Quartet: A Lightweight Authenticated Cipher
(v1)

Submitter: Bin Zhang
martin_zhangbin@hotmail.com, +86-13520126526
TCA, SKLCS, Institute of Software, Chinese Academy of Sciences
100190, Beijing, China

Contents

Introduction

Quartet v1 is a lightweight authenticated cipher with a 128-bit secret key and a 96-bit IV. It is oriented to be efficiently implemented in the constrained hardware environments and to have a reasonably good performance in software on various platforms. Quartet v1 is designed bo be secure in the nonce-respecting setting. So far, no attack faster than 2^{112} has been identified in the single key model.

Chapter 1

Specification

The specification of Quartet v1 is given in this charpter.

1.1 Parameters

Quartet v1[1] is a stream cipher-based authenticated encryption primitive. It has three parameters: key length, nonce length and tag length. The parameter space is as follows. The key length is 16 bytes, the public nonce length is 12 bytes and the tag length is 16-byte or 8-byte. From a 128-bit secret key K and a 96-bit public $Nonce$, or initialization vector (IV), Quartet generates the keystream of length up to 2^{64} bits.

The inputs are a public message number $Nonce$, i.e., IV, and a secret key K, a plaintext $M = (m_0, m_1, \cdots, m_{ml-1})$ of ml bytes, the associated data $A = (ad_0, ad_1, \cdots, ad_{al-1})$ of al bytes. The length of M is up to 2^{64} bits, i.e., less than or equal to 2^{61} bytes. The length of A is up to $2^{50} - 1$ bytes. There is no secret message number, i.e., the secret message number is empty. Formally, the authenticated encryption procedure is

$$\texttt{Quartet\_AE}\,(K, IV, A, M) = (C, T).$$

The output of the authenticated encryption is (C, T), where C is the ciphertext of the plaintext M and T is the authenticated tag of 16-byte or 8-byte, which authenticates both A and M. The length of the ciphertext is exactly the same as the plaintext M. Thus, the number of bytes in M plus the tag length in bytes equals to the output length in bytes.

The verification and decryption procedure takes as input the same secret key K and the public IV, the associated data A, the ciphertext C and the received authenticated tag T, and outputs the plaintext M only if the verification of the tag is correct or $\perp$ when the verification of the tag fails. Formally, this procedure can be written as

$$\texttt{Quartet\_VD}\,(K, IV, A, C, T) = \{M, \perp\}.$$

[1] We use Quartet to denote Quartet v1 hereafter.

1.2 Recommended Parameter Sets

Primary recommended parameter set of `Quartet v1`: 16-byte (128-bit) key, 12-byte (96-bit) nonce, 16-byte (128-bit) tag. Second recommended parameter set of `Quartet v1`: 16-byte (128-bit) key, 12-byte (96-bit) nonce, 8-byte (64-bit) tag.

1.3 Operations and Variables

The following operations and variables are used in the description.

- The bitwise logic AND is denoted by $\cdot$
- The bitwise exclusive OR is denoted by $\oplus$
- The bit or bit-string concatenation is denoted by $\|$
- The bitwise right shift of a 32-bit word is $\gg_{32}$
- The bitwise left rotation of a 32-bit word is $\lll_{32}$
- The bitwise right rotation of a 64-bit word is $\ggg_{64}$
- The associated data is A, which will not be encrypted or decrypted
- One byte of the associated data is ad_i
- The byte length of the associated data is al with $0 \leq al < 2^{50} - 1$
- The plaintext is M with one byte of plaintext as m_i and the 64-bit plaintext word $M_{i,8} = m_{i+7} \| m_{i+6} \| m_{i+5} \| m_{i+4} \| m_{i+3} \| m_{i+2} \| m_{i+1} \| m_i$
- The byte length of the plaintext is ml with $0 \leq ml < 2^{61}$
- The ciphertext is C with one byte of ciphertext as c_i and the 64-bit ciphertext word $C_{i,8} = c_{i+7} \| c_{i+6} \| c_{i+5} \| c_{i+4} \ \| c_{i+3} \| c_{i+2} \| c_{i+1} \| c_i$
- The authenticated tag is T of length 16-byte or 8-byte
- $K = (K_{15}, K_{14}, \cdots, K_1, K_0)$, the 128-bit secret key used in Quartet, where K_i for $0 \leq i \leq 15$ are the byte values with K_0 being the least significant byte and K_{15} being the most significant byte
- $IV = (IV_{11}, IV_{10}, \cdots, IV_1, IV_0)$, the 96-bit initialization vector IV used in Quartet, where IV_i for $0 \leq i \leq 11$ are the byte values with IV_0 being the least significant byte and IV_{11} being the most significant byte
- D_i for $0 \leq i \leq 3$ are the 8-bit constants used in Quartet
- The jth bit $(0 \leq j \leq 63)$ of a 64-bit word x_i is $x_i[j]$, where $x_i[0]$ is the least significant bit and $x_i[63]$ is the most significant bit

