Microprocessors and Microsystems 45 (2016) 45–55

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

QTL: A new ultra-lightweight block cipher

Lang Li*, Botao Liu, Hui Wang

Department of Computer Science, Hengyang Normal University, Hengyang, 421002, China

ARTICLE INFO

Article history:
Received 6 January 2015
Revised 14 January 2016
Accepted 11 March 2016
Available online 21 March 2016

Keywords:
Lightweight block cipher
Feistel-type structures
Generalized Feistel network
SPNs

ABSTRACT

We propose a new ultra-lightweight block cipher, QTL. The 64 bits block cipher QTL supports 64 and 128 bits keys. To solve the slow diffusion of the traditional Feistel-type structures we have used a new variant of generalized Feistel network structure in design of the QTL. Traditional Feistel-type structures change only half of block messages in an iterative round, but our structure overcomes this disadvantage and changes all block messages. Thus, our structure has the fast diffusion of the Substitution Permutation Networks (SPNs) structures, which improves the security of lightweight block cipher in Feistel-type structures. Moreover, QTL algorithm has the same encryption and decryption processes, so it will occupy less area in resource-constrained applications. Furthermore, to reduce the cost of energy consumption in hardware implementation of the cipher while maintaining security, we decide not to use a key schedule. We show that QTL offers an adequate security level against classic analyses. Our hardware implementation for the 64 and the 128 bits keys modes only require 1025.52 and 1206.52 gate equivalents, respectively. QTL achieves high security and compact implementation in hardware. QTL is one of the most competitive ultra-lightweight block ciphers, which is suitable for extremely constrained devices.

1. Introduction

1.1. Background and motivation

In cryptography, a block cipher is a deterministic algorithm operating on fixed-length groups of bits. Essentially, by limited iteration with block cipher, fixed plaintext is converted to ciphertext. Block cipher algorithm is fast and easy to standardize and facilitate the software and hardware implementations. Hence, in information security block cipher has core values and is a main constrained environment cipher. Recently, with the development of resource-constrained embedded systems, embedded block ciphers are essential primitives for cryptographic applications, such as wireless sensor networks (WSNs), RFID tags, and sensor nodes. These devices manufactures have lower maintenance cost, higher network robustness, stronger self-organization and broader applicability features, which have become a key part of the networking industry. But WSNs and RFID tags are based on wireless networks to transmit information, and this information can easily be acquired, interposed or even destroyed. Under resource-constrained environment, the block cipher has extremely restricted hardware resources. A basic RFID tag chip area generally needs around 1000 to 10,000 gate equivalents (GE) [1]. However, the block cipher algorithm only takes RFID tag chip area around 200 to 2000 GE. Traditional block ciphers use a lot of GE in hardware implementation. For example, low-cost implementation of the AES [2] requires around 3600 GE, which is far more than 2000 GE. Thus, traditional block ciphers are not suitable for resource-constrained environment equipment. In this case, low-cost and efficient lightweight block ciphers became a hot topic of the research.

Nowadays, in the area of lightweight block cipher some of the lightweight block ciphers are proposed, such as PRESENT [3], LBlock [4], TWINE[5], KLEIN [6], MIBS [7], LED [8], PRINCE [9], Piccolo [10], ITUbee [11], EPCBC [12], PRINTcipher [13] and RECTANGLE[14]. Particularly, PRESENT is the most representative of the block cipher. Structures of these lightweight ciphers as like traditional block ciphers are generally developed into two main classical structures: SPNs and Feistel-type structures.

On the one hand, PRESENT was proposed in 2007, based on a SPN, and its hardware requirement is less than 2000 GE. But PRESENT is susceptible to threats of side-channel attacks [15, 16]. Recently, LED was designed for RFID tags, which is also an instantiation of a SPN. It achieves remarkably compact hardware implementation, and explores the role of a non-existent key schedule. KLEIN also adopts a SPN. KLEIN and LED balance the tradeoffs between hardware and software implementations [17].

On the other hand, MIBS and LBock are designed for low resource devices, and ITUbee is a software oriented lightweight block cipher. Those are an instantiation of the Feistel networks. Piccolo is an instantiation of the generalized Feistel networks (GFNs), which

* Corresponding author at: 16 henghua Rd.,Zhuihui District, Hengyang, Hunan 421002, China, Tel:+86 15873438955
E-mail address: lilang911@126.com (L. Li).

http://dx.doi.org/10.1016/j.micpro.2016.03.011
0141-9331/

has low-cost in hardware implementation, but the Biclique cryptanalysis can reduce security of Piccolo [18].

The SPN structure is developed using round function on the whole data block [19]. The slow diffusion of the traditional Feistel-type structures will result in some security problems. Therefore, to solve these problems the ciphers in traditional Feistel-type structures commonly demand a lot of rounds in contrast to the ciphers based on SPNs; thus, this increases energy consumption. Nevertheless, compared to SPNs, the traditional Feistel-type structures have more features. Firstly, it has a small and simple round function. Secondly, it has the same program for encryption and decryption processes to reduce decryption implementation cost. Thus, the traditional Feistel-type structures are more suitable for designing lightweight block ciphers.

In this paper, we aim to address the slow diffusion of the lightweight block cipher in traditional Feistel-type structures. It is significant to design a new variant of generalized Feistel network structures with the fast diffusion of the SPNs.

1.2. Our contributions

We propose a new ultra-lightweight block cipher called QTL, which is optimized for extremely resource-constrained devices. QTL is a new variant of generalized Feistel network structure algorithm, which supports 64 bits block with 64 or 128 bits keys. QTL has the fast diffusion of the SPNs, which improves the security of lightweight block cipher in Feistel-type structures. QTL has many numbers of active S-boxes on certain bounds during encryption process. We decide not to use a key schedule to reduce the cost of energy consumption in hardware implementation of the cipher. The ciphers without key schedule are from SPNs structure such as LED and PRINCE, and the only cipher in Feistel networks structure without key schedule is ITUbee.

However, if a lightweight block cipher in Feistel-type structures has no key schedule, it is susceptible to related-key attacks [11]. We have analyzed related-key attacks in Section 4.3. Meanwhile, we demonstrate that QTL offers an adequate security level against classic analyses. Moreover, the area for the 64 and 128 bits keys modes only require 1025.52 and 1206.52 GE. Therefore, QTL achieves high security and compact implementation in hardware.

The organization of the rest paper is as the following. We describe the specification of QTL in Section 2, explain the design rationale for QTL in Section 3, present the result of security analysis in Section 4 along with the performance results of hardware implementation in Section 5, and conclude the paper in Section 6.

2. Specification of QTL

QTL is a 64 bits lightweight block cipher, and the key lengths are 64 bits or 128 bits. The 64 and 128 bits keys modes are referred as QTL-64 and QTL-128, and QTL is a new variant of generalized Feistel network structure block cipher. We choose the number of iterative rounds NR as 16/20 for QTL-64/QTL-128. We will describe QTL encryption algorithm and decryption algorithm in details.

2.1. Notations

We use the following notations in the paper:

X	64 bits plaintext
Y	64 bits ciphertext
$K_{(64)}$	64 bits master key
$K_{(128)}$	128 bits master key
F_1	F_1-function
F_2	F_2-function

Algorithm 1
The encryption routine of QTL.

