Implementation of a Broadcast Authentication Mechanism in ZigBee

Ji-Tsong Shieh, Li-chun Ko
Networks and Multimedia Institute, Institute for Information Industry
6FL., 218,Tun-Hwa S. Rd., Sec. 2, Taipei, 106, Taiwan, R.O.C.
{sjt,lcko}@iii.org.tw

Abstract

In this paper, we implement a broadcast authentication mechanism for the coordinators defined in the ZigBee standard [1]. The job of a Zigbee coordinator is to management a Zigbee network which usually consists of a large number of wireless sensors. A coordinator has the privilege to decide which device could stay in the network and which device should leave and thus it plays an important role in the network. In the original Zigbee standard, it supports basic data encryption / decryption abilities based on AES-CCM mechanism [3]. It also provides a basic set of security functions such as key management and key distribution. However, the original security in Zigbee lacks of the ability to perform secure broadcasting. In this case, a malicious node in the network would have a chance to forge a malign broadcast message and hurt the network. To overcome this, we propose a mechanism to add a modified one-way signature in the original broadcast frame for authentication. Through the help of our mechanism, we can enhance the security level of ZigBee. This enhancement is also vital to any application which needs to use broadcasts as a means to transmit its global commands.

1. Introduction

ZigBee/IEEE 802.15.4 is a wireless communication protocol defined for wireless sensor networks. In the physical layer, it uses DSSS (Direct Sequence Spread Spectrum), and supports 868 MHz, 915 MHz and 2.4 GHz radio frequency with up to 20kbps, 40kbps and 250kbps raw data rate respectively. It also supports different kinds of network topologies, including point-to-point, mesh and star networks. It is a low-cost, v low power consumption, two-way, wireless communications standard. Solutions adopting ZigBee can be embedded in consumer electronics, home and building automation, industrial controls, PC peripherals, medical sensor applications, toys and games.

Although the Zigbee standard has already been opened to the public, it is still incomplete in many aspects. For example, it doesn't support broadcast authentication. This means, if there is an invader existing in the network, it can forge a malicious message and perform spoofing attack easily. According to the original standard, the ZigBee coordinator is responsible for network formation, operation channel selection, key dispatching, data routing, and the management of the entire network. Since the ZigBee coordinator is crucial to the network, it is no doubt that we have to protect its packets from being attacked by a malicious node. And one of the most important things is to protect its broadcast messages.

As we mentioned before, Zigbee doesn't support broadcast authentication. To reinforce this, we adapt a one-way signature which comprises the following steps. First, the coordinator distributes a public key to every node. Second, when the coordinator wants to send a packet, it will first generate a signature with the payload, and then transmit it to the destination. Third, when other nodes receive the broadcast packet, they will perform signature verification and make sure the data is valid. Of course, this process is also protected by the AES-CCM defined in ZigBee. So the whole operation is secure and we don't need to afraid that someone listens in the radio channel.

2. Broadcast authentication algorithm

Our broadcast authentication algorithm was inspired by two previous works: HORS [4] and Merkle Hash Tree [8] [9] [10]. HORS [4] is a one-time signature scheme using one way function. It was proposed by *Leonid Reyzin* and *Natan Reyzin* in 2002. In HORS, it includes three phases: Key Generation, Signing, and Verifying. Merkle Hash Tree is a binary tree constructed from leaves. It can reduce storage requirement. In the following section, we will give a brief introduction to both of them.

2.1. HORS schemes

HORS [4] is a one-time signature scheme which improves BiBa [7] scheme. The name "HORS" comes from "Hash to Obtain Random Subset". The complete scheme is illustrated in Figure 1. The advantages of HORS including extremely efficient signing, one time evaluation of the cryptographic hash function, efficient verifying, and requiring one evaluation of the hash function in addition to k evaluations of the one-way function f. HORS also requires only one call to the hash function H. In addition, a small adjusting in its parameters will only results will not affect its security level significantly [7].

