ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/334266145

Physical Unclonable Functions Based Secret Keys Scheme for Securing Big Data Infrastructure Communication

Article *in* Information Sciences · July 2019

DOI: 10.1016/j.ins.2019.06.066

CITATIONS
0

READS
82

5 authors, including:

Fadi Farha
University of Science and Technology Beijing
14 PUBLICATIONS **20** CITATIONS

SEE PROFILE

Huansheng Ning
University of Science and Technology Beijing
200 PUBLICATIONS **3,929** CITATIONS

SEE PROFILE

Laurence Tianruo Yang
St. Francis Xavier University
971 PUBLICATIONS **14,563** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Research on Robustness for Large-Scale Heterogeneous Sensor Networks in Interent of Things View project

Cyberspace View project

All content following this page was uploaded by Fadi Farha on 30 April 2020.

The user has requested enhancement of the downloaded file.

Physical Unclonable Functions Based Secret Keys Scheme for Securing Big Data Infrastructure Communication

Fadi Farha[a], Huansheng Ning[a,*], Hong Liu[b], Laurence T. Yang[c] and Liming Chen[d]

[a]The School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. 100083
[b]The School of Computer Science and Software Engineering, East China Normal University, China
[c]The Department of Computer Science, St. Francis Xavier University, Antigonish, NS, Canada.
[d]The School of Computer Science and Informatics, De Montfort University, Leicester, UK.

ARTICLE INFO

Keywords:
IoT Security
ZigBee Security
SRAM-PUF
Hardware Security
Secret Keys

ABSTRACT

Internet of Things (IoT) is expanding rapidly and so is the number of devices, sensors and actuators joining this world. IoT devices are an important part of the data collection process in Big Data systems, so by protecting them we support and improve the security of the whole system. ZigBee is a secure communication system for the underlying Internet of Things (IoT) infrastructure. Even though ZigBee has a strong security stack built on a variety of secret keys, ZigBee devices are vulnerable to the side-channel and key extraction attacks. Due to the low cost and limited resources, most ZigBee devices store their secret keys in plaintext. In this paper, we focus on protecting the storage of ZigBee secret keys and show how Physical Unclonable Functions (PUFs) can help the ZigBee devices to be robust tamper-resistant against the physical attacks. The proposed schemes include PUF-based key storage protection and key generation. The experiments in this paper were done using SRAM-PUF. Furthermore, two algorithms were proposed to overcome the defects in the randomness of keys generated using SRAM-PUF and, at the same time, to increase the reliability of these keys. We were able to significantly improve the hardware security of ZEDs by protecting their keying materials using costless, high secure, random, stable and volatile PUF-based secret keys.

1. Introduction

With the rapid development of IoT and the emerging of new technology applications and projects, such as smart home, intelligent transport, smart cities and smart energy, many researchers and companies start to pay more attention to the IoT communication infrastructure in order to handle the fast growth in the number of connected devices [24]. As a result, new communication systems have been introduced to meet the demands of low-power consumption, devices management and ensuring the security of the connection by applying ciphering and policies. In addition, new technologies such as blockchain have been deployed to build a decentralized and trustworthy structure for ID management in IoT [19]. With the penetration of IoT into our daily life, sensors are sending sensitive data which could be medical or personal [17]. Therefore, there are many proposed researches to protect the end-user data whether they are on the infrastructure level [13], the cloud storage level [18] or any point along the IoT system architecture. IoT security is non-negotiable, and it is mandatorily required in every new technology joining the IoT world [6].

ZigBee was invented as a new communication system to be used in IoT underlying infrastructure in order to satisfy the demands of security, scalability and power consumption management. It is an open stack suitable for the sensing and control networks. In addition, it provides a good built-in security scheme at both of the network and the application layers. The security services provided by ZigBee include methods for key establishment, key transport, frame protection and device management. ZigBee security system deploys a variety of secret keys which can be used by IoT applications to guarantee secure transmission of the raw data and control commands. Since all messages sent over the air are secured with ZigBee secret keys, ZigBee security services are tightly correlated to the safe installation and storage of the keys materials [2]. However, invoking these secret keys will break the whole system down and could lead to severe problems. As shown in Table 1, there are six types of secret keys defined by the ZigBee Alliance. Some of them, such as master key, link key and network (NWK) key, could be pre-configured and stored in the Non-Volatile

*Corresponding author
fadi_farha@ieee.org (Fadi Farha); ninghuansheng@ustb.edu.cn (H. Ning); liuhongler@ieee.org (H. Liu); lyang@stfx.ca (L.T. Yang); liming.chen@dmu.ac.uk (L. Chen)
ORCID(s):

Table 1
ZigBee Secret Keys

Key type	Acquirement	Description
Master key	Via key-transport or pre-installation	It is shared between two devices to ensure long-term end-to-end communication, besides generating link keys for these devices.
Link key	Via key-transport, key-establishment, or pre-installation	It is exclusively shared between two devices for securing unicast (end-to-end) communication.
Network key	Via key-transport, key-establishment, or pre-installation	It is shared by the whole network devices for securing all the messages on the network layer level and broadcast communications.
Key-load key	Derived from link key after executing specialized keyed hash function	It is used to protect key transport messages carrying the master and link keys.
Key-transport key	Derived from link key after executing specialized keyed hash function	It is used to protect key transport messages carrying NWK keys.
Data key	Equal to the link key	It is used to protect data messages.

Memory (NVM), while the others, such as data-key key, key-load key and key-transport key, are derived from the link key during the runtime by executing keyed Hash Message Authentication Code (HMAC) [2]. Even though each layer in ZigBee has its own secret keys, ZigBee standard allows NWK keys to be used at both of the network and the application layers. Because each layer runs its own ciphering operations, the secret keys at different layers should not be the same to avoid unnecessary iteration encryption using the same secret key. All the secret keys used in ZigBee, mentioned in Table 1, need protection to ensure ZigBee network security. The most serious attack targeting the secret keys is reading them out from the NVM. In secure ZigBee network, at least one key (master key or link key) should be pre-installed in the device before it is deployed in the system, in which way the device can safely receive the active NWK key sent by the Trust Center (TC) and encrypted by key-transport key. Most of ZigBee devices are configured with Advanced Encryption Standard (AES) specified-hardware which is usually a co-processor for executing AES encryption/decryption operations because AES-128 is the only cryptographic algorithm adopted by ZigBee Alliance [2].