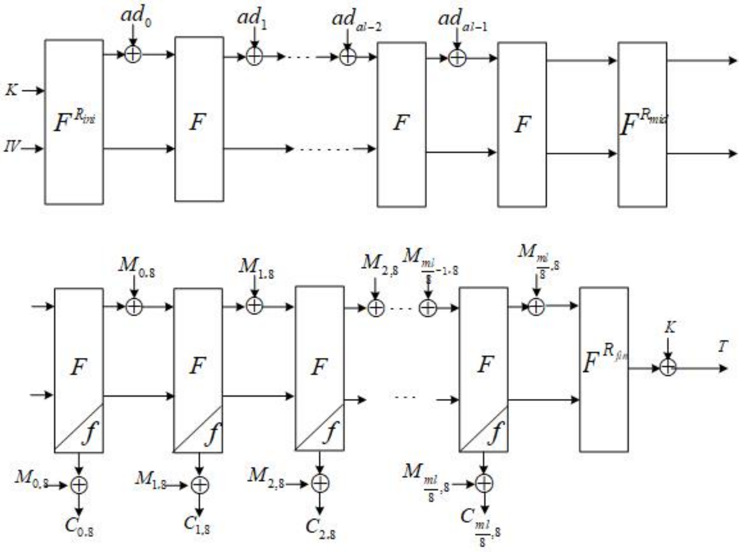

Figure 1.1: The authenticated encryption mode of Quartet v1

1.4 Mode of Operation

The mode of operation of Quartet is depicted in Figure 1.1, where F is the state updating function that operate on an internal state of 256 bits, and f is the output function that generates a 64-bit keystream word at each time instant. For a more detailed description, please see the following sections.

1.5 Description of Quartet

As depicted in Fig.1.2, there are 4 64-bit lanes involved in the algorithm: x_0, x_1, x_2 and x_3. We define 5 functions based on these 4 64-bit lanes: the χ function is the only non-linear function in the state updating of Quartet, which updates x_i by taking x_i, x_{i+1} and x_{i+2} as inputs with the index addition being the addition modulo 4; the ρ function is a linear function on one 64-bit lane, which divides the 64-bit lane into 2 32-bit words and rotates each 32-bit word left by the same number of bits; the λ function is also a linear function, but operates on one 64-bit lane x_i. It is defined by two rotation parameters $r_{i,1}$ and $r_{i,2}$ as $\lambda(x_i, r_{i,1}, r_{i,2})$; the constant addition function τ and the output function ζ which produces a 64-bit keystream word at each step. Next, we will present Quartet's components one-by-one.

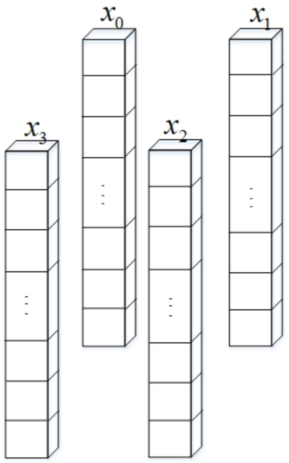

Figure 1.2: The 4 64-bit lanes in Quartet

1.5.1 The χ Function

We first look at the internal state, i.e., the 4 64-bit lanes in Fig.1.2. Based on x_0, x_1, x_2 and x_3, the only non-linear function in Quartet is the χ function defined as

$$\chi(x_i, x_{i+1}, x_{i+2}) : x_i \leftarrow x_i \oplus (x_{i+2} \oplus 1) \cdot x_{i+1}, \text{ for } 0 \le i \le 3.$$

Precisely for $0 \le i \le 3$, the corresponding state updating functions are:

$$\begin{aligned} \chi(x_0, x_1, x_2) :&\ x_0 \leftarrow x_0 \oplus (x_2 \oplus 1) \cdot x_1 \\ \chi(x_1, x_2, x_3) :&\ x_1 \leftarrow x_1 \oplus (x_3 \oplus 1) \cdot x_2 \\ \chi(x_2, x_3, x_0) :&\ x_2 \leftarrow x_2 \oplus (x_0 \oplus 1) \cdot x_3 \\ \chi(x_3, x_0, x_1) :&\ x_3 \leftarrow x_3 \oplus (x_1 \oplus 1) \cdot x_0 \ . \end{aligned}$$

Note that the χ function operates on the 64-bit lanes directly, which can be efficiently implemented in hardware/software environments. Besides, the χ function is similar to the corresponding χ function in Keccak [3], but with a different input variable order.

1.5.2 The ρ Function

The ρ function is defined as below. It divides the 64-bit lane x_i into 2 32-bit words and rotates each 32-bit word left by the same number of bits n_i.

$$\rho(x_i) : x_i \leftarrow ((x_i \gg_{32} 32) \cdot \texttt{0xffffffff} \lll_{32} n_i) \| (x_i \cdot \texttt{0xffffffff} \lll_{32} n_i),$$

where $0 \leq i \leq 3$. Precisely for $0 \leq i \leq 3$, the corresponding state updating functions are:

$$\rho(x_0) : x_0 \leftarrow ((x_0 \gg_{32} 32) \cdot \mathtt{0xffffffff} \lll_{32} n_0) \| (x_0 \cdot \mathtt{0xffffffff} \lll_{32} n_0),$$

$$\rho(x_1) : x_1 \leftarrow ((x_1 \gg_{32} 32) \cdot \mathtt{0xffffffff} \lll_{32} n_1) \| (x_1 \cdot \mathtt{0xffffffff} \lll_{32} n_1),$$

$$\rho(x_2) : x_2 \leftarrow ((x_2 \gg_{32} 32) \cdot \mathtt{0xffffffff} \lll_{32} n_2) \| (x_2 \cdot \mathtt{0xffffffff} \lll_{32} n_2),$$

$$\rho(x_3) : x_3 \leftarrow ((x_3 \gg_{32} 32) \cdot \mathtt{0xffffffff} \lll_{32} n_3) \| (x_3 \cdot \mathtt{0xffffffff} \lll_{32} n_3).$$

The left rotation constants are listed in the following table. These constants are chosen to maximize the linear diffusion effect in Quartet.

