IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SECURITY AND TRUSTED COMPUTING FOR INDUSTRIAL INTERNET OF THINGS

Received May 28, 2018, accepted July 1, 2018, date of publication July 12, 2018, date of current version August 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2855408

Compact Hardware Implementation of a SHA-3 Core for Wireless Body Sensor Networks

YI YANG[1,2], DEBIAO HE[1,2], NEERAJ KUMAR[3], (Member, IEEE), AND SHERALI ZEADALLY[4]

[1]Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
[2]Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
[3]Department of Computer Science and Engineering, Thapar University, Patiala 147004, India
[4]College of Communication and Information, University of Kentucky, Lexington, KY 40506, USA

Corresponding author: Debiao He (hedebiao@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61501333, Grant 61572379, Grant 61572370, and Grant U1536204, in part by the National High-Tech Research and Development Program of China (863 Program) under Grant 2015AA016004, and in part by the Natural Science Foundation of Hubei Province of China under Grant 2015CFB257.

ABSTRACT One of the most important Internet of Things applications is the wireless body sensor network (WBSN), which can provide universal health care, disease prevention, and control. Due to large deployments of small scale smart sensors in WBSNs, security, and privacy guarantees (e.g., security and safety-critical data, sensitive private information) are becoming a challenging issue because these sensor nodes communicate using an open channel, i.e., Internet. We implement data integrity (to resist against malicious tampering) using the secure hash algorithm 3 (SHA-3) when smart sensors in WBSNs communicate with each other using the Internet. Due to the limited resources (i.e., storage, computation, and communication capabilities) of sensors in WBSNs, a lightweight implementation of SHA-3 is needed. To address this challenge, we propose a new implementation of the SHA-3, which has a compact hardware architecture. Our implementation of SHA-3 consists of a reliable logic structure, random access memory, and an enhanced finite state machine. The simulation on a Vitrtex-5 field programmable gate array shows that the proposed implementation is suitable for the WBSN on different applications. We evaluate the sensor area of the proposed SHA-3 implementation and compare it with other recently proposed hardware implementations of SHA-3. In addition, our hardware implementation approach reduces the area by almost 74.7% compared with the recently proposed hardware implementation which has the smallest area.

INDEX TERMS Body sensor network, cryptographic hash function, encryption, FPGA, hardware, IoT, SHA-3, wireless.

I. INTRODUCTION

Nowadays, the Internet of Things (IoT), is widely applied in network convergence through intelligent sensing, pervasive computing and other communication sensing technology, which are gradually changing the daily lives of people. IoT is also called the third wave of the world information industry after the computer and the Internet [1] eras. One of the most important IoT applications is the Wireless Body Sensor Network (WBSN) (Fig 1). A WBSN enables the physiological parameters of the human body to be collected by body sensors. The WBSN is not only a new solution for universal health care, disease prevention and control, but also an important part of the IoT [2] ecosystem.

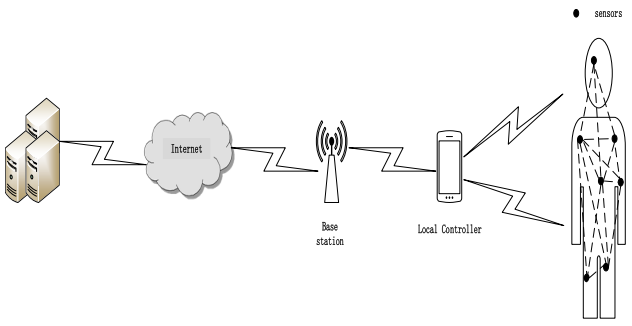


FIGURE 1. The WBSN based on the IoT.

The WBSN enables the integration of intelligent, miniaturized and low power sensor nodes to monitor the body's function and the surrounding environment. As shown in figure 1, when the local controller sends the instructions to the sensors, they will extract the designated physiological

body signals and transmit the information to the local controllers. The body sensors are integrated with bio-sensors to measure blood pressure, electrocardiogram and so on. Then the local controller sends the collected signal values to the base station which delivers the WBSN physiological data to the servers for analysis and storage. Due to the massive scale and distributed nature of IoT networks [3], [4], security and privacy guarantees (e.g., security and safety-critical data, sensitive private information) are major challenges [5], [6]. It is important to keep data integrity [7] in the WBSN environment. If the WBSN has no integrity checking, then the attacker can change the data by adding additional data. The WBSN also needs to ensure the sensor area are as small as possible. However, the current implementations of WBSN cannot meet lightweight security requirements [8] essential for WBSN applications.

In this work, we propose an implementation of SHA-3 with a compact hardware architecture for the WBSN to ensure these aforementioned requirements by using the SHA-3 data encryption algorithm implementation on the FPGA for sensors. The main goal is to make messages of variable lengths map to the message digest of fixed length. With the rapid development of computer technology and advances in cryptology, NIST issued the SHA-3 algorithm which is based on the Keccak algorithm. In a sense, the SHA-3 algorithm is the other name for the Keccak algorithm. The Keccak algorithm is faster than the Blake algorithm, the JH algorithm and Skein algorithm when it performs integrity verification of a large amount of data and it also has good security.

The compact hardware architecture makes the sensor area in the WBSN as small as possible. We have used the Random Access Memory (RAM) to store intermediate variables to reduce areas. The FPGA technology can provide speed and stability without high investments on the initial cost of ASIC design. The FPGA is also easy to maintain for a long time [9], [10].

A. OUR CONTRIBUTIONS

We summarize the main contributions of this work as follows:

1) We propose a compact hardware implementation of SHA-3 by using FPGA suitable for the sensors of WBSN.

2) We conduct a performance evaluation about the sensor area of the proposed implementation and compare our results with other recently proposed hardware implementations of SHA-3.

B. ORGANIZATION OF THE REST PAPER

The rest of the paper is organized as follows. Section 2 describes the SHA-3 algorithm. Section 3 reviews related work recent hardware implementations of SHA-3. Section 4 presents our proposed compact hardware implementation of SHA-3. Section 5 evaluates the occupied slices of the proposed SHA-3 implementation and compare our results with other hardware implementations of SHA-3.

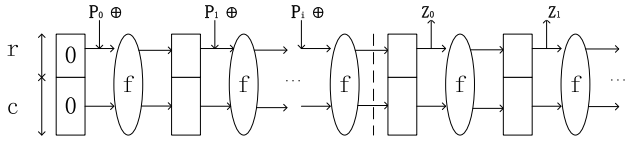


FIGURE 2. Hermetic sponge strategy.

