A Security Protocol for Wireless Sensor Networks

Chang N. Zhang [1], Qian Yu [1], Xun Huang [1], Cungang Yang [2]

[1] Department of Computer Science, University of Regina
Regina, SK, S4S 0A2, Canada
{zhang, yu209, huang24x}@cs.uregina.ca

[2] Department of Electrical Engineering, Ryerson University
Toronto, ON, M5B 2K3, Canada
cungang@ee.ryerson.ca

Abstract. In this paper, we present an security protocol for Wireless Sensor Networks (WSNs). It is based on the forward and backward property of RC4 states and achieves data confidentiality, data authentication, data integrity, and data freshness with low overhead and simple operation. Furthermore, an RC4-based hash function for the generation of Message Authentication Code (MAC) is presented. The proposed protocol is an ideal solution for wireless sensor networks and other resource-constrained devices where the communication nodes have limited power resources and computational capabilities, and can be widely used in the applications of one-to-one communications as well as broadcasting and multicasting.

Keywords: RC4, Security Protocol, Hash Function, Backward Property of RC4 States, Wireless Sensor Networks, Resource-constrained Communications

1 Introduction

Wireless sensor networks (WSNs), radio frequency identification devices (RFIDs), and some of other resource-constrained applications are being deployed today and will soon become an important part of our infrastructure. Security is a critical factor of these applications due to their impact on privacy, trust and control [1].

Size and cost constraints on resource-constrained devices result in corresponding constraints on resources such as processor capacity, memory, power, and bandwidth. For example, current wireless sensor devices use simple, battery powered 4-bit or 8-bit processors and small amount of memory to perform a few bit-wise and simple arithmetic operations. However, traditional cryptographic algorithms are too complex and power hungry. In order to provide secure communications to resource-constrained devices, lightweight should be the first concern.

A. Perrig et al presented a set of security protocols named SPINS [2] for wireless sensor networks. SPINS has two security building blocks: SNEP provides data

Please use the following format when citing this chapter:

Zhang, C.N., et al., 2008, in IFIP International Federation for Information Processing, Volume 264; Wireless Sensor and Actor Networks II; Ali Miri; (Boston: Springer), pp. 113–124.

confidentiality, two-party data authentication, as well as data freshness, and μTESLA provides broadcast authentication. SPINS is a typical security protocol for wireless sensor networks, especially it avoids asymmetric cryptography to achieve broadcast authentication. SPINS provides data confidentiality by using RC5 block cipher. According to the analysis, stream ciphers are almost always faster and use far less code than do block ciphers, and the operations provided by most ultra-low devices are limited to bit-wise logic operations and simple arithmetic operations [3, 4]. Hence block cipher is too heavy and stream cipher should be considered.

State Based Key Hop (SBKH) protocol [5] is an RC4-based one-to-one security protocol in which two communication nodes share the common knowledge of the RC4 states. SBKH introduces offset to avoid RC4 weak key issue and continually updates and reuses RC4 state for the whole communication process. But SBKH suffers from fake acknowledgement attacks and does not support multicasting and broadcasting due to the strong resynchronization requirement.

In this paper, we present an RC4-based security protocol for communications on wireless sensor networks. Firstly, we indicate that the RC4 state has the forward and backward property. Secondly, we present a data transmission scheme based on RC4 and its backward property, and claim that it greatly reduces the computation overhead and eliminates the strong resynchronization requirement. Thirdly, an RC4-based hash function is presented to generate MAC for data authentication and data integrity. Our simulation and analysis shows that the proposed protocol does not reduce the security strength provided by original RC4 and it works lightly with less implementation complexity and cost. By using offset, the security performance is improved.

