texttt{MultiColl}].\end{aligned}$$

Using the result from Equation (4) in [8], we can bound the probability of $\texttt{MultiColl}$ as:

$$\Pr[\texttt{MultiColl}] \leq (\frac{e\sigma_{\mathcal{E}}}{\rho 2^r})^{\rho} \cdot \frac{2^r}{\sqrt{\rho}}. \tag{5.1}$$

Next, we analyse the probability for event $\texttt{Coll}_{\mathcal{E}}$ when $\texttt{MultiColl}$ is false. Consider a state $s^{\mathcal{E}}_{\alpha,i}$, which is the i-th block in the α-th query. Suppose that $\texttt{Coll}_{\mathcal{E}}$ occurs between $s^{\mathcal{E}}_{\alpha,i}$ and a previous block $s^{\mathcal{E}}_{\beta,j}$, we compute the probability as follows.

Case 1. $s^{\mathcal{E}}_{\alpha,i-1}$ is a new value which is not equal to any of the previous states. In this case, $P_K(s^{\mathcal{E}}_{\alpha,i-1})$ is randomly chosen from $\{0,1\}^b$. Hence, the probability that $\texttt{Coll}_{\mathcal{E}}$ occurs between $s^{\mathcal{E}}_{\alpha,i}$ and a previous block $s^{\mathcal{E}}_{\beta,j}$ is bounded by $(\sum_{i=1}^{\alpha-1}\sigma_{\mathcal{E},i}+i-1)\cdot\frac{1}{2^n}$.

Case 2. $s^{\mathcal{E}}_{\alpha,i-1}$ is a state that has been queried before. In this case, the output of $P_K(s^{\mathcal{E}}_{\alpha,i-1})$ is a fixed value. Since $s^{\mathcal{E}}_{\alpha,i} = P_K(s^{\mathcal{E}}_{\alpha,i-1}) \oplus (M \parallel 0^r \parallel (frameBits \parallel 0^{c-4})$, $[s^{\mathcal{E}}_{\alpha,i}]_r$ is a known value. The event $\texttt{Coll}_{\mathcal{E}}$ implies that $[s^{\mathcal{E}}_{\alpha,i}]_r = [s^{\mathcal{E}}_{\beta,j}]_r$ and $s^{\mathcal{E}}_{\alpha,i-1} \neq s^{\mathcal{E}}_{\beta,j-1}$. Suppose that there are θ classes of multi-collisions on $[s^{\mathcal{E}}_{\alpha,i}]_r$, which are $\Gamma_1,\Gamma_2,...,\Gamma_\theta$ such that the previous states are mutually distinct. For each Γ_i, there are β_i different messages been queried before. Therefore, the probability that the message segment $[s^{\mathcal{E}}_{\alpha,i}]_v$ collide with a previous blocks queried in class Γ_i is given by $\beta_i/2^v$. The rate segments $[s^{\mathcal{E}}_{\alpha,i}]_r$ must collide from the definition of multi-collision. The capacity segment has collision probability $1/2^c$ since the previous blocks are different. In summary, the probability that $s^{\mathcal{E}}_{\alpha,i}$ collides with a previous state is $\Sigma_{i=1}^{\theta}\beta_i/2^{c+v}$.

Next, we compute the state collision probability for all authenticated encryption queries. We divide the state s into disjoint sets $\Theta_1 \bigcup \Theta_2 \bigcup ... \bigcup \Theta_t$ according to the values of the rate segment $[s]_r$. Clearly, any state collision can only occur within the same set.

Then we compute the collision probability within a set, say Θ_x. Suppose that there are θ multi-collision classes, $\Gamma_1,\Gamma_2,...,\Gamma_\theta$, in Θ_x. Each class Γ_i is corresponding to β_i different messages with the same input state. Then, the probability that a state collision between any two classes Γ_i and Γ_j is given by $\min\{\frac{\beta_i\beta_j}{2^v},1\}\cdot\frac{1}{2^c}$. Thus, the overall probability that there is a state collision in set Θ_x is

$$\Pr[Coll\mathcal{E} \text{ in } \Theta_x|\neg(\texttt{MultiColl})] = \sum_{i\neq j,1\leq i,j\leq\theta}\min\{\frac{\beta_i\beta_j}{2^v},1\}\cdot\frac{1}{2^c} \qquad (5.2)$$

When $\beta_i = 2^{v/2}$ for all $1 \leq i \leq \theta$, the above Equation 5.2 has the maximum probability $\binom{\theta}{2}\cdot\frac{1}{2^c}$ with the minimum number of block queries $\gamma = \sum_{i=1}^{\theta}\beta_i = 2^{v/2}\theta$. We explain the details in Appendix A.