ENC_{NR}
Input: $X_{(64)}$, $K_{0(16)}$, $K_{1(16)}$, $K_{2(16)}$, $K_{3(16)}$
Output: $Y_{(64)}$
1:$X_{(64)} \rightarrow X_{0(16)} \| X_{1(16)} \| X_{2(16)} \| X_{3(16)}$
2:for i=1 to (NR-1) do the following
3: for j=1 to 2 do the following
4: if(j= =1) then
5: $X_{1(16)} \leftarrow X_{1(16)} \oplus F_1(K_{0(16)}, X_{0(16)})$, $X_{3(16)} \leftarrow X_{3(16)} \oplus F_2(K_{1(16)}, X_{2(16)})$
6: $X_{0(16)} \| X_{1(16)} \| X_{2(16)} \| X_{3(16)} \leftarrow X_{1(16)} \| X_{0(16)} \| X_{3(16)} \| X_{2(16)}$
7: end if
8: if(j= =2) then
9: $X_{1(16)} \leftarrow X_{1(16)} \oplus F_1(K_{2(16)}, X_{0(16)})$, $X_{3(16)} \leftarrow X_{3(16)} \oplus F_2(K_{3(16)}, X_{2(16)})$
10: end if
11: end for
12: $X_{0(16)} \| X_{1(16)} \| X_{2(16)} \| X_{3(16)} \leftarrow RT(X_{2(16)} \| X_{1(16)} \| X_{0(16)} \| X_{3(16)})$
13:end for
14:for j=1 to 2 do the following
15: if(j= =1) then
16: $X_{1(16)} \leftarrow X_{1(16)} \oplus F_1(K_{0(16)}, X_{0(16)})$, $X_{3(16)} \leftarrow X_{3(16)} \oplus F_2(K_{1(16)}, X_{2(16)})$
17: $X_{0(16)} \| X_{1(16)} \| X_{2(16)} \| X_{3(16)} \leftarrow X_{1(16)} \| X_{0(16)} \| X_{3(16)} \| X_{2(16)}$
18: end if
19: if(j= =2) then
20: $X_{1(16)} \leftarrow X_{1(16)} \oplus F_1(K_{2(16)}, X_{0(16)})$, $X_{3(16)} \leftarrow X_{3(16)} \oplus F_2(K_{3(16)}, X_{2(16)})$
21: end if
22:end for
23:$Y_{0(16)} \| Y_{1(16)} \| Y_{2(16)} \| Y_{3(16)} \leftarrow X_{0(16)} \| X_{1(16)} \| X_{2(16)} \| X_{3(16)}$
24:$Y_{(64)} \leftarrow Y_{0(16)} \| Y_{1(16)} \| Y_{2(16)} \| Y_{3(16)}$
25:**Return** $Y_{(64)}$

AC	AddConstants
P	Permutation operate on 16 bits
S-box	4×4 S-box
S	S-box layer
RT	Round transposing
$\|\|$	Concatenation of two binary strings
$\oplus$	Bitwise exclusive-OR operation
CON	Round constants

2.2. Encryption algorithm

The encryption algorithm of QTL has the number of iterative rounds NR. Fig. 1 illustrates the encryption procedure. Then we will explain encryption algorithm steps of QTL in details as below.

The data processing part of QTL consists of NR rounds. ENC_{NR} inputs a 64 bits plaintext data $X \in \{0,1\}^{64}$, four 16 bits round sub-keys $K_{i(16)}$ $(0 \le i \le 3)$, and outputs a 64 bits ciphertext data $Y \in \{0,1\}^{64}$ (see Algorithm 1). Specifically, the AddRoundKey illustrates the structure of round sub-keys $K_{i(16)}$ in details. ENC_{NR} is defined as below:

$$ENC_{NR}: \left\{ \begin{array}{c} \{0,1\}^{64} \times \{\{0,1\}^{16}\}^4 \rightarrow \{0,1\}^{64} \\ (X_{(64)}, K_{0(16)}, K_{1(16)}, K_{2(16)}, K_{3(16)}) \rightarrow Y_{(64)} \end{array} \right\}$$

where F_1 and F_2 are 16 bits F-functions, and RT is a 64 bits permutation defined in the following sections. **F-functions.** Round function of QTL consists of two different 16 bits F-functions F_1 and F_2. F_1 and F_2 have the same composition of the structure. We define the input and output of F_1 and F_2 as follows:

$$\left\{ \begin{array}{c} \{0,1\}^{16} \rightarrow \{0,1\}^{16} \\ F_1 : (X_i, K_i)| \rightarrow Y_i = S_1(P(S_1(X_i \oplus CON_1 \oplus K_j))) \quad (i = 0 \text{ or } 1) \quad (j = 0 \text{ or} 2) \\ F_2 : (X_i, K_i)| \rightarrow Y_i = S_2(P(S_2(X_i \oplus CON_2 \oplus K_j))) \quad (i = 2 \text{ or } 3) \quad (j = 1 \text{ or } 3) \end{array} \right\}$$

The F_1 and F_2 consist of AddConstants and AddRoundKey where the S_1s and S_2s are separated by a P. And the F-function process is illustrated as: AddConstants → AddRoundKey → S-box layer → P permutation layer → S-box layer (see Fig. 2).

The AddConstants updates a 16 bits data $X_{(16)}$ as follows:

The round constants (CON) consist of CON_1 and CON_2. We define the nth for $(0 \le n \le NR)$ round constants as the CON_1^n and

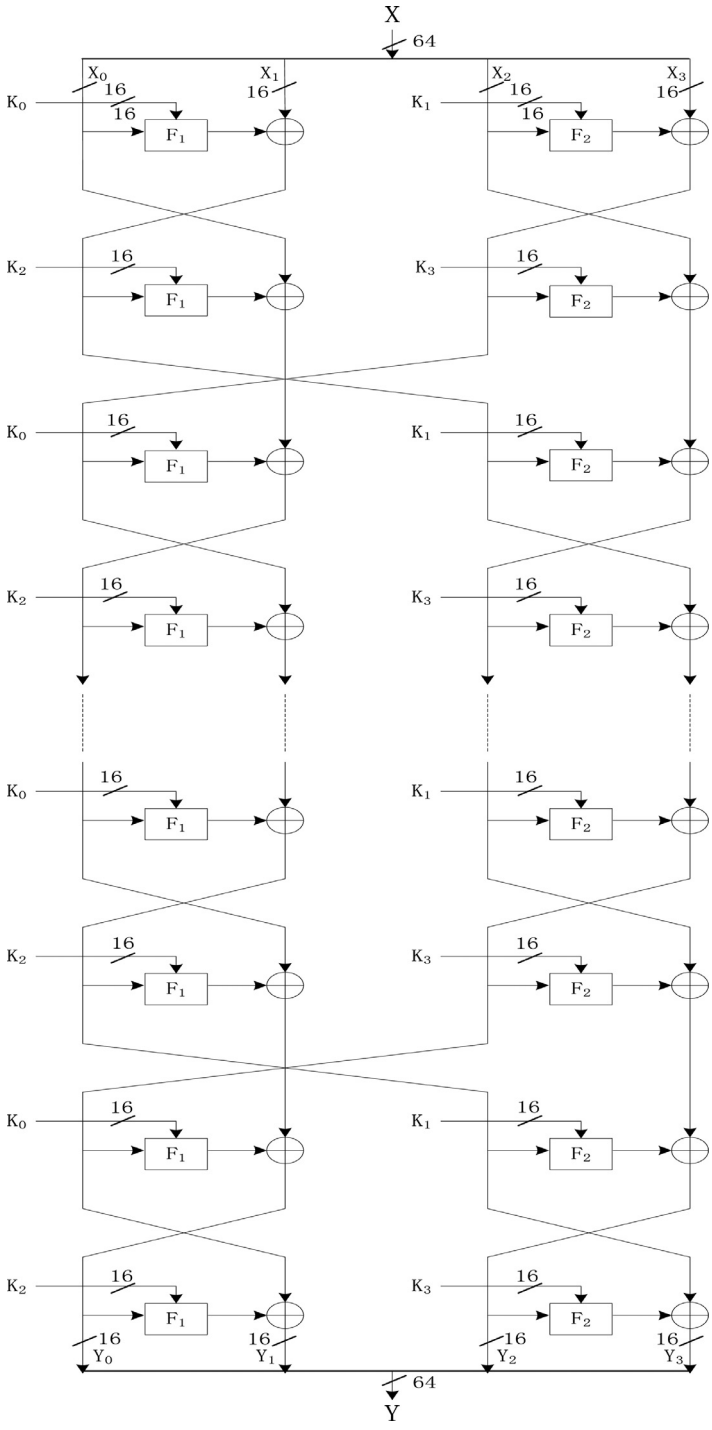


Fig. 1. Encryption procedure of QTL.