2 RC4 and its Backward Property

Stream ciphers and block ciphers are two classes of encryption algorithms. Stream ciphers encrypt individual byte or bit of plaintext one at a time, using a simple time-dependent encryption transformation. Block ciphers simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation [3]. Stream ciphers and block ciphers have their respective characteristics, but the stream ciphers are almost always faster and use far less code than do block ciphers [3, 4]. RC4 is probably the most widely used stream cipher nowadays due to its simplicity and high efficiency. RC4 is a variable key-size stream cipher based on a 256-byte secret internal state and two one-byte indexes. The data are encrypted by XORing data with the key stream which is generated by RC4 from a base key. RC4 consists of two parts which are key-scheduling algorithm (KSA) and pseudo-random generation algorithm (PRGA). For a given base key, KSA generates an initial permutation state denoted by S_0. PRGA is a repeated loop procedure and each loop generates a one-byte pseudo-random output as the stream key. That is: at each loop, a one-byte stream key is generated and it is XORed with one-byte of the plaintext, in the meantime a new 256-byte permutation state S as well as two one-byte indices i and j are

updated, which defined by $(S_{k+1}, i_{k+1}, j_{k+1}) = PRGA(S_k, i_k, j_k)$ where $i_{k+1}$and j_{k+1} are the indices and S_{k+1} is the state updated from i_k, j_k, and S_k by applying one loop of PRGA.

On the issue of the security strength of the RC4, a number of papers have been published to analyze the possible methods of attacking RC4, but none is practical with a reasonable key length, such as 128 bits [4]. WEP is an algorithm to secure IEEE 802.11 wireless network and it requires the use of RC4. Beginning in 2001, several serious weakness (including [6, 7]) were reported and they demonstrate that WEP protocol is vulnerable in a number of areas. In essence, the problem is not with RC4 itself but the way in which keys are generated for use as input to RC4.

In this paper, we present an RC4-based security protocol which maintains the simplicity and efficiency of the RC4, and eliminates its limitations (e.g. it allows delayed data packet). It is based on the following property of RC4 states.

Theorem 1: If $(S^*, i^*, j^*) = PRGA^k(S, i, j)$, then it has $(S, i, j) = IPRGA^K(S^*, i^*, j^*)$ where $PRGA^k$ denotes applying PRGA by k loops (same for $IPRGA^k$) and IPRGA is the reverse algorithm of PRGA (without generating a stream key).

Theorem 1 indicates that any former RC4 state can be recovered from later RC4 state by applying IPRGA corresponding loops. The operation codes of PRGA and IPRGA are depicted in Figure 1 and the proof of the Theorem 1 is provided in Appendix. Thereby, we can conclude that any RC4 state can be forward to a new RC4 state by PRGA and backward to a previous RC4 state by IPRGA. For example, it is easy to generate a previous RC4 state from current RC4 state.

$PRGA(S)$
$Generation\ \ loop:$
$i \leftarrow (i+1) \bmod 256$
$j \leftarrow (j + S[i]) \bmod 256$
$S[i] \leftrightarrow S[j]$
$Output\quad z \leftarrow S[(S[i] + S[j]) \bmod 256]$

$IPRGA(S, i, j)$
$Generation\ loop:$
$S[i] \leftrightarrow S[j]$
$j \leftarrow (j - S[i] + 256) \bmod 256$
$i \leftarrow (i-1) \bmod 256$

Figure 1: The operation codes of PRGA and IPRGA

3 RC4-based Security Protocol

In this section, an RC4-based security protocol for wireless sensor networks (WSNs) is proposed.

3.1 Terminologies and Notations

The terminologies and notations used in this section are described below:

RC4 State: An RC4 state is a permutation state with 256 state elements and 2 indices (i and j) denoted by (S, i, j). Each state element and each index is of 8 bits in length, making the overall RC4 state to be of 258 bytes in total.
Corresponding RC4 State (CRS): A certain RC4 state generates a certain stream key by RC4, so the same RC4 state should be used for both in encryption and decryption for a given message. In the proposed protocol, each communication node keeps an RC4 state called corresponding RC4 state (CRS). The CRS is to be updated after the encryption or decryption of each fixed length data packet.
Offset (O): Offset is an integer which indicates the number of loops to applying PRGA. The offset loops of PRGA takes place after obtaining the initial permutation state S_0 from KSA, but before encrypting or decrypting message. It carries out only after the base key change takes place. We also use offset to name the process of applying the offset loops of PRGA. The purpose of using offset is to discard the offset number of bytes from the beginning of the stream to avoid major statistical bias in the distribution of the initial bytes of RC4 streams [5], in order to strengthen RC4.
Sequence Counter (SC): The sequence counter stores a single natural number (initially zero) and increases by one each interval. The sender and each receiver have their own sequence counter (SC) in their memories.