II. THE SHA-3 ALGORITHM

The SHA-3 algorithm has a sponge structure and the core of the sponge structure is the round operation which we will describe in this section.

A. THE SPONGE STRUCTURE

The SHA-3 algorithm has the Hermetic Sponge Strategy (HSS). As shown in figure 2, P is the input and Z is the output after the hash is done. f is the permutation function. r is the baud rate. c is the capacity. We need to guarantee that the sum of r and c is 1600. The input message gets into the input state through the padding function. The permutation function operates on the padded message in the state which can be seen as a 5*5*w array and can be expressed as a[5][5][w]. The w is the length of the state's lane. It also can be seen as the w-bit CPU's one byte [9], [10].

In this paper, we use the Keccak-512 algorithm. The value of r is 512 and the value of c is 1088. The details about the sponge structure are shown as follows.

Keccak[r,c,d](M)	
Initialization and padding:	
$S[x,y] = 0$	$\forall x, y \in [0,4]$
$P = M\|\|byte(d)\|\|byte(r/8)\|\|0x01\|\|0x00\|\|...\|\|0x00$	
Absorbing Phase for every block Pi in P:	
$S[x,y] = S[x,y] \oplus Pi[x+5y]$	$x+5y < r/w$
$S = Keccakf[r+c](S)$	
Squeezing Phase:	
$Z = emptystring$	
While output is requested	
$Z = Z\|\|S[x,y]$	$x+5y < r/w$
$S = Keccak - f[r+c](S)$	
return Z	

1) INITIALIZATION AND PADDING

A 1600-bit number is defined and initialized 0. This number is divided into 25 groups. Each group represents a state and the value of the initial state is set to 0. We input a message M and the length of M is L. If L is a multiple of 576 bits, it does not need to be padded. Otherwise,L is padded as the smallest multiple of 576 bits. The padding rule has three steps: first, padding 1 and the number of ones is one; then, padding 0 and the number of zero is n and the last padding byte is 0x80 (i.e.10000000); after padding, the length of message is the smallest multiple of 576 bits.

2) ABSORBING PHASE

The absorbing phase divides the padded message into n 576-bit message blocks. Each 576-bit message block is divided into nine states. These 9 states do the XOR operation with 9 states which are from 25 initial states and output the results into these 9 states. The results become the initial states for processing the next 576-bit message block. The above

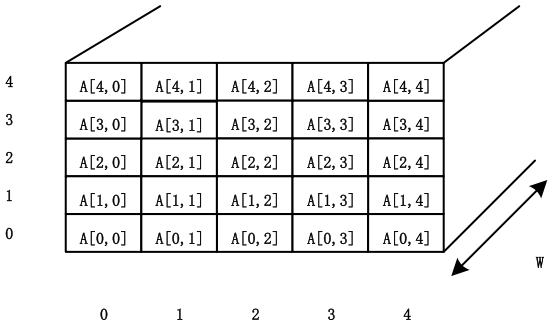


FIGURE 3. Status of the 25 states.

operations are repeated until all 576-bit message blocks are absorbed. The final states are used in the round operation.

3) SQUEEZING PHASE

The compression function is the final stage of the output. As we used a fixed 512-bit output length in this paper, we need take the 64-bit data from one of 25 states. This process is repeated 8 times and the 25 states are obtained by executing the round operation at each iteration.

B. ROUND OPERATION

In the Keccak-512 algorithm, there are 24 rounds in the compression function and every round has five steps. As mentioned previously, one of the most important parts in the compression function is the 25 states at every round (Fig 3).

The pseudo code of the round operation is as follows. The complete state array is represented by A and A[i,j]. The Rotation Operation (ROT) is used to do the bit-rotation operation. The r[i,j] and RC are constants [9], [10].

Keccak[r,c,d](M)
Initialization and padding:
$S[x, y] = 0$ $\forall x, y \in [0, 4]$
$P = M||byte(d)||byte(r/8)||0x01||0x00||...||0x00$
Absorbing Phase for every block Pi in P:
$S[x, y] = S[x, y] \oplus Pi[x + 5y]$ $x + 5y < r/w$
$S = Keccakf[r + c](S)$
Squeezing Phase:
$Z = emptystring$
While output is requested
$Z = Z||S[x, y]$ $x + 5y < r/w$
$S = Keccak - f[r + c](S)$
return Z

III. RELATED WORK

There are some papers which have recently been published on the FPGA implementation of SHA-3. Most of the research efforts have focused on different iterative architectures, pipelining and the simple Hardware Descriptive Language (HDL) implementations. Rao *et al.* [11] and Latif *et al.* [12] proposed an implementation of SHA-3 by using Xilinx Look-Up-Table(LUT) which has high speed execution which makes the scheme suitable for supporting Bump in The Wire (BITW) security for IoT applications. Gaj *et al.* [13] proposed the use of pipelining to deliver high throughput. Provelengios *et al.* [14] implemented SHA-3 without DSP48E and has high TPA (throughput/area). But the use of Digital Signal Processor (DSP) devices was an inefficient method. Baldwin *et al.* [15] presented all the proposed designs that participated in the 2nd round of the SHA-3 competition. Jungk and Stöttinger [16] proposed a slice-oriented architecture with different data path widths and they obtained higher throughput when the data path width is wider. However, to date, little work has been undertaken on the design of compact hardware implementation of the SHA-3 for WBSNs.

In this work, we propose a hardware implementation method of SHA-3 which we describe next.

IV. HARDWARE IMPLEMENTATION

To achieve the lightweight requirement, the main goal of the proposed hardware implementation of SHA-3 is to have a compact hardware architecture. The proposed hardware implementation of SHA-3 for the WBSN involves two steps. The first step is the primitive design and the second step is the compact hardware architecture.

A. THE PRIMITIVE DESIGN

In this section, we present the logic circuit of the round operation and the running processes of the finite-state machine (FSM) for the primitive design. We use a FSM called FSM-SHA3 to control the logic module of SHA3 and another FSM called FSM-Round to control the round operation. Through the primitive design, we can better understand how to optimize a compact hardware architecture.