CON_2^n, respectively. Table 1 shows the CON_1^n and CON_2^n of QTL-64. Table 2 shows the CON_1^n and CON_2^n of QTL-128.

AddConstants is calculated by adding 8 bits round constant to the leftmost 8 bits of$X_{(16)}$.

The AddRoundKey updates a 16 bits data $X_{(16)}$ as follows:

QTL can take keys of either 64 or 128 bits. To start with, we introduce the 64 bits master key$K_{(64)}$, which is denoted as$K_{(64)} = k_{63}||k_{62}||k_{61}.....k_2||k_1||k_0$. The round sub-keys $K_{i(16)}$for $(0 \leq i \leq 3)$are considered as 16 bits and denoted as

$$K_{0(16)} = k_{63}||k_{62}||k_{61}.....k_{50}||k_{49}||k_{48},$$

$$K_{1(16)} = k_{47}||k_{46}||k_{45}.....k_{34}||k_{33}||k_{32},$$

$$K_{2(16)} = k_{31}||k_{30}||k_{29}.....k_{18}||k_{17}||k_{16},$$

$$K_{3(16)} = k_{15}||k_{14}||k_{13}.....k_2||k_1||k_0.$$

Then we introduce the 128 bits master key$K_{(128)}$, which is denoted as $K_{(128)} = k_{127}||k_{126}||k_{125}.....k_2||k_1||k_0$. The master 128 bits key is divided into two numbers of round sub-keys in the QTL-128 operation process. We define the $K^1_{(64)}$ and $K^2_{(64)}$ as the left and the right half of $K_{(128)}$, respectively. Thus, we develop an equation

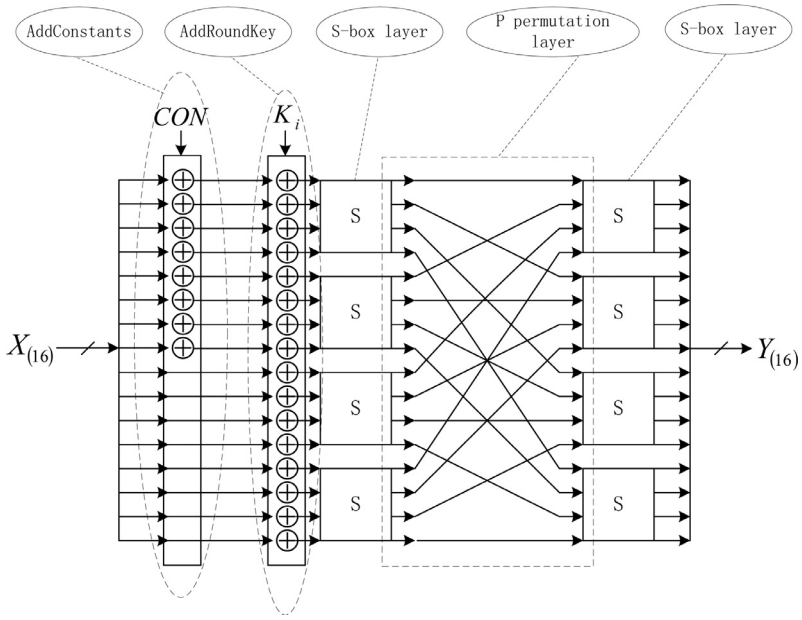


Fig. 2. F-Function of QTL.

Table 1
The CON_1^n and CON_2^n of QTL-64 in hexadecimal form.

n	0	1	2	3	4	5	6	7
CON_1^n	00	01	02	03	04	05	06	07
CON_2^n	10	11	12	13	14	15	16	17
n	8	9	10	11	12	13	14	15
CON_1^n	08	09	0A	0B	0C	0D	0E	0F
CON_2^n	18	19	1A	1B	1C	1D	1E	1F

Table 2
The CON_1^n and CON_2^n of QTL-128 in hexadecimal form.

n	0	1	2	3	4	5	6	7	8	9
CON_1^n	00	01	02	03	04	05	06	07	08	09
CON_2^n	14	15	16	17	18	19	1A	1B	1C	1D
n	10	11	12	13	14	15	16	17	18	19
CON_{1^n}	0A	0B	0C	0D	0E	0F	10	11	12	13
CON_{2^n}	1E	1F	20	21	22	23	24	25	26	27

as$K_{(128)} = K^1_{(64)}||K^2_{(64)}$. The $K^1_{(64)}$ is the odd number of round sub-keys, and the $K^2_{(64)}$ is the even number of round sub-keys.

The odd number of round sub-keys $K_{i(16)}$ of $K^1_{(64)}$ is denoted as

$$K_{0(16)} = k_{127}||k_{126}||k_{125}.....k_{114}||k_{113}||k_{112},$$

$$K_{1(16)} = k_{111}||k_{110}||k_{109}.....k_{98}||k_{97}||k_{96},$$

$$K_{2(16)} = k_{95}||k_{94}||k_{93}.....k_{82}||k_{81}||k_{80},$$

$$K_{3(16)} = k_{79}||k_{78}||k_{77}.....k_{66}||k_{65}||k_{64}.$$

The even number of round sub-keys $K_{i(16)}$ of $K^2_{(64)}$ is denoted as

$$K_{0(16)} = k_{63}||k_{62}||k_{61}.....k_{50}||k_{49}||k_{48},$$

$$K_{1(16)} = k_{47}||k_{46}||k_{45}.....k_{34}||k_{33}||k_{32},$$

$$K_{2(16)} = k_{31}||k_{30}||k_{29}.....k_{18}||k_{17}||k_{16},$$

Table 3
4 bits bijective S-box S_1 in hexadecimal form.

X	0	1	2	3	4	5	6	7
$S_1[x]$	C	5	6	B	9	0	A	D
X	8	9	A	B	C	D	E	F
$S_1[x]$	3	E	F	8	4	7	1	2

Table 4
4 bits bijective S-box S_2 in hexadecimal form.

X	0	1	2	3	4	5	6	7
$S_2[x]$	4	F	3	8	D	A	C	0
X	8	9	A	B	C	D	E	F
$S_2[x]$	B	5	7	E	2	6	1	9

$$K_{3(16)} = k_{15}||k_{14}||k_{13}.....k_2||k_1||k_0.$$

Given 16 bits round sub-key $K_{(16)}$ and 16 bits data $X_{(16)}$ input, AddRoundKey consists of the $X_{(16)} \rightarrow X_{(16)} \oplus K_{(16)}$ operation.

The S_1 and S_2 are different nonlinear 4 bits S-boxes. Table 3 shows the input and output values of S_1. Table 4 shows the input and output values of S_2.

The S-box layers update a 16 bits data $X_{(16)}$ as follows:

$$S_1, S_2 : \left\{ \begin{array}{l} (x_{0(4)}, x_{1(4)}, x_{2(4)}, x_{3(4)}) \leftarrow (S_1(x_{0(4)}), S_1(x_{1(4)}), \\ \quad S_1(x_{2(4)}), S_1(x_{3(4)})) \\ (x_{0(4)}, x_{1(4)}, x_{2(4)}, x_{3(4)}) \leftarrow (S_2(x_{0(4)}), \\ \quad S_2(x_{1(4)}), S_2(x_{2(4)}), S_2(x_{3(4)})) \end{array} \right\}$$

where $X_{(16)} = x_{0(4)}||x_{1(4)}||x_{2(4)}||x_{3(4)}$. Then the P permutation layer updates a 16 bits data $X_{(16)}$ as follows:

The bit permutation used in F-Function F_1 and F_2 is given in the following Table 5, and the bit permutation can be expressed as$P$: $\{0, 1\}^{16} \rightarrow \{0, 1\}^{16}$. Bit j for$(0 \le j \le 15)$of $X_{i(16)}$ for $(0 \le i \le 3)$ moves to bit position$P(j)$.