Nowadays, physical attacks against cryptography systems represent a big challenge to the modern technologies due to the advanced tools used by attackers in order to extract critical information from the hardware devices (known as side-channel attacks) or inject false data in an attempt to break the system security down (known as fault injection attacks) [3]. All that threatens the plaintext traditional key storage of the security systems especially in the high-risk areas. Cryptographic algorithms in modern technology are so strong that they can protect the logical links between endpoint devices. Therefore, physical devices have become the most vulnerable point in the whole security system [20]. As a result, many researchers are working on hardware security because physical attacks are as dangerous as other threats and can endanger the security of IoT in general and the Field Programmable Gate Arrays (FPGA) devices in particular [16].

Hardware Intrinsic Security (HIS) is one of the strongly recommended solutions to enhance the hardware security of devices starting by, but not limited to, securing secret key storage. One of the famous HIS implementations is Physical Unclonable Function (PUF). The main concept of PUF is extracting useful information from the intrinsic physical properties of the objects. Researchers found that slight mismatches in some physical characters, such as threshold voltage and mobility in Metal-Oxide Semiconductor (MOS), still exist even when producing identical electronic elements because of uncontrollable production variations. Nowadays, there are many types of PUFs which use different mismatches sources and can meet the hardware key requirements of being random, unpredictable, and tamper-resistant. Any attempt to remove the PUF will come with a high risk of destroying it and wasting the key forever [22]. PUF could be used for circuit identification, such as assigning an ID to a circuit or a device [10], as a seed for a pseudo-random number generator [1], as a secret key generator [12] and for authentication using Challenge-Response pairs (CRPs) [23].

The most commonly used PUF for securing the storage of secret keys are the memory-based PUFs for two reasons [7]: first, their components, such as SRAM, latches and flip-flops, are widely used by electronic devices or FPGAs;

second, generating the memory-based PUFs output when needed is relatively easier and quite faster than generating the output of other categories.

In this paper, we focus on Static Random Access Memory (SRAM-PUF) [15] to be used in protecting ZigBee keys storage. SRAM-PUF output is generated by reading the Start-up Values of SRAM Cells (SVSCs) of the local devices. We choose SRAM-PUF because SRAM is already used as a fast memory in many electronic systems nowadays, and thus, there is no need to add extra components to benefit from the PUF. The key contributions of this paper are as follows;

- Two algorithms were proposed to overcome the defects in the randomness of keys generated using SRAM-PUF and, at the same time, to increase the reliability of these keys.
- We have proved that SRAM-PUF can effectively protect the keying materials of ZigBee devices whether by generating the secret keys or by securing the secret keys stored in the NVM of the local device with no need to install any new equipment.

The remaining part of this paper is organized as follows: Section II is a literature review; Section III covers SRAM-PUF experiments and improvements; Section IV is about securing ZigBee devices using PUF; Section V concludes our work and future work.

2. Related Works

SRAM-PUF was introduced by Guajardo et al. in [11] and Holcomb et al. in [15] where PUF randomness comes from the SVSCs. Each SRAM cell can represent one bit which could be 0 or 1. Choosing one of those initial states when the SRAM cell is powered on corresponds to the production process variations of its components, especially the MOS transistors, which makes the initial states random and unpredictable. Experimental results in previous researches [15] showed that the majority of the cells are more likely to start in the state "1". As a result, the inter-die Hamming Distance (HD), which is the average of aggregated differences between the bits from each tested memory and that from the same location in other memory chips, was 27.62%, and the intra-die HD, which is the average of aggregated differences obtained by reading the same memory cells for multiple times, was 4%. To solve the unbalanced 0:1 ratio of the SVSCs, some researchers [21] [8] have proposed hardware modifications to the SRAM structure. In this paper, we focused on using software solution instead of the hardware one for two reasons; first to keep the currently used devices without the need to change their local memories in order to be used as PUF; second, the IoT devices have microprocessors and can run some codes originated to mitigate the inequality of the 0s and 1s numbers and meanwhile keep the randomness and stability of PUF output. Therefore, we have proposed two algorithms to increase the inter-die HD, and meanwhile reducing the intra-die HD of the SRAM-PUF generated keys. After being fixed, the PUF output will be ready to be used for protecting the secret keys.

In other researches [7], Eichhorn et al. in order to protect the storage of secret keys, they suggested encrypting the whole external memory where the secret keys are stored. It is a good idea which will not only protect the secret keys but also protect the installed firmware. However, it is non-feasible for IoT small devices with limited resources, because it will take them long time to decrypt the whole firmware in order to keep the communication stack (ZigBee stack) running well. What is more, the stack is big to be decrypted and then runs from internal RAM. Therefore, from another point of view, we can just encrypt a small part of the memory which contains the secret keys and the sensitive data. That is exactly what we have done in this paper. We used PUF to encrypt the part of the memory, where the secret keys are stored, to make them safe and hard to expose or analyze.

Moreover, SRAM-PUF is used as a hardware key, so once the PUF is compromised, it will be invalid forever and the hardware chip, from which it is generated, should be replaced. To avoid such a situation, we need to protect the inputs and outputs of the PUF. There should be no direct access to the PUF unit except the board itself. Thus, "Controlled PUF" idea was proposed in [9] where all inputs and outputs of PUF are covered and cannot be accessed except by the processor. In this paper, we used SRAM, which is only connected to the processor and cannot be read outside the microcontroller, as the PUF. However, some parts of the RAM could be directly connected to input/output pins called Direct Memory Access (DMA), which can be read directly from the device outside. As a result, memory cells assigned as DMA cannot be used as PUF for the aforementioned security reason.