Table 1.1: The left rotation constants on 32-bit word in Quartet

variable	n_0	n_1	n_2	n_3
value	20	23	5	26

1.5.3 The λ Function

The λ function is defined as below. It is also a linear function, but operates on the 64-bit lane itself. It is used to provide diffusion within each of the 64-bit lane. The λ function operating on the lane x_i with the two rotation parameters $r_{i,1}$ and $r_{i,2}$ is denoted by $\lambda(x_i, r_{i,1}, r_{i,2})$.

$$\lambda(x_i, r_{i,1}, r_{i,2}) : x_i \leftarrow x_i \oplus (x_i \ggg_{64} r_{i,1}) \oplus (x_i \ggg_{64} r_{i,2}),$$

where $0 \leq i \leq 3$. Precisely for $0 \leq i \leq 3$, the corresponding state updating functions are:

$$\lambda(x_0, r_{0,1}, r_{0,2}) : x_0 \leftarrow x_0 \oplus (x_0 \ggg_{64} r_{0,1}) \oplus (x_0 \ggg_{64} r_{0,2}),$$

$$\lambda(x_1, r_{1,1}, r_{1,2}) : x_1 \leftarrow x_1 \oplus (x_1 \ggg_{64} r_{1,1}) \oplus (x_1 \ggg_{64} r_{1,2}),$$

$$\lambda(x_2, r_{2,1}, r_{2,2}) : x_2 \leftarrow x_2 \oplus (x_2 \ggg_{64} r_{2,1}) \oplus (x_2 \ggg_{64} r_{2,2}),$$

$$\lambda(x_3, r_{3,1}, r_{3,2}) : x_3 \leftarrow x_3 \oplus (x_3 \ggg_{64} r_{3,1}) \oplus (x_3 \ggg_{64} r_{3,2}).$$

Table 1.2: The right rotation constants on 64-bit lanes for the keystream generation in Quartet

lane	x_0	x_1	x_2	x_3
variable	$(r_{0,1}, r_{0,2})$	$(r_{1,1}, r_{1,2})$	$(r_{2,1}, r_{2,2})$	$(r_{3,1}, r_{3,2})$
value	$(1, 6)$	$(10, 17)$	$(7, 41)$	$(61, 39)$

For the different functionalities in AEAD, we set different combinations of the right rotation constants in Quartet.

Precisely, for the normal keystream generation, the combination of right rotation constants is shown in Table 1.2.

When processing the associated data A, the combination of right rotation constants is shown in Table 1.3.

Table 1.3: The right rotation constants on 64-bit lanes for processing the associated data in Quartet

lane	x_0	x_1	x_2	x_3
variable	$(r_{0,1}, r_{0,2})$	$(r_{1,1}, r_{1,2})$	$(r_{2,1}, r_{2,2})$	$(r_{3,1}, r_{3,2})$
value	$(1, 6)$	$(10, 17)$	$(7, 41)$	$(19, 28)$

For the finalization phase to produce the tag T, the combination of right rotation constants is shown in Table 1.4.

Table 1.4: The right rotation constants on 64-bit lane for the finalization and tag generation in Quartet

lane	x_0	x_1	x_2	x_3
variable	$(r_{0,1}, r_{0,2})$	$(r_{1,1}, r_{1,2})$	$(r_{2,1}, r_{2,2})$	$(r_{3,1}, r_{3,2})$
value	$(61, 39)$	$(10, 17)$	$(7, 41)$	$(19, 28)$

The purpose of these different combinations of right rotation constants is to make the domain separation in AEAD, i.e., we use different state updating functions when dealing with different kinds of functionality. Besides, all of the right rotation constants are used in ASCON [13] as well, and are known to be a permutation on the input lane itself.

1.5.4 The τ Function

The τ function is used to perform the constant addition in the initialization phase. Since there are 24 rounds of initialization in Quartet, it adds a round constant c_r to the lane x_3 in the 256-bit internal state at each round.

$$\tau(x_3) : x_3 \leftarrow x_3 \oplus c_r, \text{ for } 0 \leq r \leq 23.$$

These round constants are shown in Table 1.5, which are chosen to prevent various attacks [16, 17].

1.5.5 The Output ζ Function

The output function ζ in Quartet is defined as

$$\zeta(x_0, x_1, x_2, x_3) : z_t \leftarrow x_2 \oplus x_3 \oplus (x_1 \oplus 1) \cdot (x_0 \ggg_{64} n_z),$$

Table 1.5: The round constants in Quartet

round	0	1	2	3	4	5	6	7
c_r	0xf0	0xe1	0xd2	0xc3	0xb4	0xa5	0x96	0x87
round	8	9	10	11	12	13	14	15
c_r	0x78	0x69	0x5a	0x4b	0x13	0x26	0x0c	0x19
round	16	17	18	19	20	21	22	23
c_r	0x32	0x25	0x0a	0x15	0x2a	0x1d	0x3a	0x2b

where $n_z = 11$ is the right rotation number of the lane x_0. Quartet generates a 64-bit keystream word z_t at each time instant t.