In our approach, we also use HORS scheme to sign data packets. When applying in the ZigBee network, the coordinator needs to generate a private key and a public key. After the key generation, the coordinator sends its public key to every node. After this, when the coordinator needs to transmit important messages, the coordinator can encode them with a signature. And when child nodes receive a packet, they should perform the Verifying procedure.

In HORS, we only need one key to sign every packet. But the draw back is that the security strength decreases in an inverse proportion of signing times with an identical key. As a result, we assume that a malicious node obtains signature on r messages of its choice, and then it tries to forge a signature on any new message m of its choice. We can calculate the probability that the adversary is able to do so without inverting the one-way function f. It is quite easy to see that this probability is no more than $(rk/t)^k$ for each invocation of Hash, i.e., the probability that after rk elements of T are fixed, k elements chosen at random are a subset of them. In other words, we get $k(\log t - \log k - \log r)$ bits of security. Thus, when we use k = 16, t = 1024 and r = 1, then the security level is 2^{-96} (A smaller value corresponds to a higher level of security). But after four signatures are given, the probability of forgery is equal to 2^{-64}. This shows the security level will decrease as the times of signature increases. In order to solve this problem, we use re-keying mechanism to reset the security level (to its original level).In the later section, we will discuss how to select the parameters which is suitable to the ZigBee protocol. For more information about HORS, please refer to related documents.

Key Generation

Input: Parameters l, k, t

Generate t random l-bit strings $s_1, s_2, \ldots, s_t$

Let $v_i = f(s_i)$ for $1 \le i \le t$

Output: PK = $(k, v_1, v_2, \ldots, v_t)$ and SK = $(k, s_1, s_2, \ldots, s_t)$

Signing

Input: Message m and secret key SK = $(k, s_1, s_2, \ldots, s_t)$

Let h = Hash(m)

Split h into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each

Interpret each h_j as an integer i_j for $1 \le j \le k$

Output: $\sigma = (s_{i1}, s_{i2}, \ldots, s_{ik})$

Verifying

Input: Message m, signature $\sigma = (s'_1, s'_2, \ldots, s'_k)$, and public key PK = $(k, v_1, v_2, \ldots, v_t)$

Let h = Hash(m)

Split into k substrings $h_1, h_2, \ldots, h_k$, of length $\log_2 t$ bits each

Interpret each h_j as an integer i_j for $1 \le j \le k$

Output: "accept" if for each j, $1 \le j \le k$, $f(s'_j) = v_{ij}$; "reject" otherwise

Figure 1. HORS one-time signature scheme

Parameter f is a one-way function and Hash is a hash function. Both f and Hash may be implemented using a standard hash function, such as SHA-1 or RIPEMD-160. Suggested parameter values are l = 80, k = 16 and t = 1024, or l = 80, k = 20 and t = 256.

2.2. Merkle Hash Tree

Intuitively, a one way function F is a function which is easy to compute its output value but hard to use its output to compute its input. Through the help of this kind of functions, we can encode a large input into a smaller output.

Merkle hash tree [8] [9] [10] is a popular hash function. We can use it as a one-way hash function to speed up our authentication process. In a Merkle hash tree, it grows from leave nodes. The main purpose we use Merkle hash tree as a one way function is to authenticate signatures easily and faster. Given a vector of data items $\underline{Y} = Y_1, Y_2, \ldots Y_n$, we can quickly authenticate a randomly chosen Y_i which only has modest memory requirements as shown in Figure 2.

To authenticate Y_i, we can apply the "divide and conquer" technique. Function $H(i,j,\underline{Y})$ is defined as follows [9] :

1.) $H(i,i,\underline{Y}) = F(Y_i)$
2.) $H(i,j,\underline{Y}) = F(<H(i,(i+j-1)/2,\underline{Y}), H((i+j+1)/2,j,\underline{Y})>)$