Since the event $\texttt{MultiColl}$ is false, the maximum possible number of multi-collisions on the rate segment is $\rho - 1$. Thus, $\theta \leq \rho - 1$. We have

$$\Pr[Coll\mathcal{E} \text{ in } \Theta_x|\neg(\texttt{MultiColl})] \leq \binom{\rho-1}{2}\frac{1}{2^c} = \frac{(\rho-1)(\rho-2)}{2^{c+1}}.$$

Given the total number of encryption query blocks $\sigma_{\mathcal{E}}$, the maximum number of disjoint sets that can reach maximum probability is $\frac{\sigma_{\mathcal{E}}}{\gamma} = \frac{\sigma_{\mathcal{E}}}{(\rho-1)2^{v/2}}$.

Finally, we are able to compute

$$\begin{aligned} \Pr[Coll\mathcal{E}|\neg(\texttt{MultiColl})] &\leq \frac{\sigma_{\mathcal{E}}}{(\rho-1)2^{v/2}} \cdot \frac{(\rho-1)(\rho-2)}{2^{c+1}} \\ &= \frac{\sigma_{\mathcal{E}}(\rho-2)}{2^{c+v/2+1}}. \end{aligned}$$

Generally, we have $\sigma_{\mathcal{E}} \cdot 2^{-r} \leq \rho$ from the Pigeonhole principle. So the probability of *Case 2* is at least the probability of *Case 1*. Hence, the sum of probabilities in *Case 2* gives the bound for $\Pr[\texttt{Coll}]_{\mathcal{E}}$:

$$\Pr[\texttt{Coll}_{\mathcal{E}}|\neg(\texttt{MultiColl})] \leq \frac{\sigma_{\mathcal{E}}(\rho-2)}{2^{c+v/2+1}}.$$

We bound the probability for event $\texttt{Coll}_{\mathcal{D}}$ when $\texttt{MultiColl}$ is false. The probability that a $\texttt{Coll}_{\mathcal{D}}$ occurs between $s^{\mathcal{D}}_{\alpha,i}$ and some block from authenticated encryption $s^{\mathcal{E}}_{\beta,j}$ can be computed in a similar way as the computation for $\texttt{Coll}_{\mathcal{E}}$, resulting the following bound:

$$\Pr[\texttt{Coll}_{\mathcal{D}}|\neg(\texttt{MultiColl})] \leq \frac{\sigma_{\mathcal{D}}(\rho-2)}{2^{c+v/2+1}}.$$

In summary, we have

$$\mathbf{Adv}^{auth}_{\text{TinyJAMBU}} \leq (\frac{e\sigma_{\mathcal{E}}}{\rho 2^r})^{\rho} \cdot \frac{2^r}{\sqrt{\rho}} + \frac{(\sigma_{\mathcal{E}}+\sigma_{\mathcal{D}})(\rho-2)}{2^{c+v/2+1}} + \frac{q_{\mathcal{D}}}{2^t} \tag{5.3}$$

□

In the TinyJAMBU authenticated cipher, $b = 128$, $r = v = 32$, $c = 64$ and $\sigma \leq 2^{48}$. For authenticity security, we choose $\rho = 2^{17.445}$, after substituting the values of r and σ, the security bound for authenticity is $\mathbf{Adv}^{auth}_{\text{TinyJAMBU}} \leq 2^{-387} + 2^{17.445} \cdot (\sigma_{\mathcal{E}} + \sigma_{\mathcal{D}})/2^{81} + (q_{\mathcal{E}} + q_{\mathcal{D}})/2^{64}$. When 2^{48} blocks are queried, the adversary has the advantage of less than $2^{-15.5}$ to forge a message successfully.