1.5.6 One Initialization Round R_{ini}

In the initialization phase, one round R_{ini} of the state updating consists of a series of operations done on the 4 64-bit lanes (x_0, x_1, x_2, x_3).

$$\begin{aligned} R_{ini} = \tau \circ \lambda(x_2, r_{2,1}, r_{2,2}) \circ \rho(x_1) \circ \chi(x_3, x_0, x_1) \circ \\ \lambda(x_1, r_{1,1}, r_{1,2}) \circ \rho(x_0) \circ \chi(x_2, x_3, x_0) \circ \\ \lambda(x_0, r_{0,1}, r_{0,2}) \circ \rho(x_3) \circ \chi(x_1, x_2, x_3) \circ \\ \lambda(x_3, r_{3,1}, r_{3,2}) \circ \rho(x_2) \circ \chi(x_0, x_1, x_2), \end{aligned}$$

where $\circ$ is the composition operation of the different mappings.

1.5.7 One Round R in Generating keystream, Processing the Associated data and Finalization/Tag generation

In the keystream generation phase, the associated data processing phase and the finalization/tag generation phase, one round of state updating consists of the following operations done on the internal state.

$$\begin{aligned} R = \lambda(x_2, r_{2,1}, r_{2,2}) \circ \rho(x_1) \circ \chi(x_3, x_0, x_1) \circ \\ \lambda(x_1, r_{1,1}, r_{1,2}) \circ \rho(x_0) \circ \chi(x_2, x_3, x_0) \circ \\ \lambda(x_0, r_{0,1}, r_{0,2}) \circ \rho(x_3) \circ \chi(x_1, x_2, x_3) \circ \\ \lambda(x_3, r_{3,1}, r_{3,2}) \circ \rho(x_2) \circ \chi(x_0, x_1, x_2), \end{aligned}$$

where $\circ$ is the composition operation of the different mappings. Note that the combinations of the right rotation constants in the λ function are taken according to the specification in section 1.5.3.

1.5.8 The Initialization Phase

The initialization of Quartet consists of first loading the key and IV into the state, and then running the cipher for 24 steps.

The Key/IV loading scheme is as follows.

$$\begin{aligned}
x_0 &= K_7\|K_6\|K_5\|K_4\|K_3\|K_2\|K_1\|K_0,\\
x_1 &= IV_7\|IV_6\|IV_5\|IV_4\|IV_3\|IV_2\|IV_1\|IV_0,\\
x_2 &= D_3\|D_2\|D_1\|IV_{11}\|IV_{10}\|D_0\|IV_9\|IV_8,\\
x_3 &= K_{15}\|K_{14}\|K_{13}\|K_{12}\|K_{11}\|K_{10}\|K_9\|K_8.
\end{aligned}$$

where $K = K_0, \cdots, K_{15}$ is the 16 secret key bytes and $IV = IV_0, \cdots, IV_{11}$ is the 12 IV bytes, where K_0 and IV_0 are the least significant bytes, while K_{15} and IV_{15} are the most significant bytes.

The constants D_i for $0 \leq i \leq 3$ are defined as follows.

$$\begin{aligned}
D_0 &= 0xff\\
D_1 &= 0x3f\\
D_2 &= 0x00\\
D_3 &= 0x80.
\end{aligned}$$

There are 24 rounds in the initialization phase in Quartet currently, which is shown as below.

1. Load the key, IV and constants into (x_0, x_1, x_2, x_3) as specified above.
2. for $i = 0$ to 23 do

   run R_{ini} as defined in section 1.5.6 with the right rotation constants in Table 1.2
3. $x_0 \leftarrow x_0 \oplus (K_7\|K_6\|K_5\|K_4\|K_3\|K_2\|K_1\|K_0)$
4. $x_1 \leftarrow x_1 \oplus (K_{15}\|K_{14}\|K_{13}\|K_{12}\|K_{11}\|K_{10}\|K_9\|K_8)$

Note that in the initialization phase, the keystream word is not used to update the internal state in Quartet.

1.5.9 Processing the Associated Data

After the initialization, the associated data A is used to update the state.

1. for $i = 0$ to $al - 1$ do.

   absorb ad_i as $x_1 = x_1 \oplus ad_i$

   run R as defined in section 1.5.7 with the right rotation constants in Table 1.3

2. for $i = 0$ to 11 do.

    run R as defined in section 1.5.7 with the right rotation constants in Table 1.2

3. $x_3 = x_3 \oplus 1$

Note that even when there is no associated data, we still need to run the cipher for 12 steps. When we process the associated data, the keystream word is not used to update the state. We should properly take the right rotation constants according to the specification in section 1.5.3 in the λ function. Then we xor 1 bit to the lane x_3 so as to separate the associated data from the plaintext/ciphertext.