For the sender, the SC number is increased by one for each new fixed-length data packet. For an encrypted fixed-length data packet, the data segment and MAC checksum are encrypted, but the SC number is not.

For each receiver, by checking the SC number of a new incoming fixed-length data packet, operations are performed as follows: if the incoming SC number is more than one integer value greater than the stored SC number, then the receiver updates its SC number to match the new value and decrypts the data segment by its CRS directly; otherwise, the receiver needs to compute the correct RC4 state by applying PRGA or IPRGA to its CRS a number of loops, and then to decrypt. The receiver has to update its SC number to match the new value which it received.

Figure 2: The (encrypted) fixed-length data packet

Fixed-length Data Packet: The proposed protocol requires that the length of the data packets which transmit between sender and each receiver is fixed. A fixed-length data packet includes a SC value, a data segment, and a MAC value. Note that if there are not enough data in the data segment of the last fixed-length data packet, the sender fills in a terminating symbol in the last packet, and adds random numbers to pad the empty place at the end of the data segment of the last fixed-length data packet.

3.2 Secure Data Transmission

This section presents the secure data transmission in the proposed protocol. During the handshaking phase (e.g. before the initial transmission, or the base key change), the sender and each receiver share the new RC4 base key, the MAC key, and the offset value O. Section 3.5 depicts MAC generation.

Note that, whereas WEP and WPA 1.0 reinitialize the RC4 state for every packet and generate the cipher stream from the initialized RC4 state, the proposed protocol does not reinitialize RC4 states frequently. Instead, the proposed protocol maintains the same RC4 base key for a duration known to each communicating node. This requires the initialization of the RC4 state (running KSA) to be carried out only when the base key changes. Furthermore, it is not necessary to update the offset value O frequently.

After acquiring the information during the handshaking phase, both sides apply KSA to generate an initial permutation of the RC4 state S_0, and then apply offset (O) loops of PRGA to S_0 to generate a new RC4 state as the CRS for key stream generation. At the same time, both sides set their sequence counters to zero (SC=0).

Figure 3 depicts the three-step operations which are executed by the sender. After these three steps, the encrypted fixed-length data packets are produced from the input plaintext message. The detailed descriptions of each step are presented below:

(1) The sender produces unencrypted fixed-length data packets. The sender divides the input plaintext message into contiguous fixed-length data segments and assigns the Sequence Counters (SC) to each of them. If there are not enough data in the data segment of the last fixed-length data packet, terminating symbol and random number (TR) as padding are added at the end of the data segment of the last fixed-length data packet for data encryption.

(2) The sender calculates the MAC checksum by inputting SC, data segments and the MAC key, and then fills the MAC checksum into the MAC segment. If there are not enough data in the data segment of the last fixed-length data packet, 0 are added at the empty place of the last fixed-length data packet for MAC generation. Section 3.5 depicts the MAC generation.

(3) The sender produces the encrypted fixed-length data packets. For each byte of the data which needs to be encrypted (here, it is the data segment and MAC segment), the sender XORs each byte with the correct stream key which is produced by the PRGA at the correct RC4 state. Following these operations for the bytes of the data which need to be encrypted, the encryption of the fixed-length data packet is completed and the packet is ready for transmission. The sender then increases its SC by one and updates its CRS.