Note that if Duplex mode is used for the same setting on block size and rate size, when 2^{48} blocks are queried , the adversary has the advantage around 2^{-1} to forge a message successfully.

Chapter 6

Security Analysis

6.1 Properties of the TinyJAMBU Mode

In an authenticated encryption mode, the probability of state collision plays the role for protecting the confidentiality of plaintext and for resisting the forgery attack on the message. The security proof of the TinyJAMBU mode were given in Chapter 5. In the following, we provide a simple analysis on the state collision of the TinyJAMBU mode.

6.1.1 State collision for unique nonce

For unique nonce, a new state is generated for each message. It is impossible to apply the adaptively chosen message attack on the TinyJAMBU mode when the nonce is unique since each message is processed using a new initial state.

Only the state size affects state collision when the nonce is unique. When we use the 128-bit state in the TinyJAMBU mode, the probability is $\binom{2^{48}}{2} \times 2^{-128} = 2^{-33}$ when a key is used to process 2^{50} message bytes (there are about 2^{48} message blocks if assume that each message is at least 8 bytes long).

6.1.2 State collision for repeated nonce

For repeated nonce, the adaptively chosen message can be applied to improve the probability of state collision.

In TinyJAMBU, after processing each plaintext block, 32 bits of the ith state are known to the attacker as keystream (denoted as U_{32}^i); 32 bits of the state are unknown to the attacker, but the attacker is able to modify these 32 bits through message injection (denoted as V_{32}^i); 64 bits of the state are unknown to the attacker (denoted as W_{64}^i.

When 2^{48} message blocks are processed, the multicollisions at U_{32} can be observed. On average, each 32-bit value of U_{32} appears 2^{16} times.

We consider 2^{16} states with the same value at U_{32}, and inject 2^{16} message blocks to V_{32} for each state to introduce collision at V_{32} with probability about

1 (i.e., two different state state before the message injection now have the same value at V_{32} for some message blocks). After the message injections, there are about 2^{16} states which are the same at $U_{32} \parallel V_{32}$. The chance that there is a whole state collision is about $\binom{2^{16}}{2} \times 2^{-64} = 2^{-33}$ after the message injections. We need to adaptively choose $2^{16} \times 2^{16} = 2^{32}$ message blocks here to carry out this attack.

Since the attacker is able to adaptively choose up to 2^{48} message blocks, the attacker is able to perform the above attack for 2^{16} times. Thus the success rate of the whole state collision is $2^{-33} \times 2^{16} = 2^{-17}$.

Experiment. We implemented the above attack using 48-bit state size instead of the 128-bit sate size. The U and V are 12 bits, W is 24 bits. The attacker can use 2^{18} chosen message blocks. We repeated the attack for 2^{15} times, the success rate for a state collision is $2^{-7.02}$. (The experimental result matches the theoretical result very well. The theoretical result is 2^{-7}.)

6.2 Properties of the Keyed Permutation P_n

6.2.1 Differential properties of the keyed permutation P_n

In this section, we analyse the differential properties [2, 3] of the TinyJAMBU permutation P_n. The following three types of differences will be analysed.

- Type 1. Input differences at $S_{96\cdots127}$
- Type 2. Arbitrary input differences, output differences at $S_{96\cdots127}$
- Type 3. Input differences at $S_{96\cdots127}$, output differences at $S_{96\cdots127}$

In the following, we will analyse the differential propagation using the Mixed Integer Linear Programming (MILP) [11]. We use the Gurobi optimizer [6] to find the bounds for the permutations.

Type 1 Differences

For the Type 1 differences, the input differences are at $S_{96\cdots127}$, and there is no restriction on the output differences. The largest differential probabilities of the Type 1 differences are summarized in Table 6.1.

Type 2 Differences

For the Type 2 differences, there is no restriction on the input differences, and output differences are at $S_{96\cdots127}$. The largest differential probabilities of the Type 2 differences are summarized in Table 6.2.