To begin with, we randomly generate each Y_i for secret and evaluate leaves by definition 1. For example, $H(5,5,\underline{Y}) = F(Y_5)$. Next, we construct a complete binary tree by definition 2. Each internal node will be computed from its children. Using this method, only $\log_2 n$ transmissions are required. A sender will send a secret data Y_i and an authentication path to a receiver. For example, when a sender A wants to send a packet to a receiver B, besides original data, A sends additional data Y_1, $H(2,2,\underline{Y})$, $H(3,4,\underline{Y})$, $H(5,8,\underline{Y})$ to B. We assume A already constructed a Merkle hash tree on local memory and had sent the tree root as public key to every node. Now B knows Y_1, so it can compute $H(1,1,\underline{Y})$. Also, B knows $H(1,1,\underline{Y})$ and $H(2,2,\underline{Y})$, so it can compute $H(1,2,\underline{Y})$. And trivially, B knows $H(1,2,\underline{Y})$ and $H(3,4,\underline{Y})$, it can compute $H(1,4,\underline{Y})$. Repeat the process, finally B can compute $H(1,8,\underline{Y})$ and compare it with the tree root received before to authenticate this packet.

Since A reveals each Y_i (maybe used before) in every outgoing packet, the security level decreases in every signing. For this reason, we should construct a new Merkle hash tree periodically. On the other hand, when we have enough memory space, we should always pre-compute all authentication paths for saving all the intermediate computations. Nevertheless, it's a trade-off between performance efficiency and memory cost.

2.3. An Improved One-time signature approach

We use a scheme that combines HORS and Merkle hash tree as a one-time signature [5] [11]. The sender generates the private key and public key. The private Key consists of *t* random numbers called *private balls*. We use private balls to construct a Merkle hash tree, and make root a public key (As shown in Figure 3.).

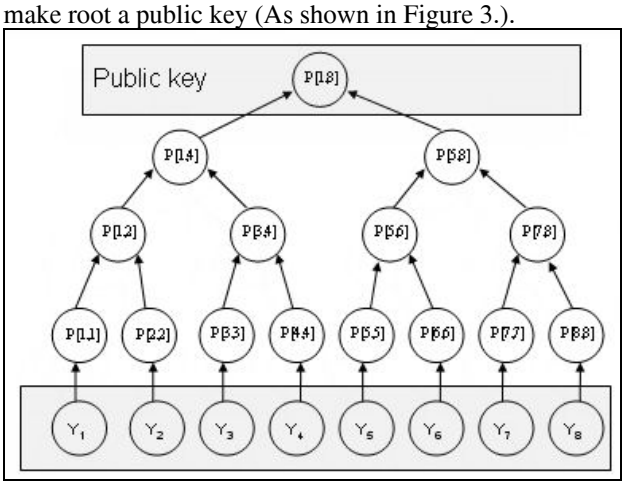

Figure 2. Merkle Hash Tree

We will choose some private balls $Y_1, Y_2, \ldots, Y_n$ as private key. It constructs from leaves by a hash function. Each internal node is computed by its children. The root will be a public key.

We generalize our scheme by first constructing many small Merkle hash trees of height *h* that hold 2^h private balls. The public key contains the root nodes of all the Merkle hash trees, and thus reduces the key size by a factor of 2^h.

In the following sections, we will explain the detailed steps of our mechanism. All the parameters we need are summarized in Table 1. The proposed scheme also has three phases as same as HORS; they are Key generation, Signature generation and Signature verification.

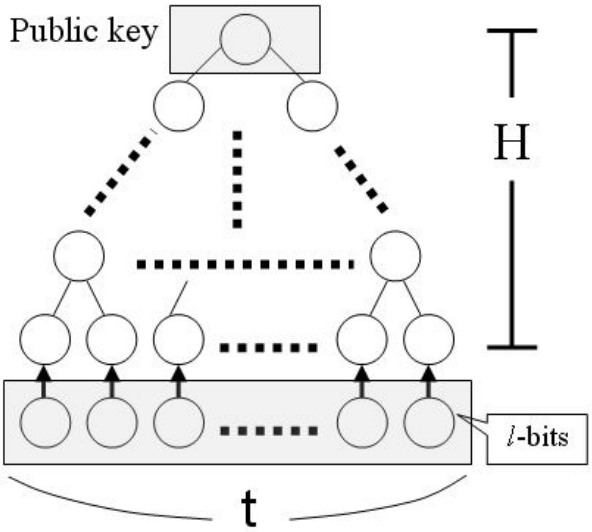

Figure 3. Key generation procedure

Table 1. System Parameters

h	cost of computing a hash function
t	number of private balls
k	number of signature balls
h_1	size of private ball's identity
f_l	size of a public ball
d	number of public balls
l	size of a ball (bits)
r	number of messages one key pair can sign