1.5.10 Processing the Plaintext

After processing the associated data, at each step of the encryption, eight plaintext bytes $M_{i,8} = m_{i+7} \| m_{i+6} \| m_{i+5} \| m_{i+4} \| m_{i+3} \| m_{i+2} \| m_{i+1} \| m_i$ are encrypted to the eight ciphertext bytes $C_{i,8} = c_{i+7} \| c_{i+6} \| c_{i+5} \| c_{i+4} \ \| c_{i+3} \| c_{i+2} \| c_{i+1} \| c_i$, and then are used to update the state. If the last plaintext block is not a full block, append a single 1 and the smallest number of 0s to pad it to 64 bits, and the padded full block is used to update the state, but only the partial block with all the padded 0s is encrypted.

1. for $i = 0$ to $\lfloor \frac{ml}{8} \rfloor - 1$ do

    compute the 64-bit keystream word z_i

    $C_{i,8} = z_i \oplus M_{i,8}$

    absorb $M_{i,8}$ as $x_0 = x_0 \oplus M_{i,8}$

    run R as defined in section 1.5.7 with the right rotation constants in Table 1.2

2. if $((ml \bmod 8) \neq 0)$ then

    compute the 64-bit keystream word $z_{\lfloor \frac{ml}{8} \rfloor}$

    $C_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8} = \texttt{Trunc}[\ z_{\lfloor \frac{ml}{8} \rfloor} \oplus M_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8}\ ]$, where

    $$M_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8} = 0x\underline{00 \cdots 00} \| m_{\lfloor \frac{ml}{8} \rfloor + ml\ mod\ 8 - 1} \| \cdots \| m_{\lfloor \frac{ml}{8} \rfloor + 0}$$

    and $\texttt{Trunc}[\cdot]$ is the truncated operation on the input argument which only keeps the least significant $ml\ mod\ 8$ bytes.

    absorb the partial plaintext block as

    $$x_0 = x_0 \oplus 0x\underline{00 \cdots 01} \| m_{\lfloor \frac{ml}{8} \rfloor + ml\ mod\ 8 - 1} \| \cdots \| m_{\lfloor \frac{ml}{8} \rfloor + 0}$$

    run R as defined in section 1.5.7 with the right rotation constants in Table 1.2

When we process the plaintext, the keystream word is not used to update the state. We should properly take the right rotation constants according to the specification in section 1.5.3 in the λ function. The cipher specification is changed so as to separate the processing of plaintext/ciphertext and the finalization.

1.5.11 Finalization and the Tag Generation

After processing all the plaintext bytes, we generate the authentication tag T.

1. for $i = 0$ to 23 do.

   run R as defined in section 1.5.7 with the right rotation constants in Table 1.4

The authentication tag T is the xored result of the secret key K and the last 2 keystream words generated from the newest internal state, i.e.,

$$T = (z_{fin+1} \| z_{fin}) \oplus (K_{15} \| K_{14} \| K_{13} \| K_{12} \| K_{11} \| K_{10} \| K_9 \| \\ K_8 \| K_7 \| K_6 \| K_5 \| K_4 \| K_3 \| K_2 \| K_1 \| K_0).$$

For the 64-bit tag, only the xored result of the least significant 64 bits of K and the last 1 keystream word generated from the newest internal state is adopted as the tag value. Note that in the finalization phase, the state is updated according to the specification in section 1.5.3 in the λ function. This is mainly used for the domain separation.

1.5.12 The Verification and Decryption

The verification and decryption procedures are very similar to the encryption and tag generation rountine. The exact values of key size, IV size, and tag size should be known to the verification and decryption processes. The decryption starts with the initialization as in section 1.5.8 and the processing of authenticated data as in section 1.5.9. Then the ciphertext is decrypted as follows. Note that if the last ciphertext block is not a full block, decrypt only the partial ciphertext block. The partial plaintext block is padded in the same way as in 1.5.10, and the padded full plaintext block is used to update the state.

1. let $ml = cl - 16$ or $ml = cl - 8$ with cl being the output byte length of `Quartet_AE`
2. for $i = 0$ to $\lfloor \frac{ml}{8} \rfloor - 1$ do

   compute the 64-bit keystream word z_i

   $M_{i,8} = z_i \oplus C_{i,8}$

   absorb $M_{i,8}$ as $x_0 = x_0 \oplus M_{i,8}$

   run R as defined in section 1.5.7 with the right rotation constants in Table 1.2

3. if $((ml \bmod 8) \neq 0)$ then

   compute the 64-bit keystream word $z_{\lfloor \frac{ml}{8} \rfloor}$

   $M_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8} = \texttt{Trunc}[\, z_{\lfloor \frac{ml}{8} \rfloor} \oplus C_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8} \,]$, where

   $$M_{\lfloor \frac{ml}{8} \rfloor, ml \bmod 8} = 0x\underline{00\cdots00}\|m_{\lfloor \frac{ml}{8} \rfloor + ml\ mod\ 8-1}\|\cdots\|m_{\lfloor \frac{ml}{8} \rfloor + 0}$$

   and $\texttt{Trunc}[\cdot]$ is the truncated operation on the input argument which only keeps the least significant *ml mod* 8 bytes.

   absorb the partial plaintext block as

   $$x_0 = x_0 \oplus 0x\underline{00\cdots01}\|m_{\lfloor \frac{ml}{8} \rfloor + ml\ mod\ 8-1}\|\cdots\|m_{\lfloor \frac{ml}{8} \rfloor + 0}$$

   run R as defined in section 1.5.7 with the right rotation constants in Table 1.2

The finalization phase in the decryption process is the same as that in the encryption process in section 1.5.11. We emphasize that if the verification fails, the decrypted plaintext and the newly generated authentication tag should not be given as output.