Table 6.1: Type 1 Differential Properties of P_n

Round	Probability	Method
256	2^{-22}	MILP
320	2^{-33}	MILP
384	2^{-45}	MILP
448	2^{-55}	MILP

Table 6.2: Type 2 Differential Properties of P_n

Round	Probability	Method
384	2^{-28}	MILP

Type 3 Differences

For the Type 3 differences, the input differences are at $S_{96\cdots127}$, and output differences are at $S_{96\cdots127}$. The largest differential probabilities of the Type 3 differences are summarized in Table 6.3.

Table 6.3: Type 3 Differential Properties of P_n

Round	Probability	Method
384	$\leq 2^{-78}$	MILP

6.2.2 Linear properties of the keyed permutation P_n

In this section, we analyse the linear properties [9, 10] of the TinyJAMBU permutation P_n . The linear bias for output bits at $S_{64\cdots95}$ will be analysed.

In the analysis, we use the Mixed Integer Linear Programming (MILP) [11]. We will use the Gurobi optimizer [6] to find the linear bias of the permutations. The results are summarized in Table 6.4.

6.2.3 Algebraic properties of the keyed permutation P_n

We consider the algebraic property for the input bits at $S_{96\cdots127}$. Our experiment shows that after 598 rounds, every output bit at $S_{64\cdots95}$ is affected by the 32-bit input cube tester [5] at $S_{96\cdots127}$.

Table 6.4: Linear bias of P_n

Round	Bias
256	2^{-13}
320	2^{-17}
384	2^{-23}
448	2^{-27}

6.3 Forgery Attacks

For an authenticated encryption scheme, an internal state collision will directly lead to a forgery attack. To produce a state collision, an attacker can inject difference into nonce, associated data or plaintext/ ciphertext, then eliminate the difference in the state using the difference in the later input blocks.

6.3.1 Forgery attacks on nonce and associated data

In TinyJAMBU, each 32-bit nonce block and associated data block is processed using P_{384} (with different Framebits for nonce and associated data). The associated data also plays the role of nonce in TinyJAMBU.

Nonce and associated data are processed in a very similar way in TinyJAMBU (the only difference is the used of different FrameBits). In the following, we only need to consider the forgery attacks on associated data. There are two cases of forgery attacks on the associated data:

Case 1. Forgery attacks with differences at only two adjacent associated data blocks, i.e., $\Delta ad_i \neq 0$ and $\Delta ad_{i+1} \neq 0$

For this type of forgery attacks, the input difference to P_{384} is at $s_{96\cdots127}$, and the output difference is at $s_{96\cdots127}$. This is the Type 3 differences analysed in Sect. 6.2.1. According to Tabl 6.3, the largest differential probability of Type 3 differences of P_{384} is at most 2^{-78}. It means that the forgery attack succeeds with probability at most 2^{-78} using this type of differential attack.

Case 2. Forgery attacks involving more than two associated data blocks, i.e., $\Delta ad_i \neq 0$ and $\Delta ad_j \neq 0$, where $j > i + 1$. (Δad_w may or may not be zero for $i < w < j$.)

For this type of forgery attacks, at least two permutations P_{384} are involved. Δad_i introduces input difference at $s_{96\cdots127}$ of P_{384}, this differential probability in P_{384} is at most 2^{-45} (according to Table 6.1). To eliminate the difference at Δad_j, the output difference of P_{384} should appear at $s_{96\cdots127}$ of P_{384}, this differential probability in P_{384} is at most 2^{-28} (according to Table 6.2). Thus the forgery attack succeeds with probability at most $2^{-55} \times 2^{-28} = 2^{-73}$. The actual forgery is expected to be successful with probability less than 2^{-73} since the two optimal differentials may not match in the middle.

The above analysis shows that the differential forgery attack on nonce and associated data succeeds with probability at most $\max(2^{-78}, 2^{-73}) = 2^{-73}$.

6.3.2 Forgery attacks on plaintext/ciphertext

It was analysed in Sect. 6.1.2 that the TinyJAMBU mode provides strong authentication security even when nonce is repeated. We also note that an attacker always has the advantage of 2^{-16} to successfully forge a message using 2^{-48} random trials. Thus an adversary has advantage of less than 2^{-15} to forge a message successfully when less than 2^{50} bytes of data are authenticated using the same key and the nonce is repeated.