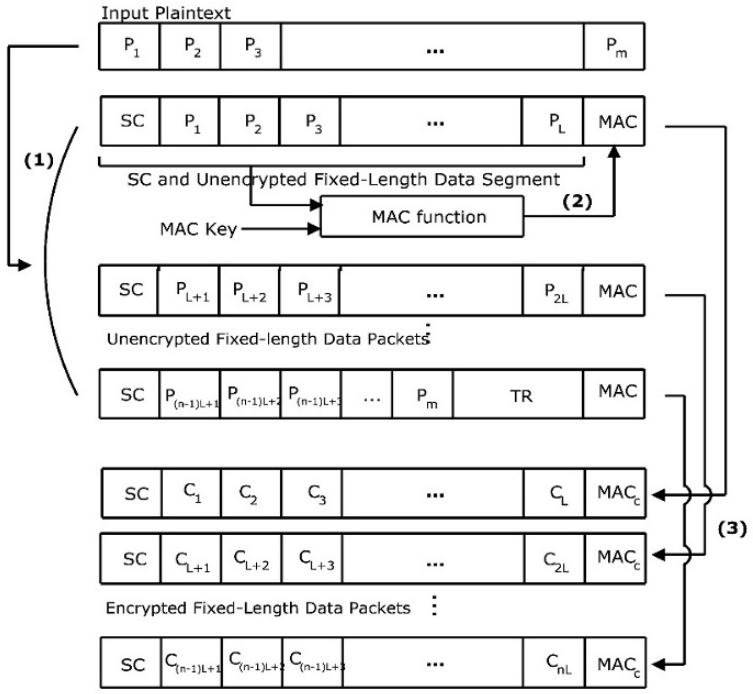

Figure 3: The sender's operations for data transmission in the proposed protocol

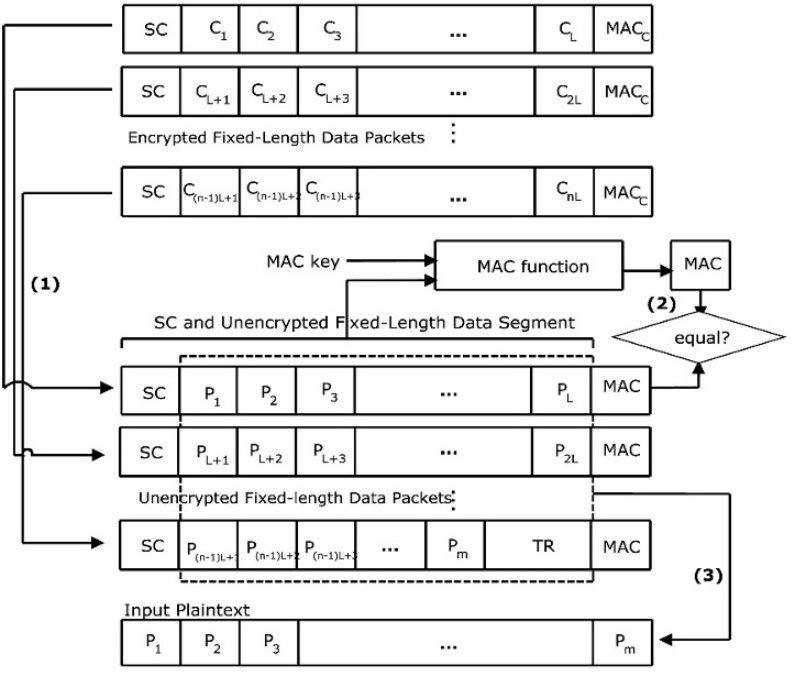

Figure 4: The receiver's operations for data transmission in the proposed protocol

Figure 4 depicts the three-step operations which are executed by the receiver. After the three steps, the input plaintext message is recovered from the corresponding fixed-length data packets. The details of each step are presented below:

(1) The receiver decrypts the encrypted fixed-length data packets. By reading the SC value of the coming fixed-length data packet, the receiver can find the right RC4 state based on the forward and backward property of RC4 states. Each receiver then calculates the correct key stream to decrypt the data which has been encrypted (here, it is the data segment and MAC segment) through XORing the corresponding key stream with the encrypted bytes.