Chapter 2

Security Goals

In Quartet, each (key, IV) pair is used to protect only one message. If verification fails, the new tag and the decrypted ciphertext should not be given as output.

Algorithm	Encryption	Authentication (128/64-bit tag)
`Quartet`	112-bit	128/64-bit

There is no secret message number in Quartet. The public message number is a nonce, i.e., the IV. The cipher does not promise any integrity or confidentiality if the legitimate key holder uses the same nonce (IV) to encrypt two different (plaintext, associated data) pairs under the same key.

If some padding scheme is needed, Quartet adopts the same padding scheme as that in [13]. Precisely, if the plaintext block size is mr bits, the padding scheme just appends a single 1 and the smallest number of 0s to the plaintext M such that the length of the padded plaintext is a multiple of mr bits. Thus the resulting padded plaintext is split into blocks of mr bits,

$$pad_{mr}(M) = M\|\ 1\|\ 0^{mr-1-(|M| \mod mr)} \text{ if } |M| > 0.$$

Similarly, the same padding process is applied to split the associated data A into blocks of ar bits, except if the length of the associated data A is zero. In this case, no padding is applied and no associated data is processed, i.e.,

$$pad_{ar}(A) = \begin{cases} A\|\ 1\|\ 0^{ar-1-(|A| \mod ar)}, & \text{if } |A| > 0 \\ \emptyset, & \text{if } |A| = 0 \end{cases} .$$

The security claim of Quartet is the 112-bit security in the single key setting. For the forgery attacks on the authentication tag, the security level is the same as the tag size and the IV is not allowed to be re-used. If the tag verification failed, no output should be generated.

Chapter 3

Security Analysis

In this section, we will analyze the security of Quartet with respect to several attacks.

3.1 Period and Time/Memory/Data Tradeoffs

The 256-bit state of Quartet ensures that the period of the keystream is large enough for any practical applications, but the exact value of the keystream period of Quartet is difficult to predict in theory. Besides, the 256-bit size internal state also eliminates the threat of the known form of the time/memory/data tradeoff attacks [4, 5, 15] with respect to 112-bit security, when taking into account the pre-computation/memory/time/data complexities.

3.2 Linear Distinguishing Attacks

We have used the linear sequential circuit approximation (LSCA) method [14] to evaluate the strength of Quartet against linear distinguishing attacks.

There is no linear trial with a weight of less than 56 active non-linear operations found so far. Further, we have restricted the length of each keystream generated from a (key, IV) pair to be less than or equal to 2^{64} bits, thus we feel that Quartet is immune to the linear distinguishing attacks.

3.3 Differential Cryptanalysis

In order to investigate the immunity of Quartet against differential attacks, we introduce a single bit difference at each internal state position and try to trace the propagation of this difference. We gather the difference biases after several number of initialization rounds and try to distinguish it from the purely random case.

Our experiments so far showed that for the full 24 rounds of initialization and the 12 middle rounds before the first keystream bit, Quartet is non-distinguishable with the purely random case with respect to the single bit differential cryptanalysis.

3.4 Cube Attacks and Variants

Cube attacks, formally introduced by Dinur and Shamir [1, 11, 12, 21], is a generic key extraction technique exploiting the simple algebraic structure of some output bits after a reasonable size of cube summation. The success of cube attacks highly depends on the sparsity of the superpoly.

Our experiments so far showed that for the full 24 rounds of initialization and the 12 middle rounds before the first keystream bit, Quartet seems to be secure against the current forms of cube attacks.

3.5 Guess and Determine Attacks

In guess-and-determine attacks, the adversary usually guesses the content of some partial internal state, and then tries to derive the rest part of the internal state with the knowledge of the corresponding keystream segment. We have tried some simple form of guess-and-determine attacks, and have not found an attack that has a complexity less than 2^{112}.

3.6 Security of the Authenticated Mechanism

We have considered some simple forms of forgery attacks [2] against the finalization and tag generation phase, our experiments so far showed that for the full 24 rounds of finalization, Quartet seems to be secure against the simple forms of forgery attacks.

Chapter 4

Features

Quartet has the following useful features.

- New structure of stream ciphers. Quartet is the result of some efforts to parallelize the execution of the whole cipher. The challenge in this design approach is to achieve faster encryption/diffusion speed, and this problem is solved by using four parallel 64-bit lanes and mixing them in a proper way. Thus it is expensive to eliminate the difference in the internal state, and it is relatively easy to analyze the authentication security.

- The 5 ASCON linear diffusion functions are used in a different and flexible way. This feature benefits lightweight hardware implementation.

- Quartet allows parallel computation. In Quartet, a 64-bit keystream word is generated at each time instance. This parallel feature benefits high speed hardware and software implementation.

- Efficient in Hardware. Quartet has an internal state of 256-bit, smaller than that of Trivium [8], which will have a low hardware area.

- Efficient in Software. In Quartet, the operation unit is 64-bit lanes, so its software speed is reasonably fast.