In the above analysis, it is assumed that the permutation is perfect. Here we analyse the forgery attacks for the concrete permutation. When an adversary introduces a difference in a plaintext block M_i or ciphertext block C_i, the difference introduces input difference at $s_{96\cdots127}$ of the permutation P_n, this differential probability in P_{448} is at most 2^{-55} (according to Table 6.1). In our MILP analysis, we noticed that the optimal differential always starts from the difference with hamming weight 1. From this observation, we are able to compute the differential for more rounds using MILP. The differential probability in P_{576} is at most 2^{-77}.

In TinyJAMBU, at least 1024 rounds are used to encrypt a plaintext block, so the differential forgery attack on plaintext/ciphertext succeeds with probability much smaller than 2^{-64}.

6.4 Key Recovery Attacks

6.4.1 Differential cryptanalysis

When nonce is misused, a difference can be injected into the state at $s_{96\cdots127}$ through a plaintext block, then the output difference can be observed in the next ciphertext block.

According to Table 6.1, the maximum differential probability is 2^{-55} after 448 rounds. In our MILP analysis, we noticed that the optimal differential always starts from the difference with hamming weight 1. From this observation, we are able to compute the differential for more rounds using MILP. The differential probability in P_{576} is at most 2^{-77}.

In TinyJAMBU, at least 1024 rounds are used to encrypt 32-bit message block. The differential probability for 1024 rounds is much smaller than 2^{-32}, we thus believe that it is impossible to recover the key of TinyJAMBU using the differential cryptanalysis.

6.4.2 Linear cryptanalysis

When nonce is misused, an attacker can try to find the linear relation between the input $s_{96\cdots127}$ and the output $s_{64\cdots95}$ of P_n.

According to Table 6.4, the linear bias is at most 2^{-27} for 448 rounds. This linear bias is much smaller than 2^{-16} (the message block size is 32-bit). At least 1024 rounds are used in the encryption of one plaintext block, we thus believe that it is impossible to recover the key of TinyJAMBU using the linear cryptanalsis.

6.4.3 Algebraic attacks

We experimentally tested the number of rounds for each output bit being affected by 32-bit cube for the input. The experimental results show that after 598 rounds, every output bits is affected by the 32-bit cube tester. Hence, we believe that the 1024-round encryption provides large security margin against the algebraic cryptanalysis since the message block size is 32-bit.

6.5 Slide attack

The slide attack is an effective tool to analyse the cipher with self-similarity round functions. Although TinyJAMBU permutation has the sliding property, the frame bits being added to the state will prevent the slide attack since the position of the frame bits is fixed. For example, for two related keys with slide property, the slide property between the two states gets eliminated with the introduction of Framebits of nonce.

Chapter 7

The Performance of TinyJAMBU

7.1 Hardware Performance

To evaluate the hardware performance of TinyJAMBU, we implemented TinyJAMBU-128 with a **fixed key** in VHDL using the CAESAR Hardware API [7]. The CAESAR Hardware API supports two lightweight inputs, one is 8-bit input data, another is 32-bit input data.

In our lightweight implementations of TinyJAMBU, we compute 8 and 32 rounds in one clock cycle. We synthesis our implementations with the Synopsys Design Compiler for an ASIC using 90 nm UMC technology. The results are summarized in Table 7.1. The result is the hardware cost of the whole cipher (including initialization, processing associated data, encryption and decryption, tag generation and verification), we exclude the cost of the CAESAR Hardware API which is the same for all the authenticated encryption algorithms.

Table 7.1: The hardware performance of TinyJAMBU with a **fixed** key on 90nm ASIC UMC Technology (The whole cipher is implemented, including the initialization, processing associated data, encryption and decryption, tag generation and verification)

Cipher	Rounds per clock cycle	Area (GE)	Throughput AD (Mbps)	Throughput Plaintext (Mbps)
TinyJAMBU-128	8 rounds	1352	65.16	24.6
TinyJAMBU-128	32 rounds	1674	243.3	91.2

7.2 Software Performance

In software implementation, the amount of RAM and ROM required by TinyJAMBU is expected to be very small. Some estimation is given below.