Even though PUF seems to be the best solution to hardware security issues, every PUF-proposed architecture has some physical-related problems still under discussion. Environmental conditions [4], such as power-supply voltage

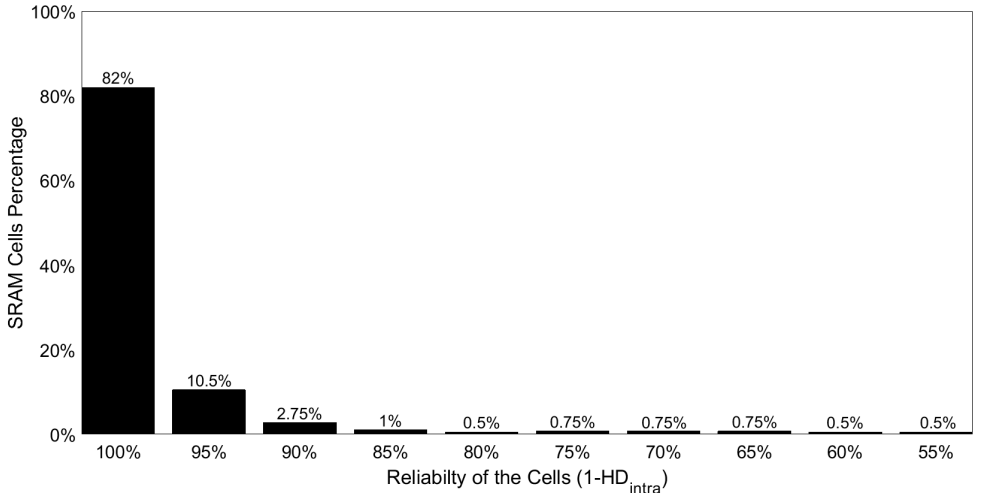

Figure 1: The Reliability of the SRAM Cells Represented by the Number of Cells Which Belong to the Same Reliability Category

changes and temperature variations, can also affect the PUF output. As a result, Error Correction Code (ECC) algorithms should be used to correct the noisy PUF output caused by these conditions. The more the cells need to be corrected, the larger the size of the generated helper data is required. Generating the helper data and the storage space needed to store it are the only extra requirements for deploying SRAM-PUF in the ZigBee security system.

To sum up, the ZigBee secret keys cannot be stored in NVM their plaintext format because that will threaten the whole system in case they are exposed. On the other side, the whole NVM cannot be encrypted to protect the secret keys especially in the case of limited-resource ZigBee devices since there is not enough RAM to decrypt the encrypted content and run the system simultaneously. Therefore, in this paper, we have proposed a low-cost, fast and secure technique to ensure the ZigBee hardware security using SRAM-PUF, which is already a part of the SRAM of the local device (no extra cost), by encrypting only the secret keys part or generating the required secret keys without the need to store them in NVM. In addition to increase the randomness of the keys generated using SRAM-PUF by proposing two algorithms which guarantee to keep on the PUF random output and generate stable and reliable secret keys.

3. The Proposed SRAM-PUF Based Secret Key Scheme

3.1. SRAM-PUF based Encoding Method

SRAM-PUF is used to generate a hardware key, and the experiment is done by using the ZigBee devices Texas Instruments CC2530 which have SRAM memory with the size of 256B. In order to calculate the reliability of SVSCs, we read the SVSCs 100 times for each device. Then, we calculate how often each individual cell start with 0 and 1. After counting the iteration rate of 0 and 1 value readings for each cell, we marke the cells as 0-biased or 1-biased depending on the most frequent reading value. i.e. If a specific cell starts as "1" with rate of 80% and as "0" with rate of 20%, it is marked as "1-biased" cell. At the end of the process, we are able to divide the memory cells into three groups:

- 0-biased cells which are more likely to start with 0;
- 1-biased cells which are more likely to start with 1;
- neutral cells which have no strong tendency to start with 0 or 1. In this case, the iteration rate of 0 or 1 is near to 50%.

As shown in Figure1, the 1-biased or 0-biased cells have different degrees of reliability. 82% of the cells are 100% reliable, which means about 1680 bits of 2048 bits (256B) in the memory always start as 1 when they are 1-biased and as 0 when they are 0-biased, 10.5% of the memory cells are 95% reliable which equivalents to 215 bits of 2048 bits and so on. The total reliability of the whole memory is 95.6% calculated by the equation 1-HD$_{intra}$ where HD$_{intra}$ is 4.4%.

Algorithm 1 The Initial Phase: Generating The Hardware Key

```
start ← suitableStartingAddress
reliabiltyRate ← %100
i ← 0 , j ← 0
while i < keyLength do
    if ReliabiltyOfCell[start+j] = reliabiltyRate then
        KeyBits[i] ← Memory[start+j]
        KeyCellAddresses[i] ← start+j
        i ← i+1
    end if
    j ← j+1
    if j = memorySize-start then
        reliabiltyRate ← reliabiltyRate-1
        j ← 0
    end if
end while
seedIndex ← KeyBits([0][1][2][3][4])
GenerateBCHhelperData(KeyBits)
```

This experiment is done on 20 ZigBee devices under the normal temperature 25 $C°$. According to the experimental results, ECC algorithms are needed to correct the SVSCs readings because some of the 1-biased cells could start as 0 or vice-versa for some reason related to environmental conditions or the reliability degree of cells. Changing one bit of secret key endangers the whole cryptography system, and since the secret hardware key generated using SRAM-PUF bits is 128-bit length, the reading error is expected in only up to 6 bits for the worst scenario based on the 95.6% reliability value obtained from the experiment. In this paper, we use BCH [14, 5] for correcting the noisy output. The more the cells need correcting, the larger the helper data size is required.

One practical solution to lower bit error rate and reduce the helper data size is to generate the key bits starting by the highly reliable cells until the device finishes generating the whole key. That is what we do in the Algorithm 1. This process runs on the manufacturer side where the manufacturer first excludes the unreliable cells from the key. To do that, the whole manufactured memories are scanned for multiple times, and then an array called *ReliabilityOfCell* is filled with the reliability rate of each memory cell. After that, the manufacturer chooses an address to start reading the key bits. This address can be any address, but it is preferred not to start with the first 32 bits (which are usually used by the CPU registers) or any bits that are located on the Special-purpose registers area. To sum up, it can be any address whose contents are not changed by the running program until the completion of key extraction. In our experiment, the starting address is 0x90. After the manufacturer chooses a suitable starting address, the algorithm begins to build the key starting by the 100% steady cells. *keyLength* in Algorithm 1 represents the secret key length which was 128 in our experiment because we are building ZigBee secret keys. If the 100% steady cells are not enough to generate *keyLength* bits, the algorithm moves to the 99% steady cells and so on. During the key building process, key bits' addresses are saved and stored in an array called *KeyCellAddresses*. When the whole key is generated, ECC algorithm which is BCH in this paper proceeds to generate the helper data which will be sent with the addresses of the key bits to the end user for correcting the generated secret keys in the future. There is also another value called *seedIndex* consists of the first five bits of the generated key and will be used later in Algorithm 2. There are many ways to deliver the helper data and the addresses to the end users, for example, they can be sent in form of a QR code stacked on the packing box or be saved on the NVM. There is no risk if the attacker accesses and obtains the helper data because the helper data itself is not enough to generate the secret key. The same is for the key bits' addresses since the attacker needs a map of SVSC of the end user device to build the secret key. By running Algorithm 1, we are able to reduce intra-HD of the generated key from 4.4%, to less than 2%. The experimental result is very close to the ideal value of intra-HD which is 0%.