2.3.1. Key generation

In our proposed scheme, we separate a Merkle hash tree into small ones to reduce the overhead results from added signature. For example, we can separate a big Merkle hash tree which consists of 15 nodes into two smaller Merkle trees each consists of 7 nodes. The number of public balls d can also be generated by Merkle trees. Separated public balls are merged into a single public key. We should first define the number of private balls t, and the size of a ball is l in bits. Next, we define value of d and we have to decide number of signature balls k. These parameters will influence each other. The complete procedure of key generation is presented in Figure 4.

KEY GENERATION
Input: parameters t, k, d, l
Output: key pair
Private Key $K_{pri} = (s_1, s_2, \dots, s_t)$
Public Key $K_{pub} = (v_1, v_2, \dots, v_d)$
1. Randomly generate t l-bit random numbers(s_1, s_2, … , s_t)
2. Construct Merkle tree from leaves, v_i is the root of every Merkle tree
3. Distribute public key

Figure 4. Key generation algorithm

2.3.2. Signature generation

To sign a message m, we need first to compute h = H(m). Then, we separate the hash value h into k substrings h_1, h_2, …, h_k, and interpret each h_j as an integer i_j for $1 \le j \le k$. We use these integers as indexes of private balls. We pick k private balls and use them, along with their associated authentication paths, as the signature of this message m. This is presented in Figure 5.

SIGNATURE GENERATION
Input: message m and K_{pri}
Output: signature $\sigma = (a_{i1}, a_{i2}, \dots, a_{ik})$,where $a_i = (s_i, ap_i)$ (ap is the authentication path of the ball)
1. Compute the hash h = H(m)
2. Split h into k pieces (h_1, h_2, …, h_k)
3. Interpret each h_j as an integer i_j, with $1 \le j \le k$

Figure 5. Key generation algorithm

2.3.3. Signature verification

When a receiver obtains a broadcast message, it has to verify it. First, the receiver should compute h = H(m). Then, the receiver separates h into k pieces as the same as key generation. And the receiver tries to use signature that contains some private balls and corresponding authentication paths to compute hash tree root. The receiver checks every authentication paths and finally compares tree hash root and public key. Third, if all are valid, then output verified. We present signature verification procedure in Figure 6.

SIGNATURE VERIFICATION
Input: message m, signature σ, and K_{pub}
Output: {true, false}
1. Check if m is in current sequence period
2. Compute the hash $h = H(m)$
   Split h into k piece pieces $(h_1,h_2,\cdots,h_k)$ of length $ln\ t$ bits each Interpret each h_j as an integer i_j, with $1 \le j \le k$
   Compute $TN_j = i_j/(t/d)$
   Check i_j with pairs $(i, TN, H(AP))$
   If index i_j already exists,
      check if $H(AP_j) = H(AP)$
   Else check that each $H(AP_j) \neq H(AP) \in TN_j$
3. Use Merkle tree to verify balls
   If $(TreeHash(r_j, AP_j) = P_{TNj}$
      then output true;
   Else output false;

Figure 6. Signature verification algorithm

3. Implementation

In this paper, we implement our mechanism on ZigBee protocol stack, and see if there is any problem or bottleneck. ZigBee is based on IEEE 802.15.4 which defines a low-rate wireless personal area network. From the network's view, 802.15.4 belongs to PHY / MAC layer, and ZigBee contains NWK layer and all application layers above them. ZigBee supports simple security called AES-CCM* [1]. It's a symmetric-key security system. CCM* is a generic combined encryption and authentication block cipher mode. Each layer is responsible their own security by applying CCM*. The

length of an 802.15.4/ZigBee packet is maximum *128* bytes. It contains very short payload. In other words, an 802.15.4/ZigBee packet can't contain too long signature. It limits our implementation. The signature length of this mechanism contains "k private balls" and their authentication paths, i.e.,

SIGSIZE = BALLSIZE * HEIGHT * k

We assume that we have 16 private balls and each ball length is 2 bytes, and we have 4 public balls. That means we have 4 small trees, and each tree has Height = 3 and contains 4 leaves. We present it in Figure 7. Thus the size of signature is 2 * 3 * 4 = 24 (bytes). Although it doesn't seem to long, it has taken great proportion of packet. Get rid of frame header and some extra information, the length that can be used is turned into less than 30 bytes.