1. RAM requirements
   - 16 bytes of RAM are used to store the 128-bit state.
   - The key may be stored in RAM depending on the applications.
   - Zero RAM for key schedule since there is no key schedule in TinyJAMBU.
2. ROM requirements
   - Constants: there are four Framebits (each number 3-bit), , two round numbers (each number at most 11-bit) and five tap positions (each tap position one byte).
   - According to the reference code, the binary code of TinyJAMBU is expected to be very small since we repeatedly using the keyed permutation in the code, and the keyed permutation is implemented using only 16 lines of code. The amount of ROM being used to store the binary code is expected to be small.

We implemented TinyJAMBU in C code. We tested the speed on Intel Core i7-6550U processor running at the turbo boost speed 3.4GHz. Operating system is 64-bit Windows 10. The compiler being used is gcc 7.4.0 of Cygwin64 3.0.0, and the optimization option "-O3" is used. The performance for associated data and plaintext are given in Table 7.2 and Table 7.3, respectively.

Table 7.2: The speed (cycles per byte) of the 32-bit implementation of TinyJAMBU on Intel Core i7-6550U processor (associated data only)

AD	8B	64B	256B	1024B	4096B
TinyJAMBU-128	114.8	31.6	22.7	20.6	20.1
TinyJAMBU-192	120.9	32.4	22.8	20.6	20.1
TinyJAMBU-256	127.1	33.1	23.0	20.7	20.1

Table 7.3: The speed (cycles per byte) of the 32-bit implementation of TinyJAMBU on Intel Core i7-6550U processor (plaintext only)

Plaintext	8B	64B	256B	1024B	4096B
TinyJAMBU-128	144.7	61.6	53.1	50.1	49.6
TinyJAMBU-192	157.3	68.5	59.4	56.3	55.8
TinyJAMBU-256	169.4	75.5	65.9	62.6	61.9

Chapter 8

Features

- Lightweight, efficient authenticated encryption mode. It is lightweight since TinyJAMBU mode is based on a 128-bit keyed permutation. It is efficient since the message block size is 32 bits for 128-bit permutation.
- Strong protection of the secret key when nonce is reused. When nonce is reused in TinyJAMBU, and each key is used to process less than 2^{50} bytes of data, the secret key cannot be recovered with less than 2^{112} computations.
- Strong authentication security when nonce is reused. When nonce is reused in TinyJAMBU, and each key is used to process less than 2^{50} bytes of data, the attacker's advantage of forgery is less than 2^{-15}.
- The associated data is part of the nonce in TinyJAMBU, i.e., the combination (nonce $\|$ associated data) is the effective nonce of the cipher.
- Lightweight Permutation. The keyed permutation is based on a 128-bit nonlinear feedback shift register with only 5 taps. It is thus efficient to implement the keyed permutation in hardware.
- No key schedule in the keyed permutation, so the hardware area is significantly reduced when a constant fixed key is used in TinyJAMBU (or when a key must be stored in the devices for protecting multiple messages).
- Lightweight Input Loading. The nonce, associated data and the plaintext/ciphertext can be loaded into the state bit-by-bit when the nonlinear feedback shift register of the keyed permutation is clocked. It is thus efficient to load the input into the cipher in hardware.
- Parallel Computation. In TinyJAMBU, 32 steps can be computed in parallel. This parallel feature benefits fast hardware and software implementations.

Chapter 9

Design Rationale

- Design goal

  We aim to design a lightweight authenticated encryption algorithm which is optimized for the devices in which a secret key is stored. In the applications in which a secret key is used to protect multiple messages, it is reasonable to store a secret key in the devices, either store the key temporarily as a session key or store the key permanently as a fixed key.

  When a key is already stored in the device, the state of the cipher could be very small since we can use the keyed permutation to prevent an attacker from computing the states offline and launching the state collision attacks using the computed states.