The other issue discussed in details in this paper is the inter-HD value. The average difference between the SVSCs from a specific device and that from other devices is about 30.65%. Since the ideal value of inter-HD between the PUF units is 50%, the SRAM-PUF output needs to be modified in order to make the inter-HD as close as possible to the ideal value. The main shortage, which makes the values fail to meet the PUF required measurements, is that the majority of SRAM cells (about 78%) start with 1 and the rest (about 22%) start with 0. As a result, when a 128-bit hardware key

Algorithm 2 The Enrollment Phase: Randomizing The Hardware Key

1: **read** *KeyCellAddresses*
2: **for** $i = 0 \rightarrow$ *keyLength-1* **do**
3: *KeyBits[i]* $\leftarrow$ *Memory[KeyCellAddresses[i]]*
4: **end for**
5: *BCH(KeyBits)*
6: *seedIndex* $\leftarrow$ *KeyBits([0][1][2][3][4])*
7: *currentPos* $\leftarrow$ *seedIndex*
8: **for** $i = 0 \rightarrow$ *zerosConvertedNumber - 1* **do**
9: **if** *KeyBits[currentPos]=1* **then**
10: *KeyBits[currentPos]* $\leftarrow$ *0*
11: **else**
12: **while** *KeyBits[currentPos]* $\neq$ *1* **do**
13: *currentPos* $\leftarrow$ *(currentPos+1)* mod *128*
14: **end while**
15: *KeyBits[currentPos]* $\leftarrow$ *0*
16: *currentPos* $\leftarrow$ (*currentPos* + *seedIndex*) mod *128*
17: **end if**
18: **end for**

is purely extracted from the raw data of SVSCs, it has the same 0 to 1 ratio i.e. the extracted key contains 30 zeros and 98 ones of 128 bits. After getting the 0s to 1s ratio of the extracted keys, we used the programming language Python to generate files that contained a large number of secret keys with the same 0s number and 1s number relation. Then we calculated inter-HD between the generated secret keys which was 34.3979%.

Generally speaking, for each key of 128-bit length, about 44 bits are different comparing with other secret keys bits. This difference rate enables us to generate about $17.6 * 10^{12}$ distinct secret keys. Even though it is a very big number, we already know what invoke the randomness of secret keys. Therefore, Algorithm 2 is proposed to enhance the randomness degree of the key bits and reduce matching odds of secret keys. The core idea is to increase the number of 0s found in the key. The newly proposed Algorithm 2 works on changing some 1s into 0s in order to balance the number of 1s and 0s inside the hardware key, and meanwhile, it keeps the original 0s generated by SRAM-PUF. Algorithm 2 runs on the end-user side. After buying a new equipment, the user scans the QR code and programs the device to extract the key using the recorded addresses. As shown in Algorithm 2, the first step is to read the *KeyCellAddresses* generated by the manufacturer. This array contains the highest reliable memory cells addresses. Those addresses are used in generating the secret key by reading the corresponding memory values as shown in step 3. After that, the key is corrected by using BCH and the helper data which is generated by Algorithm 1. After completing step 5, we will get a correct and reliable hardware key. In the next steps, Algorithm 2 works on increasing the key uniqueness, so it changes some 1s into 0s. The number of 1s which needs to be changed into 0s is stored in *zerosConvertedNumber*. This number is equal to 28% of the *keyLength* according to the experiment results in this paper, after which the number of 0s is going to be equal to the number of 1s. Because we are generating a 128-bit secret key, 36 bits get affected, and thus *zerosConvertedNumber* is 36. The bit location of the affected cell is calculated by adding *seedIndex* to the last memory address location where the algorithm arrives at. For the *seedIndex*, it should be not only random to be robust against the outsider attacks, but also stable to enable the local device to obtain the same *seedIndex* whenever there is a need to generate the same modified key. In addition, choosing the locations of converted 1s should not be the same for all secret keys to avoid getting identical secret keys. To do that, *seedIndex* is calculated by getting the decimal equivalent to the first 5 bits of each generated key. As shown in step 6, *seedIndex* is generated from the SVSCs which makes it as a small SRAM-PUF output leading into reliable (the bits at the beginning of the key have the highest reliability rates) and random (its value depends on the production process variations). Since *seedIndex* is a part of the generated key, even if some of its bits flip, they will be corrected when running ECC algorithm to correct the generated key. After we have generated the *seedIndex*, everything is ready to start the randomizing process. The location of the target cell is stored in *currentPos*. Each target cell changes its value from 1 to 0. If the target cell already has 0 value, the algorithm searches for the next nearest cell which has 1 value. Algorithm 2 keeps running until it finishes converting *zerosConvertedNumber* cells. The final result after the completion of Algorithm 2 is a key where the number of 0s and