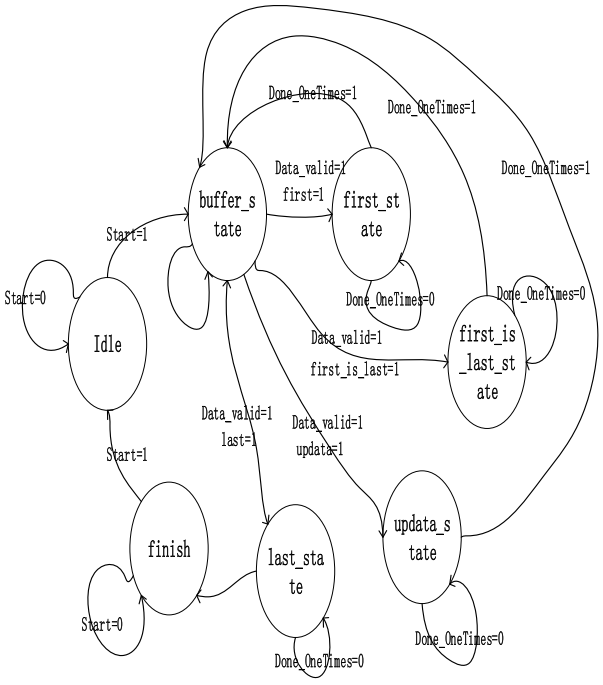


FIGURE 4. State transition diagram of the FSM-SHA3.

The state transition diagram of the FSM-SHA3 (Fig 4) shows how the FSM-SHA3 works. When the values of these signals change, the states also change. Using these states,

the FSM-SHA3 can control the logic operations to output the result:

1) The state ''idle'' means the logic module is idle. If the value of ''start'' is zero, we stay in this state. When the value of ''start'' is one, the state transitions to 2).

2) The state ''buffer-state'' means the logic module is ready to work. When the value of ''Data-valid'' and ''first'' are both one, the state goes to 3). When the value of ''Data-valid'' and ''first-is-last'' are both one, the state goes to 4). When the value of ''Data-valid'' and ''updata'' are both one, the state goes to 5). When value of ''Data-valid'' and ''last'' are both one, the state goes to 6).

3) The state ''first-state'' means the logic module is going to handle the first message block. If the value of ''Done-OneTimes'' is zero, we stay in this state. When the value of ''Done-OneTimes'' is one, the state goes to 2).

4) The state ''first-is-last-state'' means the logic module is going to handle only one message block. If the value of ''Done-OneTimes'' is zero, we stay in this state. When the value of ''Done-OneTimes'' is one, the state goes to 2).

5) The state ''updata-state'' means that the logic module is going to update the message block. If the value of ''Done-OneTimes'' is zero, we stay in this state. When the value of ''Done-OneTimes'' is one, the state goes to 2).

6) The state ''last-state'' means logic module is going to handle the last message block. If the value of ''Done-OneTimes'' is zero, we stay in this state. When the value of ''Done-OneTimes'' is one, the state goes to 7).

7) The state ''finish'' means the logic module has already handled all messages. If the value of ''start'' is zero, we stay in this state. When the value of ''start'' is one, the state transitions to the 1).

The core operation which is the round operation has been described clearly for the primitive hardware design in THETA step (Fig 5), RHO and PI step (Fig 6), CHI and IOTA step (Fig 7). They are steps of the SHA-3 core. In this design, only five outputs ($A[0, 0]$, $A[0, 1]$, $A[0, 2]$, $A[0, 3]$, $A[0, 4]$) are shown out of the 25 possible outputs. Other outputs can be obtained by the same operation. Each input, output, and individual line is a 64-bit word.

Since the core operation is the round operation, the core module that runs the round operation is called the round module. When the message block is handled, the Round module is executed. It is important to design the Round module to guarantee the correctness of the message block processing. It includes two parts: the FSM to control the round operation called FSM-Round, and the round logic module.

The state transition of the FSM-Round (Fig 8) shows how the FSM-Round works. When the value of these signals change, the states change. Through these states, the FSM-Round can control the logic operation to output the result:

1) The state ''idle'' means the round logic module is idle. If the value of ''star'' is zero, we stay in this state. When the value of ''start'' is one, the state goes to 2).

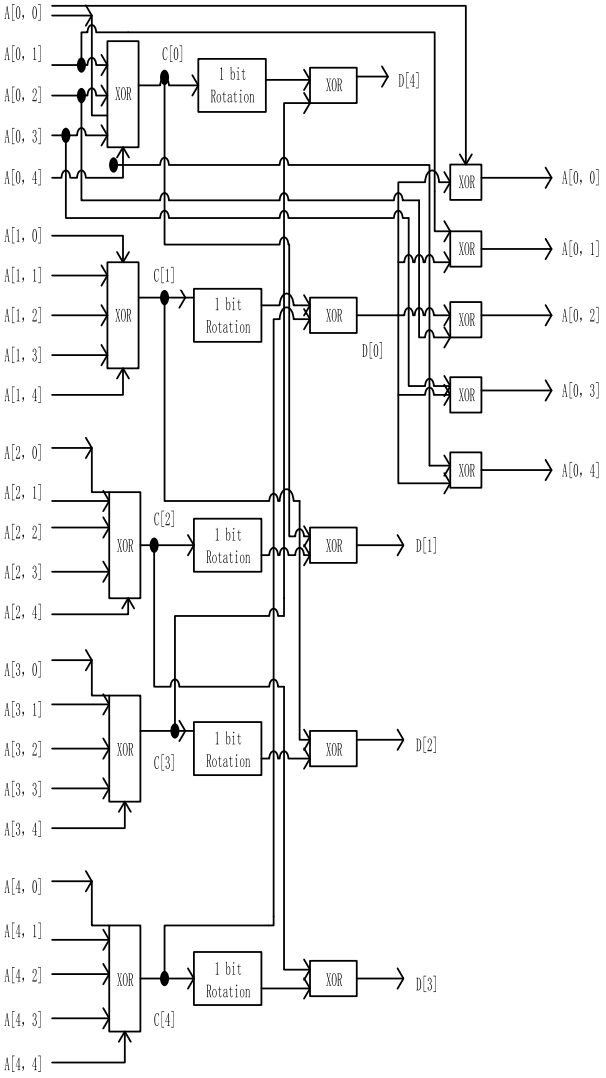


FIGURE 5. THETA step.

2) The state ''buffer1'' means the round logic module is ready to work. This state moves to 3).

3) The state ''run'' means the round logic module is running. If the value of ''Round'' is 22, the state goes to 4). When the value of ''Round'' is less than 22, the state goes to 2).