- Quartet has several advantages over AES-GCM: Quartet is more hardware efficient than AES-GCM (especially for constrained hardware resource and energy consumption). On the general computing devices (no AES-NI and no polynomial computing circuits), Quartet is more efficient than AES-GCM in software. The code size of Quartet is also small.

Chapter 5

Performance

5.1 Hardware

Since Quartet only has a 256-bit internal state and consists of only 4 64-bit lanes, the hardware performance is expected to be reasonably good.

5.2 Software

We have implemented Quartet in C language. We tested the speed on an Intel Core i5-4300U 1.9GHz processor running 64-bit Windows 10. The current reference implementation reaches a speed of 8.82 cycles/byte. After further optimization, it is expected that the software speed will be faster than this reference value.

Chapter 6

Design Rationale

Quartet is designed to be efficient in the constrained hardware environments, and also efficient in software on some platforms.

In order to be efficient in hardware, we adopt 4 64-bit lanes in Quartet as Keccak [3], since it is well-known that the latter is quite efficient in hardware. In order to resist the traditional attacks (correlation attacks [6, 7, 18, 19, 20, 22] and algebraic attacks [9, 10]) on stream cipher, the state is updated in a nonlinear way. We inject the message into the internal state so that we could obtain authentication security almost for free. The challenge is that in a word-oriented stream cipher based on nonlinear state updating functions, it is difficult to trace the differential propagation in the state, especially if we want to achieve high authentication security (such as 128-bit). Our design focus is to solve this problem so that the authentication security could be relatively easily analyzed. Our solution is to use the 5 ASCON [13] linear diffusion functions in a flexible way to ensure that once there is difference in the state, the number of difference bits in the state would be sufficiently large before the difference gets eliminated. When there are difference bits in the state, the linear functions and the non-linear χ function introduces the difference noise into the state quickly to reduce the success rate of forgery attack. If an attacker intends to modify the ciphertext, the difference in the keystream bits would also affect the state through the decrypted plaintext bits. In order to make the domain separation in AEAD, different state updating functions are adopted in the different functionalities. Separating the plaintext from the associated data means that an attacker cannot use part of the plaintext bits as associated data, and vice versa. Separating the encryption/decryption from the finalization means that an attacker cannot use part of the keystream as the authentication tag.

Chapter 7

Test Vectors

Some test vectors of Quartet are provided in this chapter.

```
====================================
Length of plaintext: 1 bytes
Length of associated data: 0 bytes
The key is: 00000000000000000000000000000000
The iv is: 000000000000000000000000
The plaintext is: 01
The associated data is
The ciphertext is: 91
The tag is: 30791089f8a5eb139852fcf282a68732
The verification is successful in decryption
The decrypted plaintext is: 01
====================================
Length of plaintext: 1 bytes
Length of associated data: 1 bytes
The key is: 01000000000000000000000000000000
The iv is: 000000000000000000000000
The plaintext is: 00
The associated data is 00
The ciphertext is: 01
The tag is: a40149cddef108b485738413fa249760
The verification is successful in decryption
The decrypted plaintext is: 00
====================================
Length of plaintext: 1 bytes
Length of associated data: 1 bytes
The key is: 00000000000000000000000000000000
The iv is: 010000000000000000000000
The plaintext is: 00
The associated data is 00
The ciphertext is: 9f
The tag is: c05e5c187bcc6fccc180b6b1d09e49a8
The verification is successful in decryption
The decrypted plaintext is: 00
==========================================
Length of plaintext: 16 bytes
Length of associated data: 16 bytes
The key is: 01010101010101010101010101010101
The iv is: 010101010101010101010101
The plaintext is: 01010101010101010101010101010101
The associated data is 01010101010101010101010101010101
The ciphertext is: fff66b06d09135e514ef721e62074c20
The tag is: f5aa8feff685ad10fcf6893ea87028dc
The verification is successful in decryption
The decrypted plaintext is: 01010101010101010101010101010101
=========================================
Length of plaintext: 16 bytes
Length of associated data: 16 bytes
The key is: 000102030405060708090a0b0c0d0e0f
The iv is: 000306090c0f1215181b1e21
The plaintext is: 01010101010101010101010101010101
The associated data is 01010101010101010101010101010101
The ciphertext is: 7b3aa38807ae112e09e451dfb19cf84c
The tag is: 66793a2e9a372444a67281a7fccd0212
The verification is successful in decryption
The decrypted plaintext is: 01010101010101010101010101010101
=========================================
Length of plaintext: 73 bytes
Length of associated data: 43 bytes
The key is: 000102030405060708090a0b0c0d0e0f
The iv is: 000306090c0f1215181b1e21
The plaintext is:
00070e151c232a31383f464d545b6269
70777e858c939aa1a8afb6bdc4cbd2d9
e0e7eef5fc030a11181f262d343b4249
50575e656c737a81888f969da4abb2b9
c0c7ced5dce3eaf1f8
The associated data is
00050a0f14191e23282d32373c41464b
50555a5f64696e73787d82878c91969b
a0a5aaafb4b9bec3c8cdd2
The ciphertext is:
6bf3d2752184ea72f43c20a72703b0ba
6515360c76add92cd426bcdcedb30feb
854e1f891485b18398923c78e476b3ac
f8978bea37d1a7292e77124664913092
0b21b7c52315a613a0
```

```
The tag is: 8afdf079f00d05aa2ae2273a9491ca23
The verification is successful in decryption
The decrypted plaintext is:
00070e151c232a31383f464d545b6269
70777e858c939aa1a8afb6bdc4cbd2d9
e0e7eef5fc030a11181f262d343b4249
50575e656c737a81888f969da4abb2b9
c0c7ced5dce3eaf1f8
```

Bibliography

[1] Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium, *Fast Software Encryption-FSE'2009*, LNCS vol. 5665, Springer, Heidelberg,(2009), pp. 1-22.