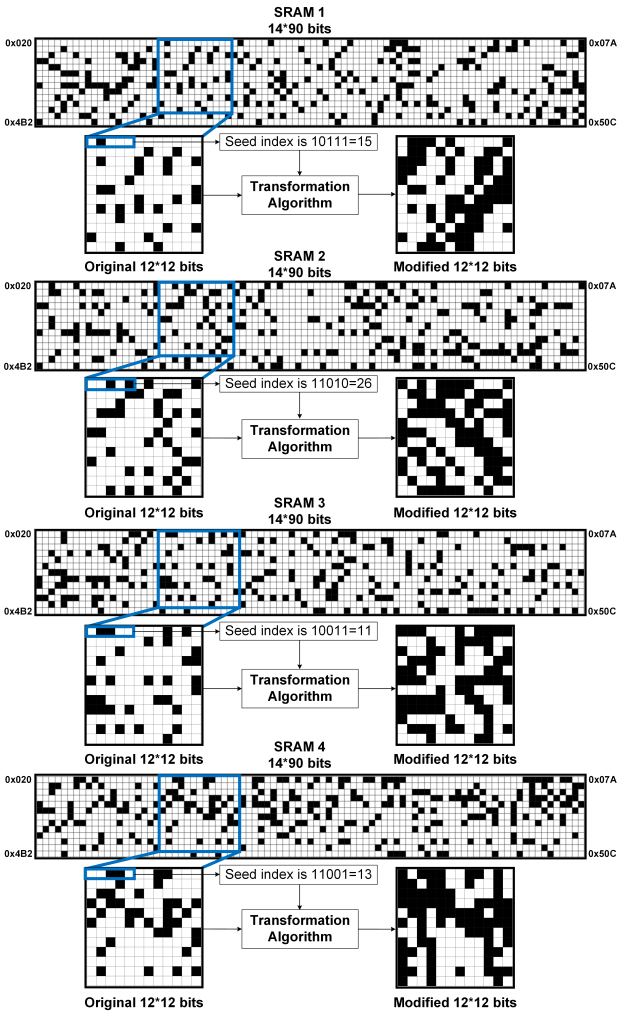

Figure 2: Real SRAM Samples with Their Original and Modified Start-up Values Before and After Running the Transformation Algorithm

1s are the same.

3.2. Results and discussion

We program and run the algorithms on real ZigBee devices. As shown in Figure2, we first read the whole SRAM memory and just show a part of the 1260 bits. The cells which tend to start with 1 are colored white, whereas the ones which tend to start with 0 are colored black. We use the same starting address for all devices. Then we choose, for the experiment, 144 bits by reading the same part of different memories. Before running Algorithm 2, most bits are 1s and thus many bits, compared with the ones of the same locations in other memories, are identical.

As shown in Figure2, "Original" is the name of a (12*12 bits) memory part before running the Algorithm 2, and "Modified" is the name of the same (12*12 bits) memory part after running the Algorithm 2. The number of 0s in the modified part increases, and we get less identical values of the cells compared with the ones of the same locations on other memories. The *seedIndex* used in Algorithm 2 to choose the locations of the converted bits is different for each memory section due to the SVSCs randomness. As shown in Figure2, it is 15, 26, 11, and 13 for SRAM1, SRAM2, SRAM3 and SRAM4 respectively. Even in the case that there is the same generated index for two different devices, the converted cells locations will not be the same after running the algorithm.

Algorithm 1 is needed to reduce the storage size of the helper data and to enhance the reliability degree of the generated key. Since it runs on the manufacturer side, it does not add any overhead to the limited resource devices

Table 2
key differences ratio

Keys number	Original	Modified
1000−10000000	34.39708852%	49.83605342%

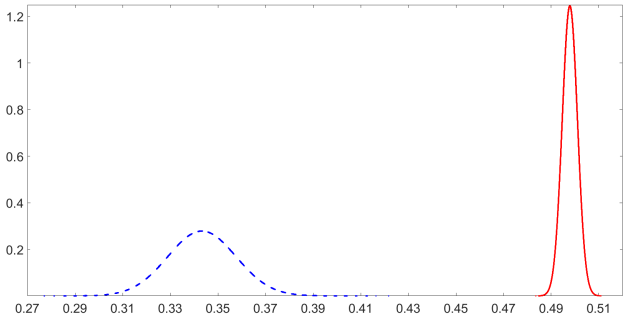

Figure 3: Inter-HD of the Secret Keys Before/After Running Algorithm 2

which are ZigBee End Devices (ZEDs) in this paper. The manufacturer can provide the end user with the highest reliable 128 cells' addresses to build a secret key or with more 128-bit combinations to give the end user the choice to choose any suitable secret key.

We also ran a simulation for generating secret keys randomly with the same real memory 0:1 ratio in order to evaluate Algorithm 2. To do that, we generated files with 10^3, 10^4, 10^5, 10^6, and 10^7 secret keys using Python. Then, we wrote a program to test the uniqueness of keys before and after running Algorithm 2. The inter-HD value was almost the same in all files. As shown in Table 2 and Figure3, the inter-HD of original secret keys was 34.4% (the dotted blue line), and after running the algorithm, the inter-HD had increased to 49.8% (the solid red line), which means Algorithm 2 is working efficiently.

Comparing with the other researches which aimed to mitigate the bias problem in SVSCs, we were able to significantly decrease the bias effect by applying two algorithms in software. The other researchers used hardware solutions such as adding a current source or voltage bias to the SRAM circuit in order to neutralize SVSCs. However, in the case of IoT and ZigBee devices, the software solution is more effective for two reasons; first, there is no need to change the SRAM architecture to be able to benefit from the PUF features; second, those devices have microcontrollers and can easily accept the software solution especially that Algorithm 2 is lightweight and runs only on the device booting.

4. Integrating SRAM-PUF with ZigBee Security System Model

As aforementioned in this paper, PUF can be used for securing ZigBee secret keys. When ZigBee network is working in high-security mode, the TC will list the network devices with their master and link keys. In addition, network devices will be pre-configured with at least a master key to be able to establish a link key and then securely acquire the active NWK key. When a ZED is not pre-installed with a master key, the active NWK key will be sent in plaintext to the newly joined device. If the plaintext NWK key is captured by an adversary, the whole network will be threatened. Furthermore, when a device receives the NWK key or any other secret keys, the secret keys will be stored in the NVM or in an external memory which also needs protection.

The process of deploying PUF in ZigBee architecture depends on what we are going to protect and in which way the secret keys are acquired. Therefore, some typical scenarios and their implementations were illustrated, and then PUF-based methods to generate pre-configured secret keys and protect the other secret keys stored in NVM were proposed.