4) The state ''finish'' means the round logic module has already worked. This state moves to 1).

The area of the primitive design is nearly 1500 slices and it is not small for the implementation of the sensors in the WBSN. First, the 5 inputs of the XOR operation need to store intermediate values which could take up many areas at the THETA step. Second, the XOR operation, the NOT operation and the AND operation also need to store intermediate values at the CHI step. Third, the basic logic circuit could take up a lot of area. In addition, at the RHO and PI step, we waste a lot of time calculating the value of the r[i,j] which slows down the speed of the implementation.

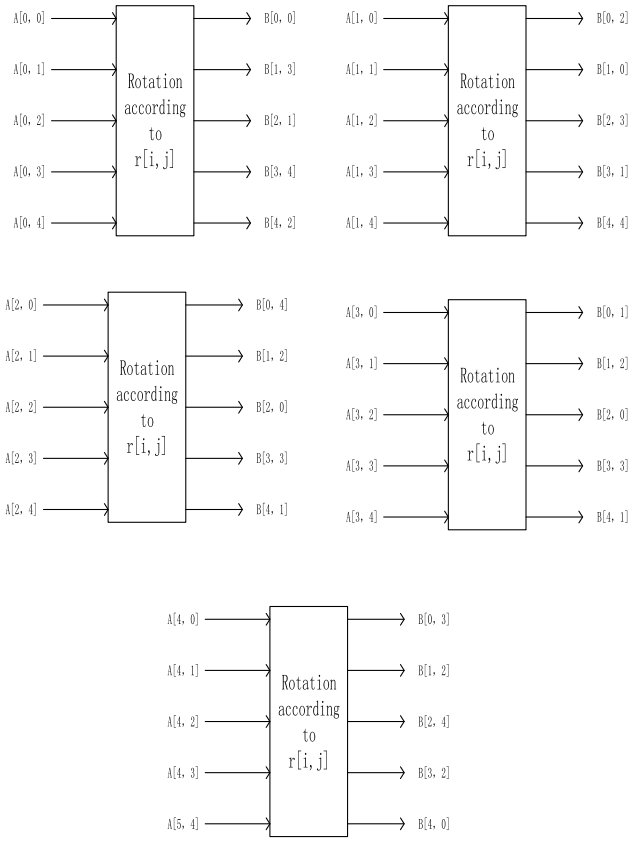


FIGURE 6. RHO and PI step.

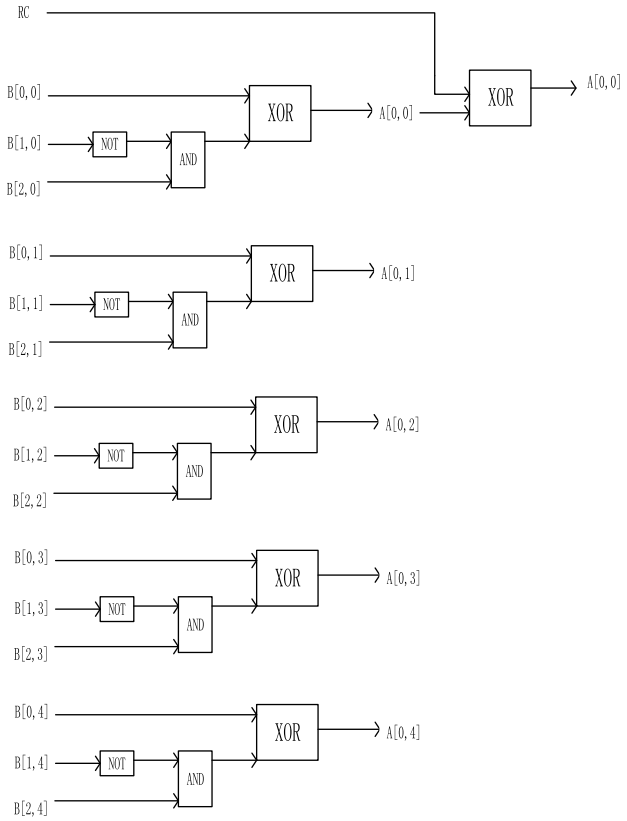


FIGURE 7. CHI and IOTA step.

B. COMPACT HARDWARE ARCHITECTURE

In this work, the SHA-3 implementation is for the sensors of the WBSN. For the manufacturer, it is important to guarantee low cost and a few other benefits. For users, it is important to ensure the availability of WBSN. So it is necessary to ensure that all sensors fit a small area.Since the primitive design which has a big area is inadequate, we design a compact hardware architecture to optimize the implementation. The compact hardware architecture uses the ARM to store the operating instructions and operating numbers. This method can minimize the area.

1) OVERALL FRAME

The implementation of SHA-3 algorithm is consists of 3 modules which are the controlling module, operating module and storing module. We propose a flowchart of modules (Fig 9) to complete the basic logic operation of the SHA-3 algorithm. First, the control module sends the controlling instruction to the operating module and the storing module. Next, the storing module sends the data to the operating module which performs the corresponding logic operations according the control instruction. Finally, the operating module sends the result to the storing module. Based on the flow chart, we design the overall frame of SHA-3 for the hardware implementation.

Based on the flowchart of the modules, we design the overall frame for SHA-3 (Fig 10) which consists of

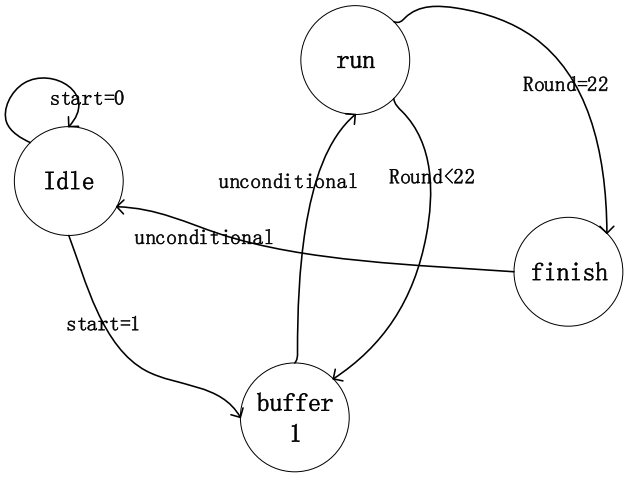


FIGURE 8. State transition diagram of the FSM-round.

the RAM-256-64 module, the SHA3-ALU module and the SHA3-FSM module. The SHA3-ALU module and the SHA3-FSM module are the core of the SHA-3 and they calculate the data as well as, send and receive data to the RAM. In figure 11, there are many interfaces after the optimization. The functions of these interfaces are key to understanding the implementation of SHA-3.