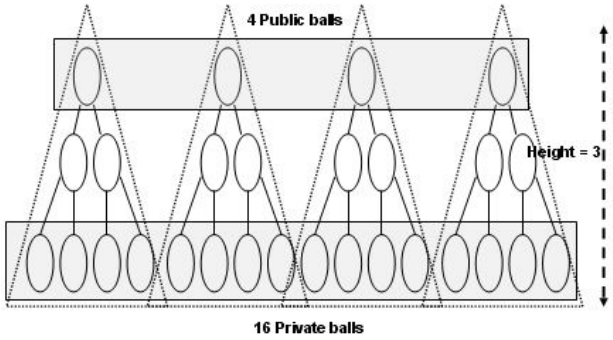

Figure 7. Implementation settings

This is a critical issue; because the security strength will be reduced with a shorter signature. The length of the signature presents a probability which can be randomly guessed. If we want a higher level of security, we must increase the length of stamped signature. But on the other hand, this will result in a shorter payload. So we need to find a balance between them. Now, please recall that the probability of forgery is

$$P = (rk/t)^k$$

It means that the security level will be decreased when the number of outgoing packets increases. Applying this formula, our settings of the experiment are k = 4, t = 16. When we send the packet at first time, $P = (1*4/16)^4 = 2^{-8}$. And after four signatures are given, the probability becomes $(4*4/16)^4 = 1$. Unfortunately, it doesn't seem very secure. Nonetheless, if we increase value of t, the length of the signature will be so long such that it can't be filled in a single ZigBee payload. As a result, this is not feasible. In the next section, we'll start to implement our mechanism and try to find a solution.

3.1. Hardware description

We select Chipcon CC2420DBK [12] as our operating platform (Figure 8.). It contains a RF chip, MCU, 2 buttons, one joystick, a RS-232 port and a temperature sensor. The MCU used is an AVR Atmega 128L from Atmel. This controller has 128 KB of Flash program memory, 4KB of SRAM data memory and 4KB of non-volatile EEPROM data memory. For a sensor system requirement, it meets the condition.

A JTAG ICE connector is provided for programming the AVR without using the serial port. The power supply section contains two voltage regulators: a 3.3 V regulator for use by the microcontroller and the I/O pins of the CC2420. So we usually use a DC as our main power. It also contains 4 LED; we can use these LEDs to debug easily. There are 2 buttons, one for functionality, and the other for reset. A joystick offers four directions and one click for functionalities. With this, we can design more complex operations. In actual situation, we select one board as the coordinator, and four boards as end-devices. And the topology of the network is star.

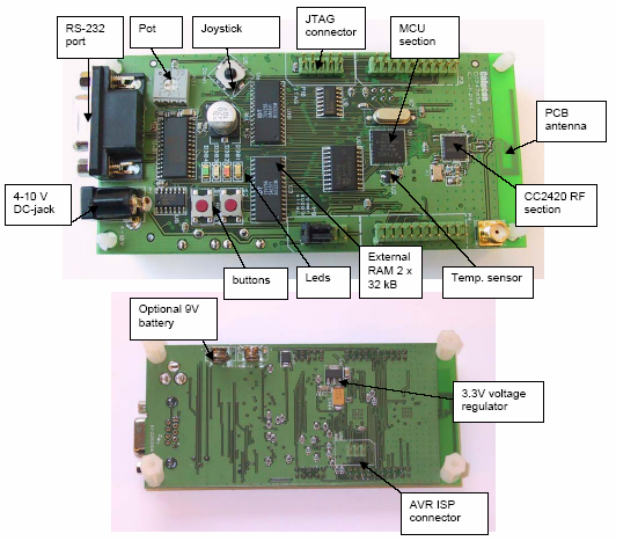

Figure 8. CC2420DB Demo board [12]