Round transposing. The Round transposing RT: $\{0, 1\}^{64} \rightarrow \{0, 1\}^{64}$divides a 64–bit input $X_{(64)}$ into 16 bits $X_{i(16)}(0 \le i \le 3)$, so we permute them in the manner of

$$\left(X_{0(16)}||X_{1(16)}||X_{2(16)}||X_{3(16)}\right) \rightarrow \left(X_{2(16)}||X_{1(16)}||X_{0(16)}||X_{3(16)}\right).$$

Table 5
The bit permutation.

J	0	1	2	3	4	5	6	7
P(j)	0	4	8	12	1	5	9	13
J	8	9	10	11	12	13	14	15
P(j)	2	6	10	14	3	7	11	15

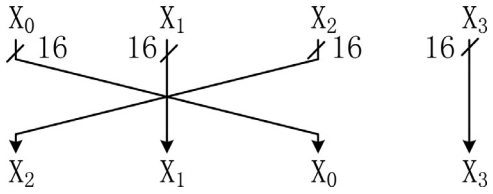


Fig. 3. Round transposing of QTL.

Finally, the Round transposing concatenates $(X_{2(16)}||X_{1(16)}||X_{0(16)}||X_{3(16)})$ into 64 bits $X_{(64)}$ (see Fig. 3).

2.3. Decryption algorithm

The decryption algorithm of QTL is the same as the encryption algorithm. But in the decryption algorithm the round constants and round sub-keys are used in a reverse order. In the following we describe the round constants and round sub-keys of changing process in details.

We define the decrypted round constants as *CON′*. And the value of QTL-64 and QTL-128 decrypted round constants are shown in Table 12 and Table 13 of Appendix III, respectively.

We define the round sub-keys of decryption as $K'_{i(16)}$ for $(0 \le i \le 3)$. In the decryption of QTL-64, we show the $K'_{i(16)}$ as

$$K'_{0(16)} \leftarrow K_{2(16)}, K'_{1(16)} \leftarrow K_{3(16)}, K'_{2(16)} \leftarrow K_{0(16)}, K'_{3(16)} \leftarrow K_{1(16)}.$$

In the decryption of QTL-128, we define that the $K^{1\prime}_{(64)}$ is the odd number of round sub-keys of decryption, and the $K^{2\prime}_{(64)}$ is the even number of round sub-keys of decryption:

$$K^{1\prime}_{(64)} \leftarrow K^{2}_{(64)}, K^{2\prime}_{(64)} \leftarrow K^{1}_{(64)}.$$

Then, in the $K^{1\prime}_{(64)}$ and $K^{2\prime}_{(64)}$, we show the $K'_{i(16)}$ as

$$K'_{0(16)} \leftarrow K_{2(16)}, K'_{1(16)} \leftarrow K_{3(16)}, K'_{2(16)} \leftarrow K_{0(16)}, K'_{3(16)} \leftarrow K_{1(16)}.$$

Appendix I shows the test vectors of QTL-64 and QTL-128.

3. Design rationale

3.1. Structure

QTL supports 64 bits block to fit standard applications, and 64 and 128 bits keys to achieve moderate security levels. To address the slow diffusion of the lightweight block cipher in traditional Feistel-type structures, we design a new variant of generalized Feistel network structure with the fast diffusion of the SPNs. Traditional Feistel-type structures change only half of block messages in an iterative round, but our structure overcomes this disadvantage and changes all block messages. The other advantage of this structure is that it enables a minimum differential characteristic probability and the best linear characteristic approximation, where the cipher for the differential analysis and linear analysis is secure. Hence, the structure of QTL improves security of lightweight block cipher in Feistel-type structures. When designing the structure of a lightweight block cipher, we take into account the security and efficient implementation. Lightweight block cipher is different from traditional block cipher in regards to its simple cipher structure and small area in hardware. The elimination of the key schedule reduces the energy consumption and area requirement. The usage of a generalized Feistel structure without key schedule provides the security against related key attacks and makes QTL distinguished from previous proposed ciphers. Furthermore, we choose Feistel-type structure, because it enables us to use the same program for encryption and decryption processes in resource-constrained applications, which leads to less area requirement.

3.2. F-function

We have used a linear permutation layer between two S-boxes layers to avoid using two consecutive S-boxes layers in the F-function. The design of F-function is characterized in an extremely light and security. The round constant does not need memory resource to save. 16 bits round sub-keys $K_{(16)}$ and input 16 bits $X_{(16)}$ carry out a *XOR* operation, which do not need to cost a lot of hardware resources. For S-box, we use two different 4×4 S-boxes and the P permutation between the same two S-boxes. Therefore, the design of F-function is suitable for an ultra-lightweight block cipher. We also obtain 2^{-2} and 2^{-2} as maximum differential probability (MDP) and maximum linear probability (MLP) of a S-box, respectively. Then, we use more S-boxes in F-function. Hence, the design of F-function is sufficiently secure, since QTL has enough differentially and linearly active S-boxes over a certain number of rounds.

3.3. Round transposing and P permutation layer

In order to improve the diffusion of QTL property and reduce linear layer the hardware implementation cost, we apply two highly efficient structures of linear layers: the P permutation layer and the Round transposing. The P permutation layer is bit permutations of 16 bits, which is extremely easy to implement in hardware. Furthermore, it can improve the security against cryptanalysis. The Round transposing is a 16 bits word-wise moving between rounds instead of a 16 bits word based cyclic shift used in the standard GFNs. Therefore, the structure of the Round transposing is special and efficient, which can be implemented in software and hardware constrained environments. It also can improve the security against cryptanalysis. For the Round transposing, we take full consideration of its involution property as the structure of a cipher, which does not destroy the encryption process and is identical to the decryption process. Specifically, under safe conditions, the two linear layers are implemented without any cost in hardware.

3.4. S-box layer

S-box layer denotes the non-linear layer of QTL. For S-box layer, we carefully consider its security and efficient implementation (such as cost and area requirement). Taking into account the cipher applies to resource-constrained devices such as RFID tags and WSNs, we pursue efficient hardware performance and choose to use 4×4 S-boxes. At present, a 4×4 S-box layer not only provides appropriate security but also achieves low complexity in hardware [7]. 4×4 S-boxes need only about 22 GE when implemented in hardware. Although 8×8 S-boxes can achieve better security, they cost high storage as well as have negative effects in the hardware implementation. For example, 8×8 S-boxes of AES require more than 200 GE. On the other hand, compared with the 4×4 S-boxes, the 3×3 S-boxes security is lower. Meanwhile, in hardware implementation, 3×3 S-boxes do not have absolute superiority. For example, 3×3 S-boxes of PRINTcipher [13] require about 11 GE. As the result, we apply two different 4×4 S-boxes, which are designed for lightweight block cipher and widely accepted by other scholars

in this field. The S_1-box is the same as the S-box of PRESENT. The S_2-box is the same as the S0 of mCRYPTON [20]. They work proficiently in maintaining the security and the hardware implementation.

3.5. AddConstants

To promote security and reduce memory requirement of QTL, we have chosen the n for $(0 \le n \le NR)$ as nth round constant of CON_1 and the $(NR+n)$ as nth round constant of CON_2 in AddConstants. We use the AddConstants to avoid the similarities between round F-functions. Thus, QTL is against self-similarity attacks. The round constant does not need memory resource to save, so it requires less cost when implemented in software and hardware.

4. Security analyses

4.1. Differential and linear cryptanalysis

When Biham and Shamir [21] proposed differential cryptanalysis and Matsui [22] proposed linear cryptanalysis, the differential and linear cryptanalysis posed a great threat to many block ciphers. Therefore, we should consider how to resist them in the design of block cipher.

In order to know how the QTL resists differential and linear cryptanalysis, several experiments have been done to test it in our paper. The numbers of active S-boxes are relied on the complexity of differential and linear cryptanalysis, based off which we calculate differential characteristic and linear approximation probabilities. In the differential and linear cryptanalysis, nonlinear operations in an encryption process are treated as linear operations with a probability to model the whole cipher as a linear algorithm. The only nonlinear part in QTL is the S-box layer operation.