The input signal interfaces of the SHA3-FSM are as follows:

- clk: the clock signal.
- rst-n: the reset signal.

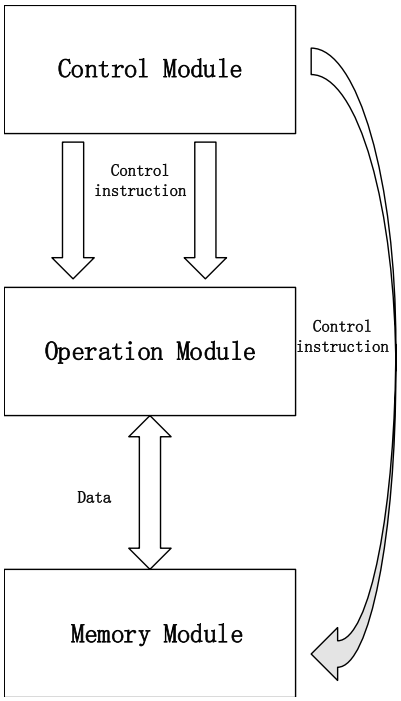


FIGURE 9. Flowchart of modules.

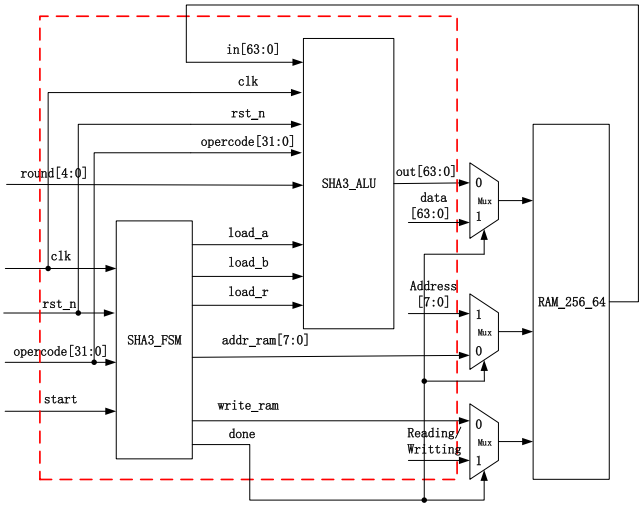


FIGURE 10. Overall frame of the SHA-3 implementation.

•opercode[31:0]: the 32-bit instruction.

•Start: the starting signal.

The output signal interfaces of the SHA3- FSM are as follows:

•load-a: the control signal storing the input data into the a.

•load-b: the control signal storing the input data into the b.

•load-r: the control signal that sends the corresponding operating result.

•done: the sign signal that indicates the operation has been completed.

•addr-ram[7:0]: the control signal of the storage's address.

•write-ram: the control signal of reading and writing.

In figure 10, the SHA-Logic consists of the SHA3-ALU module and the SHA3-FSM module. The outer part of the red line is a single port RAM with a depth of 256-bit and a width of 64-bit. The external machine can read from RAM and write to RAM when the controller is in the idle state. When the controller is in the operating state, the RAM is totally controlled by the controller and it will notified by the "done" signal. When the value of "done" is 1, we are in the "operating" state. When the value of "done" is 0, the state is idle. Most of the time, the value of "done" is 1. It is easy for the message block to be written to the RAM at any time.

2) ROUND OPERATION

This method has a compact hardware architecture and stores the intermediate results into the Random Access Memory (RAM). It needs to split the intermediate logic of the algorithm and arrange the data for the address in RAM. The core of the SHA-3 is the round operation. From the pseudo code of the round operation described above, we note that the basic logical operations of the SHA-3 algorithm are AND, XOR, NOR and SHIFT. By dividing the round operation into the concrete logic operations we can show the specific optimizing process. In this design, only five outputs ($A[0, 0]$, $A[0, 1]$, $A[0, 2]$, $A[0, 3]$, $A[0, 4]$) are shown out of the 25 possible outputs. Other outputs can be obtained by the same operation.

The THETA step (Fig 11) needs to be divided into an XOR of two input values inputting because of $C[i]$ and $D[i]$ which consist of the XOR of five input values. The process is as follows:

1) Calculate $A[i, 0] \oplus A[i, 1]$ and put the result into a cache address of the RAM. Then call the cache data Temp. Then, $Temp = A[i, 0] \oplus A[i, 1]$.

2) Use the value stored in Temp and do an XOR with $A[i, 2]$ and store the result into the cache address which stores the Temp. Thus, $Temp = A[i, 0] \oplus A[i, 1] \oplus A[i, 2]$.

3) Use the value stored in Temp and do an XOR with $A[i, 3]$ and store the result into the cache address which stores the Temp. Thus, $Temp = A[i, 0] \oplus A[i, 1] \oplus A[i, 2] \oplus A[i, 3]$.

4) Use the value stored in Temp and do an XOR with $A[i, 4]$. Then the result is $C[i]$.

5) Calculate $C[i+1] <<< 1$ which needs a rotate left-shift and put the result into the cache address.

6) Take the data of the cache address and do an XOR with $C[i - 1]$. Then the result is $D[i]$.

7) Get $A[i, j]$ which needs an XOR of two input values.

The RH0 and PI step only needs SHIFT. It can increase the computing speed through pre-computation (such as computing all numbers of the shifting operation before the step). These numbers include 0, 36, 3, 41, 18, 1, 44, 10, 45, 2, 62, 6, 43, 15, 61, 28, 55, 25, 21, 56, 27.

The CHI step(Fig 12) needs the operations of XOR, AND and NOR. The CHI step also need to be divided in order to optimize the logic operations. The process is as follows:

1) Perform the operation of NOR and store the result into the cache address, that is, $Temp = (\sim B[i + 1, j])$.

2) Take the data from the cache address and do an AND with $B[i + 2j]$. Then store the result into the cache address, that is, $Temp = (\sim B[i + 1, j]) \bigwedge B[i + 2j]$.