3.2. ZigBee stack IZAP

Institute for information industry (III) has been developing ZigBee stack for years. We have completed a ZigBee protocol library corresponding to Zigbee standard version 1.0. The library we developed is called III ZigBee Advanced Platform (IZAP) protocol stack. It passed Zigbee compliant platform (ZCP) certificate in late 2005. Since IZAP implements the whole ZigBee spec, it contains basic security. In ZigBee protocol, the ZigBee coordinator will form a network first, and other nodes start to join. In the procedure of joining, the

coordinator decides if a node is allowed to join or not. If the coordinator allows this node to join, it will dispatch a key called "network key" to that node. Only after receiving network key do the nodes can transmit their packets. And all transmissions are protected by AES-CCM*. The sender encrypts transmitting packets and the receiver decrypts them.

3.3. System Implementation

In the ZigBee spec, it describes a hash function called MMO hash (Matyas-Meyer-Oseas). It is based on AES and it's a block cipher with fixed length 16 bytes. We also link IZAP library to support more complex applications.

As we said that before, we only have limited space in the data payload. To overcome this, we came up a different approach. We first fragment a single message in to a number of packets in the sender side. And then, we reorganize these packets and verify them in a single run. In the original ZigBee spec, it doesn't describe how to fragment packets. But we can refer some methods from other network protocols. (For example, IEEE 802.3.) We apply a simple fragment mechanism on the original protocol. That is to cut original data into several fixed length packets and to add additional indexes to each of them. Since we know the probability of forgery is related to *r*, *t*, *and k*. If we extend *t*, we will get more high security. So we reset system parameters. Let each ball size = 1 bytes, t = 64, k = 4. Then we have four trees that each tree contains 16 leaves and height = 5. After changing parameters, the length of signature becomes

SIGSIZE = 1 * 5 * 4 = 20 (bytes)

And probability of forgery becomes

$P = (1*4/64)^4 = 2^{-16}$ (at first time)

We slightly modify original design. The receiver collects many times and does verifying once. That means we need a simple authentication path transmitting scheduler. We transmit private balls and some (not all) of authentication paths. Please recall that a Merkle hash tree presents every private ball's authentication path. Thus, there are some duplicate authentication paths. For example, if we send private ball Y_1, then it's authentication path is H(2,2,Y), H(3,4,Y), H(5,8,Y). If we send private ball Y2, then it's authentication path is H(1,1,Y), H(3,4,Y), H(5,8,Y). Thus H(3,4,Y) and H(5,8,Y) are duplicate.

Now, we can extend our signature to a higher degree of security. Typically, we send half of original authentication paths. And remained authentication paths will store in a queue and wait for being sent in the next transmission. Although this results in a longer stamped signature, we can increases the security level. The drawback of this approach is that we need to collect all the verification information. In this case, it is no longer a one-time signature, and we also need to consider the issues of time and possible loses of the intermediate packets**.**

Now, we try to find a suitable solution for the ZigBee protocol. Suppose that each ball size *l* decides the security strength against a brute-force attack. Although HORS recommends *l* = 80, we don't have enough space in a single payload to satisfy it. So we select *l* = 16. Though it seems very short, we can combine re-keying scheme to recover this drawback. We list all possible parameter settings on a table, and try to find a suited setting. They are presented in table 2. The parameter *t* means the numbers of private ball, and *k* means selected private ball as described before. The parameter *r* means that we sign maximum *r* times, usually satisfies (*rk*/*t*) = 1/2 that presents the security strength decreases by increasing *r*. The parameter *security level* presents the security strength of this setting. The parameter *TH* means each Merkle hash tree height. *AP #* is the number of authentication paths of one private ball. *SIGSIZE* is the size of the signature. *Only with 2 AP* means that every sending and each private ball only contains two authentication paths. *Only with 3 AP* has the same meaning.