Definition 1. For m, m' and $n$are positive integers, and $m \ge m'$. $S_i : GF(2)^m \to GF(2)^{m'}$for $(i = 1, 2, \ldots, n-1, n)$. For any given Δx, $\Gamma x \in GF(2)^m$and Δy, $\Gamma y \in GF(2)^{m'}$, the differential probabilities of each S-box are defined as $DP^{S_i}(\Delta x \to \Delta y)$, and the linear probabilities of each S-box are defined as $LP^{S_i}(\Gamma y \to \Gamma x)$, then

$$DP^{S_i}(\Delta x \to \Delta y) = \frac{\#\left\{x \in GF(2)^m | S_i(x) \oplus S_i(x \oplus \Delta x) = \Delta y\right\}}{2^m}$$

$$LP^{S_i}(\Gamma y \to \Gamma x) = \left(\frac{\#\left\{x \in GF(2)^m | x \cdot \Gamma x = S_i(x) \cdot \Gamma y\right\}}{2^{m-1}} - 1\right)^2$$

where $x \cdot \Gamma x$ denotes the parity (0 or 1) of bitwise product of $x$and Γx [7].

Theorem 1. *For* $(i = 1, 2, \ldots, n-1, n)$,

$$DP^{S_i}(\Delta x \to \Delta y) = \begin{Bmatrix} DP^{S_i}(\Delta x \to \Delta y) \le 1, & if(\Delta x \ne 0, \Delta y \ne 0) \\ DP^{S_i}(\Delta x \to 0) \le 1, & if(\Delta x \ne 0, \Delta y = 0) \\ 0, & if(\Delta x = 0, \Delta y \ne 0) \\ 1, & if(\Delta x = 0, \Delta y = 0) \end{Bmatrix}$$

$$LP^{S_i}(\Gamma y \to \Gamma x) = \begin{Bmatrix} LP^{S_i}(\Gamma x \to \Gamma y) \le 1, & if(\Gamma y \ne 0, \Gamma x \ne 0) \\ 0, & if(\Gamma y \ne 0, \Gamma x = 0) \\ 0, & if(\Gamma y = 0, \Gamma x \ne 0) \\ 1, & if(\Gamma y = 0, \Gamma x = 0) \end{Bmatrix}$$

Definition 2. The maximum differential probabilities of S-boxes are defined asMDP^{S_i}, and the maximum linear probabilities of S-boxes are defined as MLP^{S_i}, then

$$MDP^{S_i} = \max_i \max_{\Delta x \ne 0, \Delta y} DP^{S_i}(\Delta x \to \Delta y)$$

$$MLP^{S_i} = \max_i \max_{\Gamma x, \Gamma y \ne 0} LP^{S_i}(\Gamma y \to \Gamma x)$$

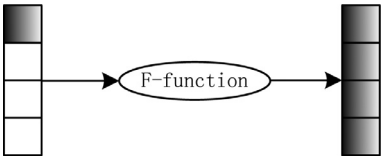


Fig. 4. The active S-boxes of F-function change process.

Definition 3. Definition of active S-boxes, a S-box given a non-zero input difference is a differential active S-box, and a S-box given a non-zero output mask value is a linear active S-box.

The number of active S-boxes changes related to branch number in round function of ciphers. And the differential branch number is identical to the linear branch number [23].

Theorem 2. *Any three-round differential characteristic of QTL has 21 active S-boxes at least.*

Proof. F-function is exceedingly important in network structure of QTL, since the F-function ensures certain number of active S-boxes.

Fig. 4 shows that the differential branch number in F-function is 5. A nibble is 4 bits, where an active S-box means 1 non-zero nibble. We define$\Delta_i = X_{0(16)}||X_{1(16)}||X_{2(16)}||X_{3(16)}$as the ith round input difference characteristic. Specifically, we prove this situation as follows:

We assume that Δ_1 is 1 non-zero nibble in $X_{0(16)}$. In the first round of the QTL, it has one active S-box. After the second round of F-function steps, Δ_2 will have at least 9 active S-boxes. After the third round of F-function steps, Δ_3 will have 11 active S-boxes. To sum up, after three rounds, the minimum active S-boxes are described as $\Delta_1 + \Delta_2 + \Delta_3 = 21$.

If we assume Δ_1 is 2 non-zero nibble in $X_{0(16)}$ (or 1 non-zero nibble in $X_{0(16)}$ and 1 non-zero nibble in $X_{1(16)}$), after the second round of F-function steps, Δ_2 will have at least 8 active S-boxes. And after the third round of F-function steps, Δ_3 will have 11 active S-boxes. When we assume Δ_1 is 1 non-zero nibble in $X_{0(16)}$and 1 non-zero nibble in $X_{2(16)}$, after the second round of F-function steps, Δ_2 will have at least 18 active S-boxes. When $\Delta_2$reaches the third round of F-function steps, it becomes$\Delta_3$which contains 1 active S-box. To sum up, after three rounds, the minimum active S-boxes are demonstrated as $\Delta_1 + \Delta_2 + \Delta_3 = 21$.

If we assume Δ_1 is 3 non-zero nibble in $X_{0(16)}$ (or 2 non-zero nibble in $X_{0(16)}$and 1 non-zero nibble in $X_{1(16)}$), after the second round of F-function steps, Δ_2 will have at least 7 active S-boxes. When Δ_2 reaches the third round of F-function steps, it becomes Δ_3 which contains 11 active S-boxes. When we assume Δ_1 is 2 non-zero nibble in $X_{0(16)}$and 1 non-zero nibble in $X_{2(16)}$ (or 1 non-zero nibble in $X_{0(16)}$, 1 non-zero nibble in $X_{1(16)}$and 1 non-zero nibble in $X_{2(16)}$), after the second round of F-function steps, Δ_2 will have at least 17 active S-boxes. When Δ_2 reaches the third round of F-function steps, it becomes Δ_3 which contains 1 active S-box. To sum up, after three rounds, the minimum active S-boxes are demonstrated as $\Delta_1 + \Delta_2 + \Delta_3 = 21$.

If we assume Δ_1 is 4 non-zero nibble in $X_{0(16)}$ (or 2 non-zero nibble in $X_{0(16)}$and 2 non-zero nibble in $X_{1(16)}$, or 3 non-zero nibble in $X_{0(16)}$and 1 non-zero nibble in $X_{1(16)}$), after the second round of F-function steps, Δ_2 will have at least 6 active S-boxes. When Δ_2 reaches the third round of F-function steps, it becomes Δ_3 which contains 11 active S-boxes. When we assume Δ_1 is 2 non-zero nibble in $X_{0(16)}$and 2 non-zero nibble in $X_{2(16)}$ (or 3 non-zero nibble in $X_{0(16)}$and 1 non-zero nibble in $X_{2(16)}$, or 1 non-zero nibble in $X_{0(16)}$, 1 non-zero nibble in $X_{1(16)}$, 1 non-zero nibble in $X_{2(16)}$and 1 non-zero nibble in $X_{3(16)}$), after the second round of F-function steps, Δ_2 will have at least 16 active S-boxes. When $\Delta_2$reaches the third round of F-function steps, it becomes Δ_3 which contains 1 active

S-box. To sum up, after three rounds, the minimum active S-boxes are demonstrated as $\Delta_1 + \Delta_2 + \Delta_3 = 21$.

Generally, when Δ_1 has more active nibbles in$X_{(64)}$, the minimum number of active S-boxes is no less than 21 after three rounds according to our experiments. Therefore, any three-round differential characteristic of QTL has 21 active S-boxes at least.

Theorem 3. *Any three-round linear approximation of QTL has 21 active S-boxes at least.*

Definition 3 illustrates the differential branch number is equal to the linear branch number. In this paper, for brevity, the proof of the Theorem 3 is omitted.