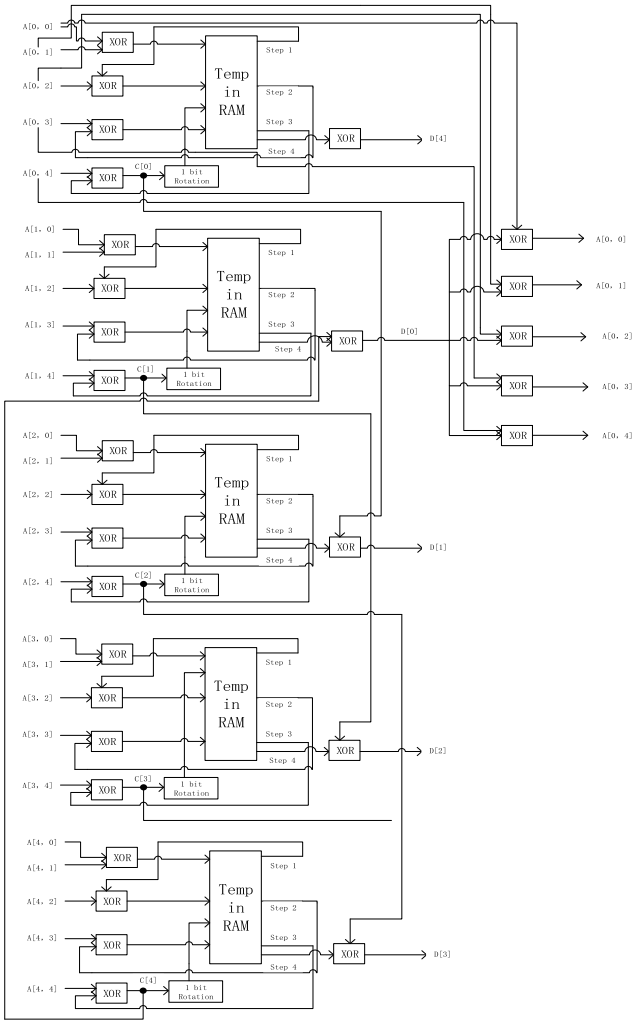


FIGURE 11. THETA step.

3) Take the data from the cache address and perform an XOR with $B[i, j]$. Then, the result is $A[i, j]$.

For the IOTA step, the operation is XOR with the constant. The IOTA step needs to add an XOR of constant input.

The controller, arithmetic unit and storage in the schematic diagram (Fig 13) correspond to with the controlling module, operating module and storing module respectively in the flowchart of the modules. Since the most frequently used logic operation is XOR, we choose to do an XOR operation as an example to show the detailed data processing of the schematic diagram. First, the controller and the arithmetic unit receive the instruction from the external machine and the format of the instruction is shown in Table-1. The controller sends the corresponding control signal according to the format of the instruction. The arithmetic unit parses the eighth bit of the operating instruction to execute the corresponding operation, which will receive the command of the XOR operation. The arithmetic unit performs the XOR operation and sends the result after receiving the control signal called ''load-r''.

The flow of an XOR operation of the controller is as follows:

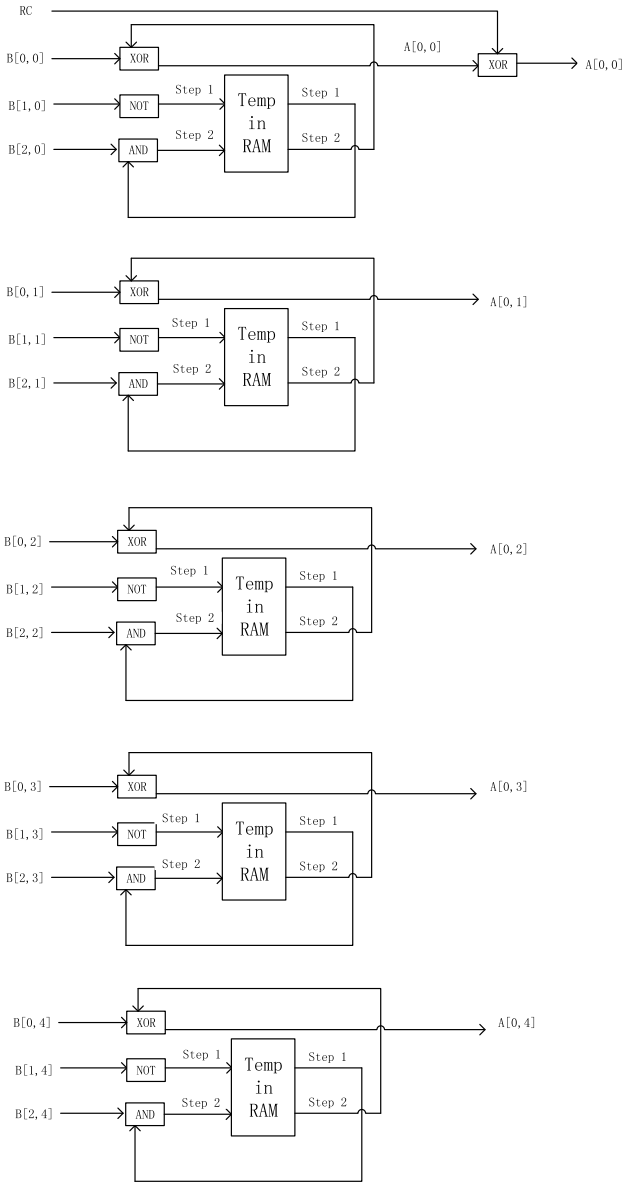


FIGURE 12. CHI step.

TABLE 1. Format of the instruction.

8-bit	8-bit	8-bit	8-bit
address of ″a″	address of ″b″	address of ″r″	operating instruction

1) The controller sends the address of the operand ''a'' to the storage and reads the data from the address location.

2) The controller sends the control signal called ''load-a'' to the arithmetic unit. Then the arithmetic unit sends the data to the register ''a''.

3) The controller sends the address of the operand ''b'' to the storage and reads the data into the address location.

4) The controller sends the control signal called ''load-b'' to the arithmetic unit. Then the arithmetic unit sends the data to the register ''b''.