We have two constraints of our solution. First, we don't want our signature length more than 40 bytes. In this case, we only send two authentication path (AP) or three authentication path corresponding to a private ball on each transmission. If we select *k* private balls, there are also *2k* or *3k* authentication paths. So the size of signature with 3 ap can be computed as:

2 (bytes) * (1(private ball) + 3(ap)) * k

Second, we wish we have a certain degree security level at least more than 20, and we want a bigger value r that we can use this signature more times.

So, finally, we can know that the best setting is *t* = 1024, *k* = 4, *r* = 128, security level = 32, tree height = 9, original AP # followed a private ball = 8, and we choose *only with 3 AP*. By these settings, we get a signature with 32 bytes. But this modified scheme needs to change slightly. Because every receiver needs to collect enough authentication paths to verify receiving signatures, they need more memory space to buffer data and they can't deal receiving data immediately. If the sender doesn't have any data to send, it also need to send some dummy data or remained authentication paths to let other nodes verifying remained signature. In our scheme, original AP # = 8, and each private ball follows only with 3 AP. That is, at first, the receiver needs at least 3 packets to collect

enough AP to verify. By time passing, the receiver needs fewer and fewer AP to verify.

We can add a re-keying scheme to our mechanism. After *r* times transmitting, the sender completes to send remained authentication paths. Then the sender decides to change a new key and transmit it to every node. During updating key, the sender can't send any broadcast. The new key also needs to be protected by broadcast authentication.

3.4. Analysis

In this section, we analyze the degree of security and the required size of transmitting payload of our choice. We set a private ball size = 2 bytes (16-bits). So we can defend 2^{-16} brute force attack for guessing each private ball. And the probability of forgery = $(rk/t)^k$, that means we have $p = (4/1024)^4 = 2^{-32}$, and after 128 sending, it becomes $p = (4*128/1024)^4 = 2^{-4}$. The security strength decreases by r increasing. That's why we need to use re-keying mechanism.

Table 2. parameters analysis

set #	t	k	times(r)	security level	TH	AP #	SIGSIZE	only with 2 ap	only with 3 ap
1	32	2	8	8	5	4	20	12	16
2	32	4	4	12	4	3	32	24	32
3	32	8	2	16	3	2	48	48	64
4	64	2	16	10	6	5	24	12	16
5	64	4	8	16	5	4	40	24	32
6	64	8	4	24	4	3	64	48	64
7	128	2	32	12	7	6	28	12	16
8	128	4	16	20	6	5	48	24	32
9	128	8	8	32	5	4	80	48	64
10	256	2	64	14	8	7	32	12	16
11	256	4	32	24	7	6	56	24	32
12	256	8	16	40	6	5	96	48	64
13	512	2	128	16	9	8	36	12	16
14	512	4	64	28	8	7	64	24	32
15	512	8	32	48	7	6	112	48	64
16	1024	2	256	18	10	9	40	12	16
17	1024	4	128	32	9	8	72	24	32
18	1024	8	64	56	8	7	128	48	64
19	1024	16	32	96	7	6	224	96	128

If we apply this settings we will produce a *32*-bytes signature. As described before, the payload length of ZigBee packet is *128*-bytes. Now we eliminate the header (about *30*-bytes), the additional data (about *20*-bytes) for ZigBee security, the signature added (*32*-bytes), we only have about *40*-bytes left to use for payload. This is illustrated in Figure 9.

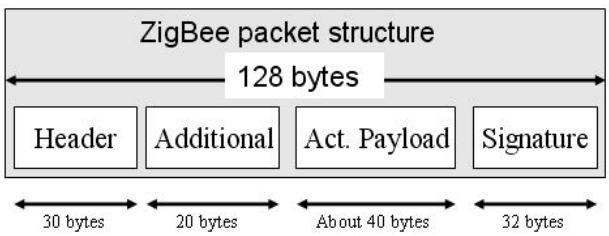

Figure 9. A ZigBee packet structure that applies our scheme

4. Application

In a ZigBee network, the coordinator (sender) plays an important role at any time. Imaging a scenario, a home security system, where the coordinator controls a security center. And one of its job is to manage other nodes (receiver) to open or close the door lock. When the coordinator wants to update all passing password, it can apply our mechanism to broadcast the message. And during the updating period, all the nodes which control door locks don't let anyone go in until the update completes.