The maximum probability of differential characteristic and linear approximation depends on the complexity of differential and linear cryptanalysis. Meanwhile, we obtain 2^{-2} and 2^{-2} as MDP and MLP of a S-box, respectively. Thus, for three rounds the MDP is $(2^{-2})^{21} = 2^{-42}$, and the MLP is $(2^{-2})^{21} = 2^{-42}$. Therefore, we expect that the full-round of QTL (16 and 20 rounds for QTL-64 and QTL-128) has enough immunity against differential and linear attacks, since it has large security margin.

4.2. Algebraic attack

When Courtois and Pieprzyk [24] proposed Algebraic attack, Algebraic attack was one of the important cryptanalysis methods. Attackers recover the secret key by solving multivariate algebraic equations. The S-box layer is the only nonlinear part of QTL, which largely determines the safety of QTL. QTL uses many S-boxes that are represented by the number of multivariate algebraic equations, which have a number of variables. If we desire to obtain the key of the block cipher by solving complex multivariate algebraic equations which are the NP-hard, the structure of QTL is able to resist the attack. The QTL S-boxes have 21 quadratic equations in the 8 input/output bits variables over GF (2) [23]. We set the number of S-boxes encryption of QTL as n. The QTL has $E = n \times 21$ quadratic equations in $V = n \times 8$variables, QTL-64 has $n = 16 \times ((4+4) \times 2 + (4+4) \times 2) = 512$S-boxes, and each round has $((4+4) \times 2 + (4+4) \times 2) = 32$ S-boxes. Hence, QTL-64 has $10752 (= 512 \times 21)$ quadratic equations of $4096 (= 512 \times 8)$ variables. QTL-128 has $640 (= 20 \times ((4+4) \times 2 + (4+4) \times 2))$ S-boxes, and which has $13440 (= 640 \times 21)$ quadratic equations of $5120 (= 640 \times 8)$ variables. In our experiment, we only get the following S-boxes algebraic equations after first round, shown as below:

$$S_1 - box \left\{ \begin{array}{c} x_0x_1 + x_1 + x_3 + x_1y_0 = 0 \\ x_1x_2 + y_1y_3 + x_2 + x_3 + y_2 = 1 \\ x_2x_3 + y_0y_3 + y_2y_3 + x_0 + y_1 = 1 \\ x_0 + x_1 + y_1 + x_0y_0 + x_0y_2 + x_0y_3 + x_3y_0 + 1 = 0 \\ x_1 + x_3 + x_0y_1 + x_0y_2 + x_0y_3 + x_1y_2 + x_1y_3 + x_2y_3 + x_3y_0 + x_3y_3 = 0 \end{array} \right\}$$

$$S_2 - box \left\{ \begin{array}{c} x_0 + x_1 + x_3 + x_0x_2 + x_1x_3 + x_1x_2x_3 + x_0x_1x_3 + y_0 = 0 \\ x_0 + x_2 + x_1x_3 + x_0x_3 + x_1x_2 + x_0x_1x_3 + x_0x_2x_3 + y_1 + 1 = 0 \\ x_0 + x_2 + x_3 + x_0x_2 + x_1x_2 + x_0x_1x_3 + x_1x_2x_3 + x_0x_2x_3 + y_2 = 0 \\ x_0 + x_1 + x_2 + x_3 + x_0x_2 + x_0x_3 + x_1x_2x_3 + x_0x_2x_3 + x_0x_1x_2 + y_3 = 0 \\ y_0 + y_1 + y_0y_3 + y_1y_2 + y_1y_2y_3 + x_3 = 0 \end{array} \right\}$$

where x_0,x_1,x_2,$x_3$are the inputs of S-boxes, and y_0,y_1,y_2,y_3 are the outputs of S-boxes. In the experiment, we have no solutions of the equations to obtain the master key. The complexity is about n index. Therefore, the algebraic attack is not a threat to QTL algorithm.

4.3. Related-key attack

In this paper, we estimate that the QTL can resist related-key attack [25,26]. We divide the master key into four parts:K_0, K_1, K_2 and K_3, which are independent from each other. The AddRoundKey is used in F-function for each round. The LED and ITUbee do not have key schedule. The QTL is the same as to them, and this way is simple and securable. In order to evaluate the resistance to related-key attack, we count the minimum number of active S-boxes in related-key settings. In the differential and linear cryptanalysis, we know that any three-round differential characteristic and linear approximation of QTL contain at least 21 active S-boxes. The sub-keys make full influence on the differential characteristic and linear approximation after three-round encryption. Consequently, QTL-64 has 16 iterative rounds, and QTL-128 has 20 iterative rounds. We obtain that QTL-64 has at least $21 \times (16 \div 3) =$ 112active S-boxes, and QTL-64 has at least $21 \times (20 \div 3) = 140$ active S-boxes in related-key settings. However, the LED and ITUbee are proved to resist related-key attack, Table 6 shows a comparison between QTL and them. Based on above theoretical analysis, we conclude that QTL can resist to related-key attack.

Table 6
Minimal number of active S-boxes and the MDP and the MLP in related-key setting.

Algorithm	Active S-boxes	MDP	MLP
LED-64	100	2^{-200}	2^{-200}
ITUbee	107	2^{-170}	–
QTL-64	112	2^{-224}	2^{-224}

5. Hardware implementation

From an aspect of efficient hardware implementations, we have designed QTL block cipher. And we take into account the low hardware resource of designing component. Meanwhile, we did experiment to check hardware complexity. QTL was implemented in Verilog-HDL. We used ModelSim SE PLUS 6.1f evaluation for simulation. We synthesized it for a standard cell library based on SMIC 0.18μm CMOS technology. The synthesis is only done for encryption using typical transistors aiming at area optimization by Synopsys design compiler version B-2008.09. One GE is equivalent to the area of a 2-way NAND.

Fig. 5 shows the datapath of QTL-64. Our design basically requires eight 16 bits wide registers, and each is composed of D-flip-flops and requires ten 2-to-1 MUX. A D-flip-flop needs to cost 4.5 GE, and a 2-to-1 MUX needs to cost 2.0 GE. Each round consists of round constant addition, key addition, S-box layer, permutation layer P, Round transposing and right data addition. The 64 bits plaintext is split into four 16 bits inputs; then, the four plaintext inputs are stored in the four 16 bits wide registers. The 64 bits key is split into four 16 bits inputs in a similar way. The four key inputs are stored in the four 16 bits wide registers. The round constant addition is implemented as bit-wire XORs. The two key additions and the two right data additions are also implemented as bit-wire XORs. The S-box layer consists of two different 4×4 S-boxes (S_1-box and S_2-box) where the S_1 is composed of four 4×4 S_1-boxes, the S_2 is composed of four 4×4 S_2-boxes, the four 4×4 S-boxes are utilized in parallel, and the 4×4 S-box is implemented with one XNOR gate, three XOR gates and four NOR gates. The permutation layer P is bit wiring, and the Round transposing is 16 bits wiring. The control logic module manages control signals.

QTL-64 encrypts a 64 bits plaintext with 64 bits key, which requires 16 clock cycles due to a round per clock cycle. QTL-64 implementation requires 1025.52 GE. When QTL-128 encrypts a 64 bits plaintext with 128 bits key, it requires 20 clock cycles. QTL-128 implementation requires 1206.52 GE. At a frequency of 100 KHz, simulated power consumption of QTL-64 and QTL-128 is 1.55 μW and 2.16 μW, respectively. Table 7 shows a comparison between the implementations of QTL-64 and other lightweight block ciphers.

The detailed description of the hardware resources occupied components of QTL block cipher is demonstrated as below. 64 bits key storage requires 344 GE, and 128 bits key storage requires 525 GE. Meanwhile, 64 bits data state requires 344 GE. In F-function, it contains the following four parts. The AddConstants consists of two 8 bits Constants XOR operations which require about 27 GE.