- Design a nonce-misuse resistant cipher

  Since we aim to design a cipher for a stored key (especially a fixed key), we design the cipher to resist the key recovery attack when the attacker is able to control the nonce, associated data and plaintext.

  We also design the cipher to provide strong authentication security when nonce is misused.

  To provide better nonce-misuse resistance, the associated data also plays the role of nonce. In case when nonce is repeated due to some error, but associated data is different, the security of the cipher would not be affected by the repeated nonce.

  Feeding the plaintext twice into the cipher (one for tag generation, another for encryption using the tag as part of the nonce) can provide high security for plaintext when nonce is misused. But such cost is high for a lightweight cipher, so we do not provide strong protection of plaintext when both nonce and associated data are repeated.

- Design the TinyJABMU authenticated encryption mode

  JAMBU mode is the smallest block cipher authenticated encryption mode in the CAESAR competition. The message block size and the state size of the JAMBU mode is 0.5 and 1.5 times of the block size of the block cipher, respectively.

If we reduce the message block size in the JAMBU mode, the state size of JAMBU can be reduced significantly, and we can get better authentication security when nonce is reused. So we designed TinyJAMBU which is a small variant of JAMBU.

In the TinyJAMBU mode, the attacker cannot control the state bits easily, so when nonce is reused, TinyJAMBU mode achieves better authentication security than the Duplex mode for the same keyed permutation and the same message block size. In th Duplex mode, part of the state can be set to any arbitrary value when nonce is reused. So we do not use he Duplex mode in our design.

- Different processing of associated data and plaintext

  In TinyJAMBU, we use a strong permutation for encrypting plaintext, while we use a relatively weak permutation for processing associated data for better efficiency. The reason is that when we are processing the associated data, there is no information leakage as long as there is no state collision. The permutation for associated data only needs to resist the forgery attack, while the permutation for plaintext should resist not only the forgery attack, but also the the key recovery attack when the nonce is reused.

- Design the keyed permutation

  The keyed permutation is based on a simple nonlinear feedback shift register with only 5 taps. There are three reasons for using the nonlinear feedback shift register to update the state:

  - The hardware cost of nonlinear feedback shift register is low.
  - A number of steps of the nonlinear feedback shift registers can be computed in parallel for efficient hardware and software implementation.
  - The stream input data can be easily loaded into the state when the state gets updated.

  There is slide property in the keyed permutation based on a feedback shift register. We use the Frame Bits for each permutation to destroy the slide property.

  In the keyed permutation, there is no key schedule. It is to reduce the hardware cost of key schedule when there is an embedded key in the device.

- Design the nonlinear feedback shift register

  We use a 128-bit nonlinear feedback shift register with 5 taps to update the state of the permutation. The number of taps is not large since we aim at lightweight hardware implementation; the number of taps is not too small so that we can get reasonably good differential and linear characteristics.

Among those five taps, two taps are NANDed together (nonlinear feedback), the other three taps are for linear feedback.

To design a nonlinear feedback shift register with 5 taps, we need to choose four tap positions.

- The tap positions are chosen to ensure that 32 steps can be computed in parallel.
- There are 15 tap distances for the feedback register with 5 taps. The tap positions are chosen to ensure that most of those 15 tap distances are coprime to each other. When two tap distances are not coprime, the greatest common divisor should be small.
- After the above filtering of tap positions, we choose the tap positions with fast diffusion (measured as the number of steps being used for one state bit affecting the whole state). We keep the top 20 sets of tap positions with fast diffusion.
- After the above filtering, we tested the differential and linear property of the permutation (for 32-bit message block). We choose the tap positions that gives excellent differential and linear characteristics.

Bibliography

[1] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography – SAC 2011*, pages 320–337.

[2] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[3] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, London, UK, 1993.

[4] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Available at `https://competitions.cr.yp.to/caesar-submissions.html`

[5] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, 2009. Springer Berlin Heidelberg.

[6] Gurobi Optimizer. Available at `http://www.gurobi.com/`

[7] Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. CAESAR hardware API. *IACR Cryptology ePrint Archive*, 2016:626, 2016.