4.1. The first scenario: Pre-configured Key

All main secret keys, including the master, the link and the NWK keys, can be pre-installed on the ZEDs before they are deployed in the network. There are two ways to pre-install the secret keys:

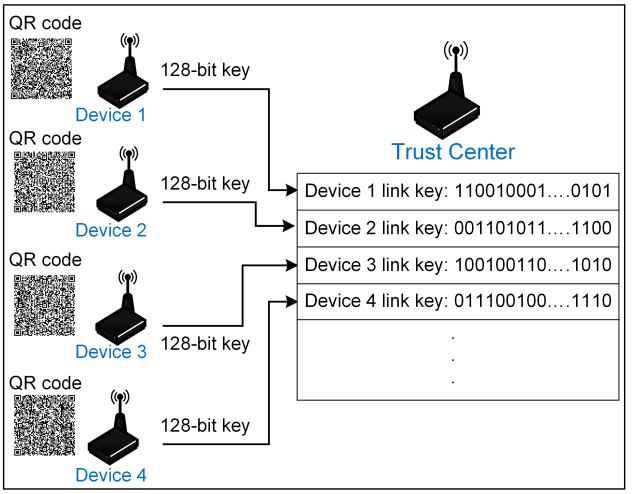

Figure 4: Installing Pre-Configured Secret Keys in TC

- The first way is to use the ZED's SRAM-PUF as a secret key which can be valid for both of the master and link keys but not for the NWK key. That is because the master and the link keys keep unchanged during the runtime whereas the NWK key can be updated. The master and the link keys belong to one pair of devices where each device shares a master and a link keys with the TC or with another device for securing the unicast communication between them.
- The second way is to generate the secret keys (master, link and NWK keys) by the network administrator and then store them in NVM of TC and ZEDs. In this case, the secret keys should be protected and encrypted inside the NVM using a hardware key generated by the SRAM-PUF of the local devices.

4.1.1. Using PUF in generating master or link key

There are two modes for the link key in ZigBee: global and unique. In the global link key mode, the TC stores one link key which must be installed in all the network devices. For unique link key mode, each device has its own link key, and the TC needs to store all these link keys of the ZigBee devices in its memory. Unique link key mode is more secure especially in case one of the ZEDs is stolen and the link key gets compromised. This problem can be solved by deleting the stolen device from the TC access list, and there is no need to take any further actions. Still, there is a disadvantage of using the unique link key mode that the memory grows with the number of joined devices. In the global link key mode, if the link key is obtained, the adversary will be able always to join the network unless the global link key changes. Changing global link key requires reinstalling of new link key for all network devices. However, PUF can be used in both cases regardless of the types of link keys. In unique link key mode, the link key between the ZED and TC will be the SRAM-PUF output of the ZED. In the global link key mode, the global link key will be the TC SRAM-PUF after the PUF output is remodeled using a Hash function or other algorithms. That is because the PUF raw output should not be compromised to the outside world.

The TC of the ZigBee network as previously mentioned contains a list of devices and their secret keys. The advantage of using SRAM-PUF as a device key is that when the device is powered off, there will be no key saved which minimizes the time window for key extraction attacks. Moreover, producing SRAM exactly the same for the purpose of obtaining the same SVSCs is impossible because SVSCs are random and cannot be controlled. Before ZEDs leave the factory, Algorithm 1 will be executed and a secret key, or maybe multiple secret keys, with its helper data will be generated. This key is the SRAM-PUF output of ZED, in other words, it is like a pre-installed key, so it could be used as a master or link key for this ZED. The manufacturer will send the addresses of the key bits to the end user using QR code, PIN code or any out-of-band communication method. As shown in Figure4, before installing the ZigBee network, the administrator will read the master or link keys of the ZEDs (e.g. scan the QR code and generate the key accordingly), add the ZEDs addresses and store them together in the TC access list.

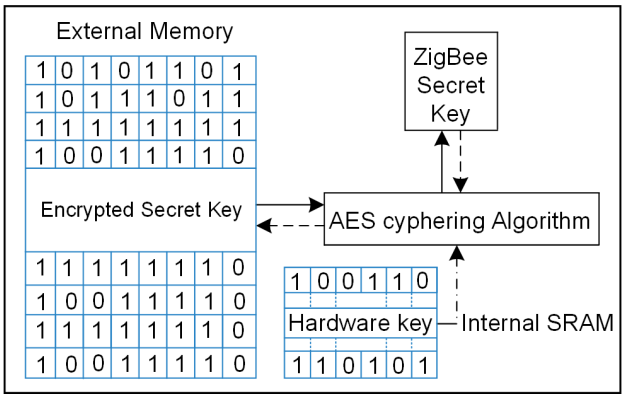

Figure 5: Protecting Secret Keys Stored Inside an External Memory

4.1.2. Using PUF in protecting the NWK key

The NWK key, on the other side, is used to protect all broadcast messages at the network layer level. It is a key used by the whole network devices, so it cannot be correlated to devices physical characters, so PUF is not valid to be used as NWK key. In addition, the NWK key can be updated during the runtime, so it is better to be saved in a memory and being protected there. PUF can be used to protect the NWK key when it is stored in the NVM. As shown in Figure5, the process is divided into two stages:

- Stage 1: 128-bit hardware key will be extracted using SRAM-PUF (or any other PUF types) from the local ZED SRAM and then stored in the internal SRAM. After that, this extracted key will be used to encrypt the pre-configured NWK key and store it in the NVM. Stage 1 is illustrated in Figure5 with dashed line.
- Stage 2: whenever the ZED needs to use the NWK key either for encrypting the outgoing frames or decrypting the incoming frames, it needs to decrypt the stored encrypted NWK key using its own hardware key. Stage 2 is illustrated in Figure5 with solid line.

By using this method, the secret key will always be protected. Even if some attacker was able to run side-channel attacks and read the NVM content, he would not be able to obtain the real NWK key to endanger the network. The encryption and decryption of the secret keys will not take a long time, especially that most of the ZigBee devices come with dedicated-hardware for AES cyphering operations.

4.2. The second scenario: Transport-Key

Another way, specified by the ZigBee Alliance, for acquiring the secret keys is by sending the keys over the air to ZEDs and then storing them in the memory of the receiving ZEDs. In the ZigBee networks where the ZEDs are not configured with pre-installed link keys, the NWK key will be sent in plaintext, which is very dangerous for the security of the ZigBee stack. However, in high secure ZigBee networks, the secret keys are protected using key-load keys when transferring master or link key and using the key-transport key when transferring NWK key. In this case, PUF cannot be used in generating the load-key or the transport-key key because, in ZigBee, both of them are derived from the link key after running keyed HMAC. nevertheless, the link key from which the load-key and transport-key key are derived could be the SRAM-PUF of the local device. In addition, PUF can be used for protecting the keys after they are received and stored in the ZED's NVM. The protection process is similar to protecting stored NWK key in NVM discussed earlier in this paper and shown in Figure5.