(2) The receiver verifies the MAC checksum. The receiver verifies this fixed-length data packet by re-computing the MAC checksum. If the recomputed MAC checksum is the same as it was received, the MAC checksum is verified. Note that, 0 are added at the empty place of the last fixed-length data packet for MAC generation if there are not enough data in the data segment of the last fixed-length data packet.

(3) The receiver recovers the corresponding input plaintext message. After all the fixed-length data packets, which correspond to an input plaintext message, have been decrypted and their MAC checksum are verified, the receiver can restore them into the corresponding input plaintext message. The receiver then sets its SC as the SC value received, and updates its CRS.

3.3 Delayed or Lost Packets

In many network-based applications, it cannot be guaranteed that packets are received in the right order or there is no lost packet. Therefore, the application of a protocol which is based on the strong synchronization requirement is limited. Due to the backward and forward property of RC4 states, the proposed protocol allows that packets be received in an arbitrary order, or resending a lost packet.

3.4 Data Authentication

In our research, we assume that all communication nodes are trusted parties. Therefore, data authentication can be achieved through a purely symmetric mechanism, in which the sender and receiver share a secret key to compute a message authentication code (MAC) for all communicated data. When a message with a correct MAC arrives, the receiver knows that it was sent by the sender. If the receivers do not have strong trust assumptions, it needs an asymmetric mechanism. The asymmetric mechanism is a whole topic unto itself and is not covered in this paper.

No matter the receiver is trusted party or not, an MAC generation scheme is required for data authentication. MAC (message authentication code) is an authentication tag which is stored in the communication packet to verify the data authentication and data integrity during data transmission. Most popular method to compute MAC is to employ a

secure hash function. Hash function-based MACs (often called HMACs) use a key or keys in conjunction with a hash function to produce a checksum that is appended to the message. There are a number of well known secure hash functions including SHA (SHA-512), MD5, Whirlpool and Ripemd-160. Some of them have been widely used and become key components of security protocols. However none of them is suitable for resource-constraint applications. For example, SHA-512 requires each block (1024 bits) to be processed by 80 rounds. Each round needs a large number of 64-bit applications [4]. In this paper, we present an RC4-based hash function in section 3.5 for MAC generation.

3.5. RC4-based Hash Function

In this section, we present an RC4-based hash function which can be used to generate MAC. Compared with the hash functions which are mentioned in section 3.4, the proposed hash function significantly reduces the computation overhead and achieves the requirements of a secure hash functions. Its security is based on the randomness of RC4 states. In the proposed RC4-based hash function, we use the notations of KSA* and PRGA*, each of which is a part of KSA or PRGA procedures that are listed below:

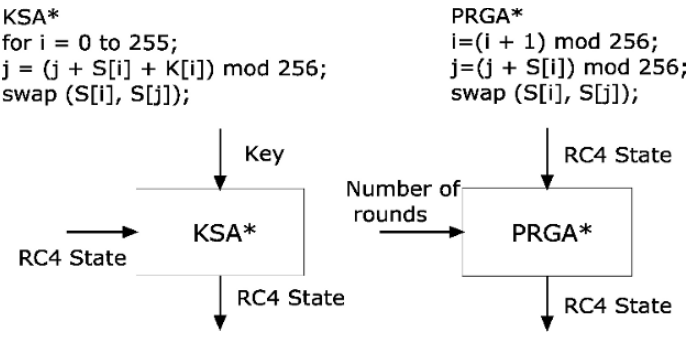

Figure 5: The operation codes of KSA* and PRGA* and their graphic representations

The input of KSA* is an RC4 state (e.g. an RC4 state which is generated by PRGA*) and a 64-byte key (K[0], K[1], ---, K[63]); the output is a new RC4 state. The input of PRGA* is an RC4 state and an integer to indicate how many loops the PRGA* applies; the output is another new RC4 state. Essentially KSA* is the same as KSA but it does not initiate the RC4 state at the beginning, and PRGA* is the same as PRGA but it does not generate the key stream.