[8] Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond conventional security in sponge-based authenticated encryption modes. *Journal of Cryptology*, Jun 2018.

[9] Mitsuru Matsui. Linear Cryptanalysis Method for DES cipher. In *Advances in Cryptology–EUROCRYPT'93*, pages 386–397. Springer, 1994.

[10] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *Advances in Cryptology – EUROCRYPT'92*, pages 81–91. Springer, 1993.

[11] Nicky Mouha, Qingju Wang, Dawu Gu, Bart Preneel. Differential and linear cryptanalysis using Mixed-Integer Linear Programming. *Information security and cryptology – Inscrypt 2011*, pages 57–76.

[12] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A lightweight blockcipher-based AEAD mode of operation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):192–217, 2018.

[13] Hongjun Wu and Tao Huang. JAMBU Lightweight Authenticated Encryption Mode and AES-JAMBU. *NIST Lightweight Cryptography Workshop 2015*. Available at `https://www.nist.gov/news-events/events/2015/07/lightweight-cryptography-workshop-2015`

Appendix A

Analysis of the Probability for Equation 5.2

Here we show how to derive the maximum probability for Equation 5.2 with minimum required message blocks. The equation is given below.

$$\Pr[Coll\mathcal{E} \text{ in } \Theta_x] = \sum_{i \neq j, 1 \leq i,j \leq \theta} \min\{\frac{\beta_i \beta_j}{2^v}, 1\} \cdot \frac{1}{2^c}$$

First, we ignore the min function, so the right hand side becomes:

$$\sum_{i \neq j, 1 \leq i,j \leq \theta} \frac{\beta_i \beta_j}{2^v} \cdot \frac{1}{2^c} = \frac{1}{2^{c+v}} \sum_{i \neq j, 1 \leq i,j \leq \theta} \beta_i \beta_j$$

Let $z = \sum_{i \neq j, 1 \leq i,j \leq \theta} \beta_i \beta_j$, we have

$$\begin{aligned} z &= \sum_{i \neq j, 1 \leq i,j \leq \theta} \beta_i \beta_j \\ &\leq \sum_{i \neq j, 1 \leq i,j \leq \theta} (\frac{\beta_i^2 + \beta_j^2}{2}) \\ &= \frac{(\theta - 1)(\sum_{i=1}^{\theta} \beta_i^2)}{2}. \end{aligned}$$

Then,

$$\sum_{i=1}^{\theta} \beta_i^2 \geq \frac{2z}{\theta - 1}$$

Add $2z$ to both size, we have

$$(\sum_{i=1}^{\theta} \beta_i)^2 = \sum_{i=1}^{\theta} \beta_i^2 + 2z \geq \frac{2z}{\theta - 1} + 2z.$$

Therefore,

$$z \leq \frac{(\theta - 1)}{2\theta}(\sum_{i=1}^{\theta} \beta_i)^2. \tag{A.1}$$

The equality holds when β_i are equal for all $1 \leq i \leq \theta$.

Now we take the min function into consideration. When $\beta_i = \beta_j \geq 2^{v/2}$, $\min\{\frac{\beta_i \beta_j}{2^v}, 1\} = 1$. Then, the Equation 5.2 obtains the maximum value which is $\sum_{i \neq j, 1 \leq i,j \leq \theta} \frac{1}{2^c} = \binom{\theta}{2} \cdot \frac{1}{2^c}$.

On the other hand, when $\beta_i = \beta_j < 2^{v/2}$, $\min\{\frac{\beta_i \beta_j}{2^v}, 1\} = \frac{\beta_i \beta_j}{2^v} < 1$, the probability cannot reach the maximum value. Moreover, from inequality A.1, z is related to the $(\sum_{i=1}^{\theta} \beta_i)^2$. Any decrease in $\sum_{i=1}^{\theta} \beta_i$ will lead to the overall probability reduce even more.

Therefore, the $\beta_i = \beta_j \geq 2^{v/2}$ will obtain the maximum value of probability with least number of message blocks $\sum_{i=1}^{\theta} \beta_i$. Any decrease in the choice of $\sum_{i=1}^{\theta} \beta_i$ will decrease the probability in a squared manner.