5) The data for the operation of the XOR is now ready. The controller sends the control signal called ''load-r''. It is

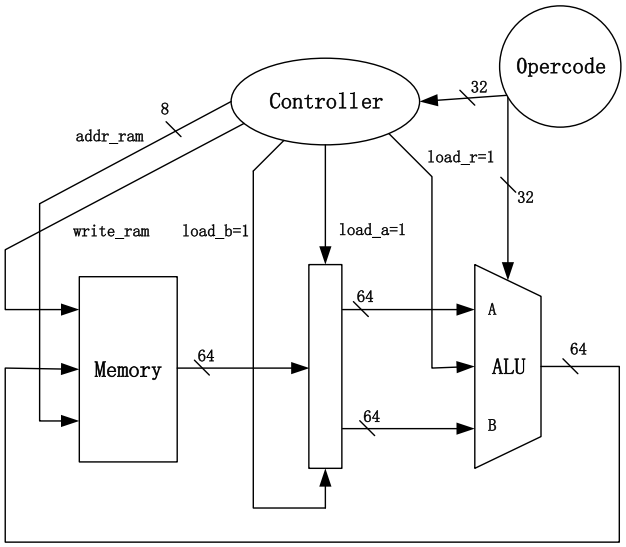


FIGURE 13. The schematic diagram.

used to instruct the arithmetic unit to send the result to the storage. In addition, the controller sends the address of the result and the written instruction to the storage at the same time.

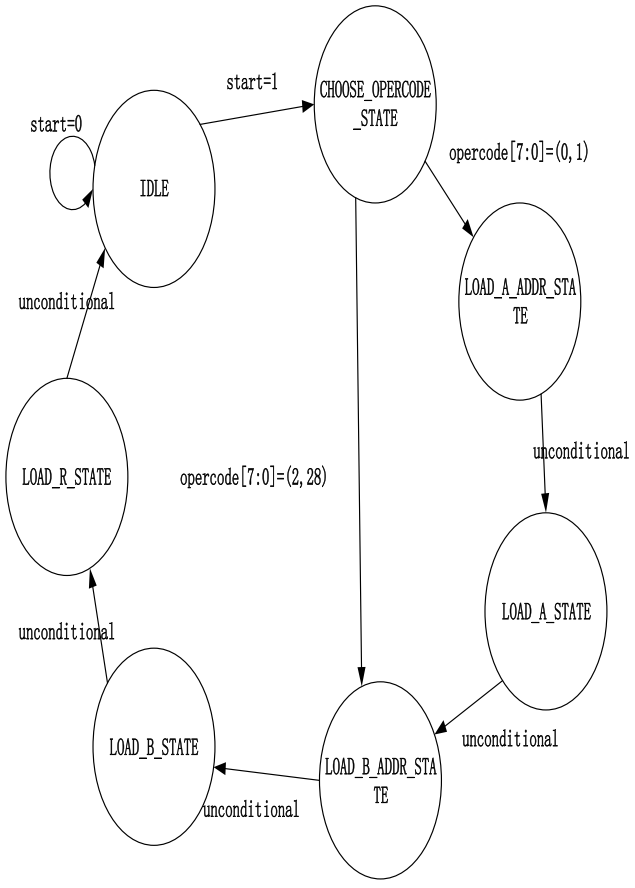


FIGURE 14. State transition diagram of the FSM.

In this method, the controller is implemented by a FSM which is shown in the state transition diagram of the FSM (Fig 14). The FSM has 7 states which include: the idle state (IDLE), the state of selecting the operating command (CHOOSE-OPERCODE-STATE), the state of loading the address of the operand "a" (LOAD-A-ADDR-STATE), the state of loading the operand "a" (LOAD-A -STATE), the state of loading the address of the operand "b" (LOAD-B-ADDR-STATE), the state of loading the operand "b" (LOAD-B -STATE), and the state of loading the result (LOAD-R-STATE).

When the starting signal (start) is invalid, the FSM in the IDLE state. When the "start" is valid, the FSM moves to the "CHOOSE-OPERCODE-STATE". At this state, the FSM moves to the corresponding state according to the low 8-bit of the operating instruction. When the value of the low 8-bit of the operating instruction is 0 or 1, the FSM moves to the "LOAD-A-ADDR-STATE". When the value of the low 8-bit of the operating instruction is between 2 and 28, the FSM moves to the "LOAD-B-ADDR-STATE". When the FSM moves to the "LOAD-A-ADDR-STATE", it will be an unconditional transfer to the "LOAD-A-STATE". The "LOAD-A-STATE" performs an unconditional transfer to the "LOAD-B-ADDR-STATE" and the "LOAD-B-ADDR-STATE" performs an unconditional transfer to the "LOAD-B-STATE". At this time, the "LOAD-B-STATE" performs an unconditional transfer to the "LOAD-R-STATE". Finally, the state switches back to the "IDLE" state from the "LOAD-R-STATE".

By using RAM to store intermediate values and FSM to control the logic module, the area of compact hardware architecture are 278 slices.

V. AREA EVALUATION AND AREA COMPARISON WITH PREVIOUS WORKS

We synthesized our method with Virtex 5 on the FPGA. Table 2 presents the area utilization of LUTs, registers, block RAM and occupied slices in our method. It shows that our hardware implementation has a compact architecture. As mentioned in Section 3, our hardware implement methods for the proposed SHA-3 that use the FPGA and different architectures for better performance. When implementing the sensors on the WBSN, the most important performance characteristic is the area. Table 3 shows the area comparisons which are judged by occupied slices between our method and other previously proposed approaches. Our method reduces the area by almost 74.7% compared with the other recently proposed technique by Latif *et al.* [12] which has the smallest area.

TABLE 2. Area utilization.

Component	SHA3
LUTs	835
Number of registers	7
Block RAM	0
Number of occupied slices	278

In this paper, the proposed method is for the WBSN where in a compact hardware architecture is needed to satisfy the small area requirements. In contrast to recently proposed

TABLE 3. Comparison with other proposed SHA-3 implementations.

References	Platform	Area(Slices)
This work	**Virtex 5**	**278**
Muzaffar et al. [11]	Virtex 5	1409
Latif et al. [12]	Virtex 5	1179
Gaj et al. [13]	Virtex 5	1320
BBaldwin et al. [15]	Virtex 5	1971

methods, our proposed compact hardware implementation of SHA-3 has higher practical performance and benefits.

VI. CONCLUSION

In this work, we have proposed a compact hardware implementation of SHA-3 which makes use of a controlling module, an operating module and a storing module. The controlling module will, according to the external instructions, send the controlling signals to the operating module and the storing module by using RAM and it also optimizes the logic operations through pre-computations. Thus, this approach results in a compact hardware architecture. The reliable hardware structure along with the enhanced FSM, the compact hardware FPGA implementation of SHA-3 is more suitable for deployment in the sensors of the WBSN.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generat. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013, doi: 10.1016/j.future.2013.01.010.