5. Conclusion

In this paper, we adapt the concepts from HORS and Merkle hash tree. We adjust them to fulfill the secure requirements of a ZigBee network. In Zigbee, the sizes of payloads are limited to 80 bytes, so we need to separate our signature into several parts in order to maintain its security level. In our mechanism, we send some of the authentication paths each time, so we need more than one time to verify a signature. As a consequence, we need additional queue to store related authentication paths and related data on the sender and the receivers. On the other hand, while we lose the advantages of one-time signature, we can still maintain the security level to an acceptable degree.

We also use a re-keying scheme in our approach. After we use *r* signatures, we change a new key. The sender should send all the authentication paths left and broadcast a new key to every node to get a refresh the network. If other nodes have enough storage, they can support complex scheme to distinguish which packet using which key. The original ZigBee spec already has a basic set of security design. It supports an encryption / decryption algorithm based on AES computing. Only nodes joining the network, they get a key to secure all outgoing and incoming packets. Although it can defend external attack, it can't protect malicious attack inside. However, our approach can enhance the original security mechanism and resist malicious broadcast messages from an internal invader or a compromised node.

There are several choices of parameters in our mechanism. All of them are listed in table 2. These parameters can influence security level and signature length. We try to find a suitable setting for ZigBee and finally we choose set # 17. Of course, the settings can be adjusted anytime to suit a real environment. Just remember that changing these settings will result in a different security level and may shorten the length of a data payload in advance.

6. Acknowledgements

This work is a result of the collaboration between *Networks and Multimedia Institute at Institute for Information Industry* and *Distributed System and Network Security Laboratory* at *National Chiao Tung University*. The author would also like to thank the support of the *Advanced Mobile Context Aware Application & Service Technology Development Project* of Institute for Information Industry and sponsored by MOEA, R.O.C.

7. References

[1] ZigBee Alliance, "ZigBee Document 053474r07, Version 1.1", September 5, 2005

[2] Edgar H. Callaway, Jr., "Wireless Sensor Networks: Architectures and Protocols", AUERBACH PUBLICATIONS, 2003

[3] ZigBee Alliance, "ZigBee Document 053474r07, Version 1.1", September 5, 2005, Annex A

[4] Leonid Reyzin, Natan Reyzin, "Better than BiBa: Short One-time Signatures with Fast Signing and Verifying", April 30, 2002

[5] Stefaan SeyS, "Power Consumption Evaluation of Efficient Digital Signature Schemes for Low Power Devices", IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WIMOB), August 23, 2005

[6] Kemal Bicakci, Gene Tsudik, Brian Tung, "How to construct optimal one-time signatures", Available at www.ElsevierComputerScience.com, April 24, 2003

[7] Adrian Perrig, "The BiBa one-time signature and broadcast authentication protocol". In Eighth ACM Conference on Computer and Communication Security, pages 28-37. ACM, November 5-8, 2001.

[8] Ralph C. Merkle, "A Digital Signature Based on a Conventional Encryption Function", Advances in Cryptology – CRYPTO'87, volume 293 of Lecture Notes in Computer Science, Springer Verlag, 1987. , pp 369-378

[9] Ralph C. Merkle, "A Certified Digital Signature", Advances in Cryptology – CRYPTO'89, volume 435 of Lecture Notes in Computer Science, Springer Verlag, 1990. , pp 218-238

[10] Ralph C. Merkle, " Protocols for public key cryptosystems" , *In Proceedings of the IEEE Symposium on Research in Security and Privacy*, Apr. 1980. , pp. 122 – 134

[11] Shang-Ming Chang, Shiuhpyng Shieh, Warren W. Lin, Chih-Ming Hsieh, "An Efficient Broadcast Authentication Scheme in Wireless Sensor Networks"

[12] CC2420DBK Quick_Start_1_0 (http://www.chipcon.com/)