[2] Agren, M., Hell, M., Johansson, T.: On hardware-oriented message authentication with applications towards RFID, in Proceedings of the 2011 Workshop on Lightweight Security and Privacy: Devices, Protocols, and Applications, E. Savas, A. A. Selcuk, and U. Uludag, Eds., pp. 26-33. (2011).

[3] Bertoni, G., Daemen, J., Peeters, M. and Van Assche G.: The Keccak reference, available at `http://keccak.noekeon.org/Keccak-reference-3.0.pdf.`

[4] Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data tradeoffs for stream ciphers, *Advances in Cryptology-ASIACRYPT'2000*, LNCS vol. 1976, Springer, Heidelberg,(2000), pp. 1-13.

[5] Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. *Fast Software Encryption–FSE 2000*, LNCS, vol. 1978, pp. 1-18, Springer, Heidelberg, 2001.

[6] Canteaut A. and Trabbia. M., Improved fast correlation attacks using parity-check equations of weight 4 and 5. In Preneel B. (eds), *Advances in Cryptology–EUROCRYPT 2000*, LNCS vol. 1807, pp. 573–588, Springer Berlin Heidelberg, 2000.

[7] Chose P., Joux A. and Mitton M., Fast correlation attacks: an algorithmic point of view. In Knudsen L. R. (eds), *Advances in Cryptology–EUROCRYPT 2002*. LNCS vol. 2332, Springer Berlin Heidelberg, pp. 209-221, 2002.

[8] Christophe De Cannière, Preneel, B: Trivium, *New Stream Cipher Designs–The eSTREAM Finalists: 2008*, LNCS vol. 4986, Springer, Heidelberg,(2008), pp. 244-266.

[9] Courtois N. T., Weier. W., Algebraic attacks on stream ciphers with linear feedback, In Biham E. (eds), *Advances in Cryptology–EUROCRYPT'2003*, LNCS vol.2656, Springer-Verlag, pp. 345–359, 2003.

[10] Courtois N. T., Fast algebraic attacks on stream ciphers with linear feedback, In Boneh D. (eds), *Advances in Cryptology–CRYPTO'2003*, LNCS vol.2729, Springer-Verlag, pp. 176–194, 2003.

[11] Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials, *Advances in Cryptology-EUROCRYPT'2009*, LNCS vol. 5479, Springer, Heidelberg,(2009), pp. 278-299.

[12] Dinur, I., Shamir, A.: Breaking Grain-128 with dynamic cube attacks, *Fast Software Encryption-FSE'2011*, LNCS vol. 6733, Springer, Heidelberg,(2011), pp. 167-187.

[13] Dobraunig C., Eichlseder M., Mendel F. and Schläffer, M.: ASCON v1.2, Submission to the Caesar Competition, `https://ascon.iaik.tugraz.at/index.html`

[14] Golić Jovan Dj.: Correlation properties of a general binary combiner with memory, *Journal of Cryptology*, vol.9, Springer-Verlag. pp. 111-126, (1996).

[15] Golić Jovan Dj.: Cryptanalysis of alleged A5 stream cipher, *Advances in Cryptology-EUROCRYPT'1997*, LNCS vol.1233, Springer-Verlag. pp. 239-255, (1997).

[16] Khovratovich D., Nikolić I.: Rotational cryptanalysis of ARX. The 17th international conference on Fast Software Encryption-FSE'2010. pp. 333-346. LNCS vol.6147, Springer-Verlag (2010)

[17] Pawe M., Josef P., and Marian S.: Rotational cryptanalysis of round-reduced Keccak. *Fast Software Encryption-FSE'2013*, LNCS vol.8424, Springer-Verlag. pp. 241-262, (2014).

[18] Willi, M., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3): 159-176, 1989.

[19] Johansson T. and Jönsson F., Improved fast correlation attacks on stream ciphers via convolutional codes, In Stern J. (eds), editor, *Advances in Cryptology–EUROCRYPT'99*, LNCS vol. 1592, pp. 347–362, Springer Berlin / Heidelberg, 1999.

[20] Johansson T. and Jönsson F., Fast correlation attacks through reconstruction of linear polynomials, In Bellare M. (eds), *Advances in Cryptology–CRYPTO 2000*, LNCS vol. 1880, pp. 300-315, 2000.

[21] Todo Y., Structural Evaluation by Generalzed Integral Property, In Oswald E. and Fischlin M. (eds), *Advances in Cryptology–EUROCRYPT'2015*, LNCS vol. 9056, pp. 287-314, 2015.

[22] Zhang B., Xu C., and Meier W., Fast correlation attacks over extension fields, Large-unit linear approximation and Cryptanalysis of SNOW 2.0, In Gennaro H. and Robshaw M. (eds), *Advances in Cryptology–CRYPTO'2015*, LNCS vol. 9215, pp. 643-662, 2015.