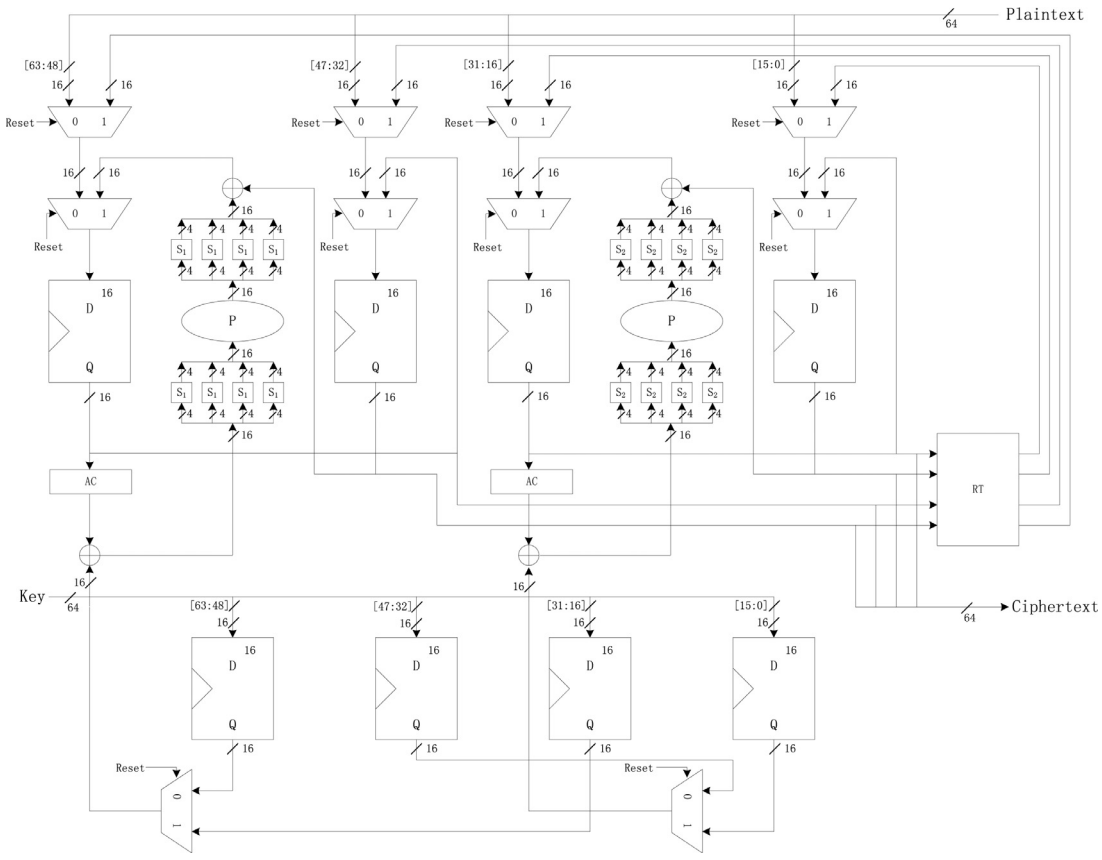


Fig. 5. The datapath of QTL-64.

Table 7
Comparison of lightweight block cipher implementations.

Algorithm	Block size (bits)	Key size (bits)	Area (GE)	Speed (kbps@100KHz)	Logic Process
Piccolo	64	80	1136	237.04	0.18 μm
PRESENT	64	80	1570	200	0.18 μm
KLEIN	64	64	1220	30.9	0.18 μm
LBock	64	80	1320	200	0.18 μm
Twine	64	80	1503	178	0.18 μm
LED	64	80	1040	3.4	0.18 μm
MIBS	64	64	1396	200	0.18 μm
RECTANGLE	64	80	1467	246	0.13 μm
QTL	64	64	1026	200	0.18 μm

The AddRoundKey consists of two 16 bits XOR operations which require about 56.6 GE. The Round transposing and P permutation layer can be simply implemented by wiring, without costing any area. The S-box layer has two different 4×4 S-boxes. The S_1 requires about 96 GE, and the S_2 requires about 89.32 GE. Another two 16 bits XOR operations require about 56.6 GE. Furthermore, for control logics and other counters, QTL-64 requires about 12 GE, and QTL-128 requires about 26 GE. Thus, the hardware implementation of QTL-64 is the sum of an estimated area of 1025.52 GE. QTL-128 is the sum of an estimated area of 1206.52 GE. Table 10 and Table 11 in Appendix II summarize the results of QTL-64 and QTL-128.

6. Conclusion

We propose a new ultra-lightweight block cipher called QTL, which is optimized for extremely resources-constrained devices, such as RFID tags and WSNs. QTL is a new variant of generalized Feistel network structure algorithm. QTL supports 64 bits block with 64 or 128 bits keys. According to the different key lengths, we will denote the ciphers by QTL-64 and QTL-128. QTL has the fast diffusion of the SPNs, which improves security of lightweight block cipher in Feistel-type structures. QTL has numbers of active S-boxes on simple bounds during encryption process. We reduce the cost of hardware implementation energy consumption of the cipher without a key schedule. Meanwhile, the Round transposing and P permutation layer improve diffusion property, and we use the P permutation layer and the Round transposing without cost of hardware. We have proved that QTL offers an adequate security level against classic analyses by experiment. Particularly, it has a positive effect on resistance differential and linear cryptanalysis. Moreover, we present that QTL achieves remarkably compact implementation in hardware. At the smallest implementation, the area for the 64 and 128 bits keys mode only require 1025.52 and 1206.52 GE, which are far less than 2000 gates. Consequently, QTL achieves not only compact implementation but also high security.

Acknowledgements

The author thanks the referees for their helpful and insightful comments. This work was supported by the National Natural Science Foundation of China (Grant No.61572174), Hunan Provincial Natural Science Foundation of China (Grant No.2015JJ4011), and the Scientific Research Fund of Hunan Provincial Education Department (Grant No.15A029).

Appendix I. Test vectors

Test vectors of QTL for each key length are given here. The data are expressed in hexadecimal form. See Tables 8 and 9.

Table 8
Test vectors for QTL-64.

Plaintext	Key	Ciphertext
0000-0000-0000-0000	0000-0000-0000-0000	3337-CF86-D478-6DB4
0000-0000-0000-0000	FFFF-FFFF-FFFF-FFFF	A0C2-5416-D1D9-ADB9
FFFF-FFFF-FFFF-FFFF	0000-0000-0000-0000	5F3D-ABE9-2E26-5246
FFFF-FFFF-FFFF-FFFF	FFFF-FFFF-FFFF-FFFF	CCC8-3079-2B87-924B
36E6-5AAE-2BC1-17D8	3995-48C2-7529-023F	9178-BEA5-0D3A-91E0

Table 9
Test vectors for QTL-128.

Plaintext	Key	Ciphertext
0000-0000-0000-0000	0000-0000-0000-0000 0000-0000-0000-0000	2F1B-A2E6-37F9-56F5
0000-0000-0000-0000	FFFF-FFFF-FFFF-FFFF FFFF-FFFF-FFFF-FFFF	873A-FB4B-9BEC-CCD4
FFFF-FFFF-FFFF-FFFF	0000-0000-0000-0000 0000-0000-0000-0000	78C5-04B4-6413-332B
FFFF-FFFF-FFFF-FFFF	FFFF-FFFF-FFFF-FFFF FFFF-FFFF-FFFF-FFFF	D0E4-5D19-C806-A90A
36E6-5AAE-2BC1-17D8	152A-8E10-564F-278B 5520-AE42-865F-0326	C884-1778-0CF2-F3FA

Table 10
Area requirement of QTL-64.

Module (Round function)	GE
Data register	344
Constants Xor	27
Two key Xors	56.6
Permutation layer P	0
Two data Xors	56.6
S-box layer	185.32
Round transposing	0
Key register	344
Control logics and other counters	12
Total	1025.52

Table 11
Area requirement of QTL-128.