An example of using the shared link key between TC and ZED is acquiring the transported NWK key in the secure ZigBee network. As shown in Figure6, the newly joined ZEDs will ask the TC to send the active NWK key in order to be able to communicate with the other devices within the network securely. TC will send the active NWK key encrypted by the key-transport key which is derived from the link key of the joined device. The ZED will also re-generate its own link key using the key generation represented by Algorithm 1 and Algorithm 2. Then, it will derive a key-transport key for decrypting the incoming frame which carries the encrypted NWK key. After being obtained, the active NWK key will be stored in the NVM after being encrypted using the local SRAM-PUF.

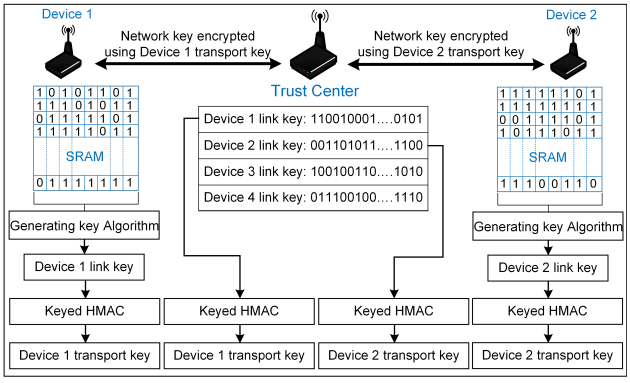

Figure 6: Acquiring Transported Network Key

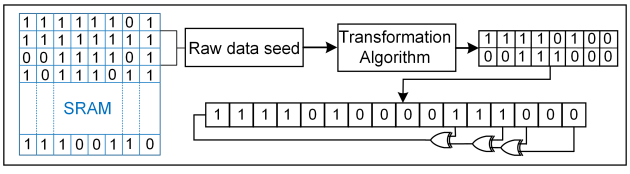

Figure 7: Generating Random Seed

4.3. Authentication and random number generators in ZigBee

Random numbers are required for ciphering and authentication operations in ZigBee. The authentication process in ZigBee is done by using Mutual Symmetric-Key Entity Authentication. In this process, each of the paired devices generates a random challenge and send it to the other paired device encrypted using the shared key. Despite these challenges having a minimum length and a maximum length, the challenge values should be random and non-repeated.

Randomness in ZigBee is also used during the ciphering operations which are done by using AES-CCM*. When encrypting the frame data, a generated random nonce is added to the frame data before the encryption process starts in order to change the generated cipher output. According to the great significance of random numbers for ZigBee security, ZigBee devices are usually provided with pseudo-random number generators. An example of random number generators used by some companies for ZigBee devices is the use of a linear feedback shift register (LFSR). In CC2530, which is used in this experiment, LFSR 16 bits is used. The only way to predict the output of the random number generating algorithm is to guess LFSR seed (the initial value of the register). Some companies use the random data from noise in the radio Analog-to-Digital Converter (ADC) as a seed. Nevertheless, PUF could be a good technique for providing the register seed with a random and unpredictable value. Obviously, there are two methods to do that; the first method is to use the neutral cells mentioned in section III whose reliability is near to 50%. Those cells are purely random and could start as 0 or 1 differently every time the SRAM starts up. The only shortage in this method is that the neutral cells represent just about 0.5% of the SRAM cells, and thus, the SRAM size required to obtain 16-bit length seed with 100% pure randomness is 3200 bits. The second method to generate the 16-bit seed, as shown in Figure7, is to read it out from SVSCs after passing through Algorithm 2 to minimize the biasing effect of SRAM-PUF as follows; firstly, 16 memory addresses of the most unreliable cells will be chosen. Then, Algorithm 2 will run to randomize the generated seed in case there are too many 1s. The *seedIndex* will be the first 3 bits, and the rest of the algorithm code will run as normal after modifying the number of 1s which are going to be turned into 0s.

5. Conclusion

IoT data occupies a large part of the big data system sources, and protecting this data on the infrastructure level is mandatory for the safety of the whole system. In this paper, we have presented PUF-based secret keys scheme to secure the memories in ZigBee physical devices by providing random, reliable and real-time generated hardware keys. PUF is considered a good solution for the devices with limited resources since it does not require any storage space and

consumes little power to generate its output. In this paper, SRAM-PUF has been used for two reasons: firstly, SRAM is already available in most of ZigBee devices, and thus, there is no need to install any new equipment. Secondly, most ZigBee devices have limited resources with no effective method to protect their own secret keys. In order to overcome the shortage in randomness of keys generated using SRAM-PUF and increase the reliability of these keys, two algorithms have been proposed and tested. The results show that the reliability has been increased to 95.6% and the Inter-HD of PUF-generated keys has been increased to 49.83%. As a result, SRAM-PUF can successfully be integrated with ZigBee stack and improve the hardware security of its devices in term of producing a robust security system.

6. Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant 61872038, 61811530335, and in part by the Fundamental Research Funds for the Central Universities under Grant FRF-BD-18-016A.

References

[1] Akriotou, M., Mesaritakis, C., Grivas, E., Chaintoutis, C., Fragkos, A., Syvridis, D., 2018. Random number generation from a secure photonic physical unclonable hardware module, in: the 1st International ISCIS Security Workshop, Springer, London, UK. pp. 28–37.

[2] Aliance, Z., 2012. Zigbee specification document 053474r20. Zigbee Standard Organisation .

[3] Armknecht, F., Maes, R., Sadeghi, A.R., Standaert, F.X., Wachsmann, C., 2011. A formalization of the security features of physical functions, IEEE. pp. 397–412. URL: http://ieeexplore.ieee.org/document/5958042/, doi:10.1109/SP.2011.10.

[4] Böhm, C., Hofer, M., 2012. Physical unclonable functions in theory and practice. 1st ed., Springer Science & Business Media.

[5] Bose, R.C., Ray-Chaudhuri, D.K., 1960. On a class of error correcting binary group codes. Information and control 3, 68–79.

[6] Chatterjee, U., Chakraborty, R.S., Mukhopadhyay, D., 2017. A puf-based secure communication protocol for iot. ACM Transactions on Embedded Computing Systems (TECS) 16, 67.