[2] S.-L. Chen, H.-Y. Lee, C.-A. Chen, H.-Y. Huang, and C.-H. Luo, "Wireless body sensor network with adaptive low-power design for biometrics and healthcare applications," *IEEE Syst. J.*, vol. 3, no. 4, pp. 398–409, Dec. 2009, doi: 10.1109/JSYST.2009.2032440.

[3] X. Lai, W. Zou, D. Xie, X. Li, and L. Fan, "DF relaying networks with randomly distributed interferers," *IEEE Access*, vol. 5, pp. 18909–18917, 2017, doi: 10.1109/ACCESS.2017.2751105.

[4] Y. Li, G. Wang, L. Nie, Q. Wang, and W. Tan, "Distance metric optimization driven convolutional neural network for age invariant face recognition," *Pattern Recognit.*, vol. 75, pp. 51–62, Mar. 2018, doi: 10.1016/j.patcog.2017.10.015.

[5] Y. Sun, C. Wong, G.-Z. Yang, and B. Lo, "Secure key generation using gait features for body sensor networks," in *Proc. 14th IEEE Int. Conf. Wearable Implantable Body Sensor Netw., (BSN)*, Eindhoven, The Netherlands, May 2017, pp. 206–210, doi: 10.1109/BSN.2017.7936042.

[6] T. Hwang and P. Gope, "Robust stream-cipher mode of authenticated encryption for secure communication in wireless sensor network," *Secur. Commun. Netw.*, vol. 9, no. 7, pp. 667–679, 2016, doi: 10.1002/sec.1388.

[7] L. Fan, X. Lei, N. Yang, T. Q. Duong, and G. K. Karagiannidis, "Secrecy cooperative networks with outdated relay selection over correlated fading channels," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7599–7603, Aug. 2017, doi: 10.1109/TVT.2017.2669240.

[8] K. E. Friedl, J. D. Hixson, M. J. Buller, and B. Lo, "Guest editorial—13th body sensor networks symposium," *IEEE J. Biomed. Health Inform.*, vol. 22, no. 1, pp. 3–4, Jan. 2018, doi: 10.1109/JBHI.2017.2779898.

[9] M. Li and L. Cheng, "Distinguishing property for full round keccak-f permutation," in *Proc. Conf. Complex, Intell., Softw. Intensive Syst.*, Torino, Italy, 2017, pp. 639–646, doi: 10.1007/978-3-319-61566-0_59.

[10] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak," Bull. Board Syst., Tech. Rep., 2015, p. 389. [Online]. Available: http://eprint.iacr.org/2015/389

[11] M. Rao, T. Newe, I. Grout, and A. Mathur, "High speed implementation of a SHA-3 core on virtex-5 and virtex-6 fpgas," *J. Circuits, Syst., Comput.*, vol. 25, no. 7, p. 1650069, 2016, doi: 10.1142/S0218126616500699.

[12] K. Latif, A. Aziz, and A. Mahboob, "Look-up table based implementations of SHA-3 finalists: Jh, keccak and skein," *KSII Trans. Internet Inf. Syst.*, vol. 6, no. 9, pp. 2388–2404, 2012, doi: 10.3837/tiis.2012.09.024.

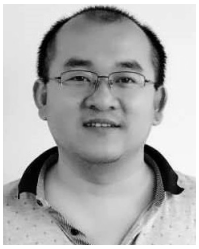
[13] K. Gaj, E. Homsirikamol, M. Rogawski, R. Shahid, and M. U. Sharif, "Comprehensive evaluation of high-speed and medium-speed implementations of five SHA-3 finalists using xilinx and altera fpgas," Bull. Board Syst., Tech. Rep., 2012, p. 368. [Online]. Available: http://eprint.iacr.org/2012/368

[14] G. Provelengios, P. Kitsos, N. Sklavos, and C. Koulamas, "FPGA-based design approaches of Keccak hash function," in *Proc. 15th Euromicro Conf. Digit. Syst. Design, (DSD)*, Izmir, Turkey, Sep. 2012, pp. 648–653, doi: 10.1109/DSD.2012.63.

[15] B. Baldwin *et al.*, "FPGA implementations of the round two SHA-3 candidates," in *Proc. Int. Conf. Field Program. Logic Appl., (FPL)*, Milano, Italy, Aug./Sep. 2010, pp. 400–407, doi: 10.1109/FPL.2010.84.

[16] B. Jungk and M. Stöttinger, "Among slow dwarfs and fast giants: A systematic design space exploration of KECCAK," in *Proc. 8th Int. Workshop Reconfigurable Commun.-Centric Syst. Chip (ReCoSoC)*, Darmstadt, Germany, Jul. 2013, pp. 1–8, doi: 10.1109/ReCoSoC.2013.6581527.

YI YANG received the bachelor's degree from Yunnan University, China. She is currently pursuing the master's degree with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan, China. Her research interests are in the areas of cryptographic algorithm implementation and side channel attacks.



DEBIAO HE received the Ph.D. degree in applied mathematics from the School of Mathematics and Statistics, Wuhan University, in 2009. He is currently a Professor with the Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University. His main research interests include cryptography and information security, in particular, cryptographic protocols.

NEERAJ KUMAR (M'16) received the Ph.D. degree in CSE from Shri Mata Vaishno Devi University, Katra, India. He has guided many students leading to M.E. and Ph.D. He is currently an Associate Professor with the Department of Computer Science and Engineering, Thapar University, Patiala, India. He has authored over 100 technical research papers in leading journals such as-IEEE TII, IEEE TIE, IEEE TDSC, IEEE ITS, IEEE TWPS, IEEE SJ, IEEE ComMag, IEEE WCMag, IEEE NetMag, and conferences. His research is focused on mobile computing, parallel/distributed computing, multi-agent systems, service oriented computing, routing and security issues in mobile ad hoc, sensor, and mesh networks. His research is supported from DST, TCS, and UGC.

SHERALI ZEADALLY received the bachelor's degree in computer science from the University of Cambridge, U.K., and the Ph.D. degree in computer science from The University of Buckingham, U.K. He is currently an Associate Professor with the College of Communication and Information, University of Kentucky. He is a fellow of the British Computer Society and the Institution of Engineering Technology, U.K.