Module (Round function)	GE
Data register	344
Constants Xor	27
Two key Xors	56.6
Permutation layer P	0
Two data Xors	56.6
S-box layer	185.32
Round transposing	0
Key register	525
Control logics and other counters	12
Total	1206.52

Appendix II. Area requirement

See Tables 10 and 11.

Appendix III. The $CON_1^{n'}$ and $CON_2^{n'}$

See Tables 12 and 13.

Table 12
The $CON_{1^{n'}}$ and $CON_{2^{n'}}$ of QTL-64.

n	0	1	2	3	4	5	6	7
$CON_1^{n'}$	0F	0E	0D	0C	0B	0A	09	08
$CON_2^{n'}$	1F	1E	1D	1C	1B	1A	19	18
n	8	9	10	11	12	13	14	15
$CON_1^{n'}$	07	06	05	04	03	02	01	00
$CON_2^{n'}$	17	16	15	14	13	12	11	10

Table 13
The $CON_1^{n'}$ and $CON_2^{n'}$ of QTL-128.

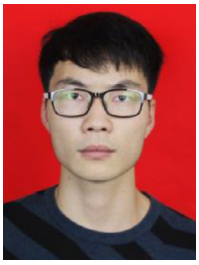
n	0	1	2	3	4	5	6	7	8	9
$CON_1^{n'}$	13	12	11	10	0F	0E	0D	0C	0B	0A
$CON_2^{n'}$	27	26	25	24	23	22	21	20	1F	1E
n	10	11	12	13	14	15	16	17	18	19
$CON_1^{n'}$	09	08	07	06	05	04	03	02	01	00
$CON_2^{n'}$	1D	1C	1B	1A	19	18	17	16	15	14

References

[1] A. Juels, S.A. Weis, Authenticating pervasive devices with human protocols, in: Proceeding of Advances in Cryptology-CRYPTO 2005, Springer, 2005, pp. 293–308.
[2] National Institute of Standards and Technology (NIST), Specification for the Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001.
[3] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT: an ultra-lightweight block cipher, in: Proceeding of Cryptographic Hardware and Embedded Systems -CHES 2007, Springer, 2007, pp. 450–466.
[4] W. Wu, L. Zhang, L. Block, A Lightweight block Cipher, in: Proceeding of Applied Cryptography and Network Security-ACNS 2011, Springer, 2011, pp. 327–344.
[5] T. Suzaki, K. Minematsu, S. Morioka, E. Kobayashi, TWINE: a lightweight, versatile block cipher, in: Proceeding of ECRYPT Workshop on Lightweight Cryptography 2011, 2011, pp. 146–169.
[6] Z. Gong, S. Nikova, Y.W. Law, KLEIN: A new family of lightweight block ciphers, in: Proceeding of RFIDSec 2011, Springer, 2012, pp. 1–18.
[7] M. Izadi, B. Sadeghiyan, S.S. Sadeghian, H.A. Khanooki, MIBS: A new lightweight block cipher, in: Proceeding of Cryptography and Network Security-CANS 2009, Springer, 2009, pp. 334–348.
[8] J. Guo, T. Peyrin, A. Poschmann, M. Robshaw, The LED block cipher, in: Proceeding of Cryptographic Hardware and Embedded Systems-CHES 2011, Springer, 2011, pp. 326–341.
[9] J. Borghoff, A. Canteaut, T. Güneysu, E.B. Kavun, M. Knezevic, L.R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S.S. Thomsen, T. Yalçın, PRINCE- A Low-Latency Block Cipher for Pervasive Computing Applications, in: Proceeding of ASIACRYPT 2012, Springer, 2012, pp. 208–225.
[10] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, T. Shirai, Piccolo: an ultra-lightweight blockcipher, in: Proceeding of Cryptographic Hardware and Embedded Systems-CHES 2011, Springer, 2011, pp. 342–357.
[11] F. Karakoc, H. Demirci, A.E. Harmancı, ITUbee: a software oriented lightweight block cipher, in: Proceeding of Lightweight Cryptography for Security and Privacy- LightSec2013, Springer, 2013, pp. 16–27.
[12] H. Yap, K. Khoo, A. Poschmann, M. Henricksen, EPCBC - a block cipher suitable for electronic product code encryption, in: Proceeding of Cryptography and Network Security- CANS 2011, Springer, 2011, pp. 76–97.
[13] L. Knudsen, G. Leander, A. Poschmann, M.J.B. Robshaw, PRINTcipher: a block cipher for IC-printing, in: Proceeding of Cryptographic Hardware and Embedded Systems-CHES 2010, Springer, 2010, pp. 16–32.
[14] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, I. Verbauwhede, RECTANGLE: A bit-slice ultra-lightweight block cipher suitable for multiple platform, Science China Information Sciences 58 (12) (2015) 1–15.
[15] M. Renauld, F.X. Standaert, Algebraic Side-Channel Attacks, in: Proceeding of Information Security and Cryptology-Inscrypt 2009, Springer, 2010, pp. 393–410.
[16] L. Yang, M. Wang, S. Qiao, Side Channel Cube Attack on PRESENT, in: Proceeding of Cryptography and Network Security- CANS 2009, Springer, 2009, pp. 379–391.
[17] C. Manifavas, G. Hatzivasilis, K. Fysarakis, K. Rantos, Lightweight Cryptography for Embedded Systems – A Comparative Analysis, in: Proceeding of DPM 2013 and SETOP 2013, Springer, 2014, pp. 333–349.
[18] Y. Wang, W. Wu, X. Yu, Biclique Cryptanalysis of Reduced-Round Piccolo Block cipher, in: Proceeding of ISPEC 2012, Springer, 2012, pp. 337–352.
[19] S. Hong, S. Lee, J. Lee, J. Sung, D. Cheon, I. Cho, Provable Security against Differential and Linear Cryptanalysis for the SPN Structure, in: Proceeding of FSE 2000, Springer, 2001, pp. 273–283.

[20] C.H. Lim, T. Korkishko, mCrypton – A lightweight block cipher for security of low-cost RFID tags and sensors, in: Proceeding of WISA 2005, Springer, 2006, pp. 243–258.
[21] E. Bilam, A. Shamir, Differential cryptanalysis of the full 16-round DES, in: Proceeding of CRYPTO 1992, Springer, 1993, pp. 487–496.
[22] M. Matsui, Linear cryptanalysis method for DES cipher, in: Proceeding of Advances in Cryptology - EUROCRYPT 1993, Springer, 1994, pp. 386–397.
[23] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E.D. Win, The cipher SHARK, in: Proceeding of Fast Software Encryption - FSE 1996, Springer, 1996, pp. 99–111.
[24] N.T. Courtois, J. Pieprzyk, Cryptanalysis of block ciphers with overdefined systems of equations, in: Proceeding of ASIACRYPT 2002, Springer, 2002, pp. 267–287.
[25] E. Biham, O. Dunkelman, N. Keller, Related-key boomerang and rectangle attacks, in: Proceeding of EUROCRYPT 2005, Springer, 2005, pp. 507–525.
[26] E. Biham, O. Dunkelman, N. Keller, A unified approach to related-key attacks, in: Proceeding of FSE 2008, Springer, 2008, pp. 73–96.

Lang Li received his Ph.D. and Master's degrees in computer science from Hunan University, Changsha, China, in 2010 and 2006, respectively, and earned his B.S. degree in circuits and systems from Hunan Normal University in 1996. Since 2011, he has been working as a professor in the Department of Computer Science at the Hengyang Normal University, Hengyang, China. He has research interests in embedded system and information security.



Botao Liu has been working towards his B.S. degree in Hengyang Normal University, Hengyang, China, since 2012. His main research interest lies in embedded system and information security.

Hui Wang received the Master's degree from the Department of Computer Science, Nanjing University, Nanjing, China, in 2008. He had ten years of working experience in Canada where he worked as a senior programmer in a software company, as a senior engineer of database, and a project manager in Morgan Stanley. His current research lies in business intelligence, data processing, information security, etc.