[7] Eichhorn, I., Koeberl, P., van der Leest, V., 2011. Logically reconfigurable pufs, memory-based secure key storage, ACM Press, Chicago, Illinois, USA. p. 59. URL: http://dl.acm.org/citation.cfm?doid=2046582.2046594, doi:10.1145/2046582.2046594.

[8] Garg, A., Kim, T.T., 2014. Design of sram puf with improved uniformity and reliability utilizing device aging effect, in: 2014 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE. pp. 1941–1944.

[9] Gassend, B., van Dijk, M., Clarke, D., Devadas, S., 2007. Controlled Physical Random Functions. Springer London. chapter chapter 14. pp. 235–253. URL: http://link.springer.com/10.1007/978-1-84628-984-2_14, doi:10.1007/978-1-84628-984-2_14.

[10] Gassend, B., Lim, D., Clarke, D., van Dijk, M., Devadas, S., 2004. Identification and authentication of integrated circuits. Concurrency and Computation Practice and Experience 16, 1077–1098. URL: http://doi.wiley.com/10.1002/cpe.805, doi:10.1002/cpe.805.

[11] Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P., 2007. Fpga intrinsic pufs and their use for ip protection, in: International workshop on cryptographic hardware and embedded systems, Springer. pp. 63–80.

[12] Günlü, O., Kernetzky, T., İş, O., Sidorenko, V., Kramer, G., Schaefer, R., 2018. Secure and reliable key agreement with physical unclonable functions. Entropy 20, 340. URL: http://www.mdpi.com/1099-4300/20/5/340, doi:10.3390/e20050340.

[13] Han, Q., Meikang, Q., Zhihui, L., MEMMI, G., 2019. An efficient key distribution system for data fusion in v2x heterogeneous networks. Information Fusion 50, 212–220.

[14] Hocquenghem, A., 1959. Codes correcteurs d'erreurs. Chiffres 2, 147–56.

[15] Holcomb, D., Burleson, W., Fu, K., 2009. Power-up sram state as an identifying fingerprint and source of true random numbers. IEEE Transactions on Computers 58, 1198–1210. URL: http://ieeexplore.ieee.org/document/4674345/, doi:10.1109/TC.2008.212.

[16] Moradi, A., Oswald, D., Paar, C., Swierczynski, P., 2013. Side-channel attacks on the bitstream encryption mechanism of altera stratix ii: facilitating black-box analysis using software reverse-engineering, in: Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, ACM, Monterey, California, USA. pp. 91–100.

[17] Psychoula, I., Singh, D., Chen, L., Chen, F., Holzinger, A., Ning, H., 2018. Users' privacy concerns in iot based applications, in: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), IEEE. pp. 1887–1894.

[18] Qiu, H., Noura, H., Qiu, M., Ming, Z., Memmi, G., 2019. A user-centric data protection method for cloud storage based on invertible dwt. IEEE Transactions on Cloud Computing .

[19] Qiu, H., Qiu, M., Memmi, G., Ming, Z., Liu, M., 2018. A dynamic scalable blockchain based communication architecture for iot, in: International Conference on Smart Blockchain, Springer. pp. 159–166.

[20] Sadeghi, A.R., Naccache, D., 2010. Towards Hardware-Intrinsic Security. 1st ed.. Springer Berlin Heidelberg. Information Security and Cryptography.

[21] Shifman, Y., Miller, A., Keren, O., Weizmann, Y., Shor, J., 2018. A method to improve reliability in a 65-nm sram puf array. IEEE Solid-State Circuits Letters 1, 138–141.

[22] Škorić, B., Tuyls, P., Ophey, W., 2005. Robust Key Extraction from Physical Uncloneable Functions. Springer Berlin Heidelberg. volume 3531 of *Lecture Notes in Computer Science*. chapter chapter 28. pp. 407–422. URL: http://link.springer.com/10.1007/11496137_28, doi:10.1007/11496137_28.

[23] Tang, Q., Zhou, C., Choi, W., Kang, G., Park, J., Parhi, K.K., Kim, C.H., 2017. A dram based physical unclonable function capable of

generating> 1032 challenge response pairs per 1kbit array for secure chip authentication, in: Custom Integrated Circuits Conference (CICC), 2017 IEEE, IEEE. pp. 1–4.

[24] Zhou, W., Jia, Y., Peng, A., Zhang, Y., Liu, P., 2018. The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved. IEEE Internet of Things Journal .

Fadi Farha received his Master degree and currently working toward Ph.D. degree in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research interests include Physical Uncolnable Function (PUF), Smart Home, Security Solutions, ZigBee, Computer Architecture and Hardware Security. He is a student member of IEEE. E-mail: fadi_farha@ieee.org

Huansheng Ning is a professor and vice dean of the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research focuses on the Internet of Things and general cyberspace. He is the founder of the Cyberspace and Cybermatics International Science and Technology Cooperation Base. His research interests include Cybermatics, Internet of Things, Cyber-Physical Social Systems. E-mail: ninghuansheng@ustb.edu.cn

Hong Liu received her PhD degree from the School of Electronic and Information Engineering, Beihang University, China. she is working at the School of Computer Science and Software Engineering, East China Normal University, China, and also with the Shanghai Trusted Industrial Control Platform Co., Ltd, China. She focuses on the security and privacy issues in radio frequency identification, vehicle-to-grid (V2G) networks, and internet of things. She is a member of the IEEE. E-mail: liuhongler@ieee.org

Laurence Tianruo Yang received PhD degree in computer science from the University of Victoria, Canada. He is a professor in the Department of Computer Science, St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous computing, and big data. His research had been supported by the National Sciences and Engineering Research Council and Canada Foundation for Innovation. E-mail: lyang@stfx.ca

Liming Chen is a Professor of computer science in the School of Computer Science and Informatics, De Montfort University. He has worked as a Senior Research Fellow in the School of Electronics and Computer Science, University of Southampton, and a Lecturer, a Senior Lecturer, and a Reader in the School of Computing and Mathematics, University of Ulster. His current research interests include intelligent systems, pervasive computing, activity modeling and recognition, personalization and adaptation, smart environment, and their application in ambient-assisted living. He has served a Guest Editor for ten journal special issues and six books, and an Associate Editor for the IEEE TRANSACTIONS ON HUMAN MACHINE SYSTEMS. E-mail: liming.chen@dmu.ac.uk.

View publication stats