The input of the proposed hash function is a message with a maximum length of less than 2^{64} bits and the output is a 258-byte message digest. The process of the proposed hash function can be described by the following two steps:

Step 1: Divide input message into 512-bit blocks.

The input message is divided into 512-bit blocks. The message is padded so that its length is congruent to 448 modulo 512 (length ≡ 448 mod 512). The padding bit is 0 and the number of padding bits is in the range of 0 to 512. Figure 6 depicts padding the input message. The output of this step is N pieces of 512-bit blocks notated by m_1, m_2, ---, m_N.

Note that when the proposed hash function is not used in the proposed protocol and the receiver does not have the length information of the input message, a segment for length information is required.

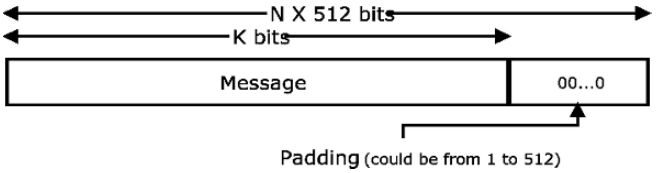

Figure 6: Padding the message

Step 2: Process message blocks.

The first 512-bit message block m_1 is processed by KSA and PRGA* in sequence. The input of the KSA is m_1 as the secure key and the output is an RC4 state named $State_{m1}$. The input of the PRGA* is $State_{m1}$ (as the initial state for PRGA*) and the output is an RC4 state named $State_1$ which is generated by applying (m_1 mod 2^5) loops of PRGA*. The output RC4 state ($State_x$) will be the initial RC4 state of KSA* to process next message block. The second 512-bit message block is processed by KSA* and PRGA* in sequence. The input of KSA* are the $State_2$ as the initial RC4 state and m_2 as the secure key and the output is $State_{m2}$. The same, $State_{m2}$ will be used as the input for the corresponding (m_2 mod 2^5) loops of PRGA* operation to generate $State_2$. The rest of blocks do the same process as the second block with corresponding input. $State_N$ is the 258-byte output of this RC4-based hash function. The process of this step is depicted in Figure 7 below.

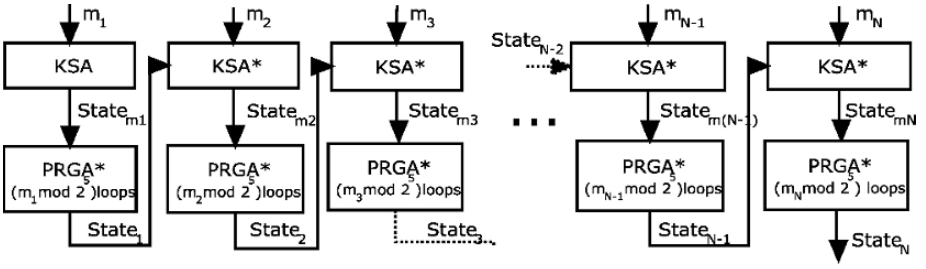

Figure 7: The procedure of the proposed RC4-based hash function

4 Analysis and Implementation

The proposed protocol is based on the RC4 algorithm with improvements and modifications. The improvements and modifications include using offset and the forward and backward property of RC4 states.

The forward and backward property of RC4 states is an intrinsic property of the RC4 algorithm. Any previous RC4 state can be recovered from the current or later RC4 state by simple addition and subtraction operations, as well as by swap operations. Since the proposed protocol simply uses this property and makes no changes to the original RC4, the proposed protocol does not reduce the security strength of RC4 by using the forward and backward property.

Offset is used in the proposed protocol to discard the first few bytes of the key stream, and data encryption or decryption begins at some later position. The purpose of using offset is to avoid the weaknesses of the initial RC4 states [3]. While the inclusion of offset is not a fundamental change to RC4, the security performance is improved. Thus, the proposed protocol does not reduce the security strength of RC4 by using offset.

For proposed RC4-based hash function, we have implemented a simulation to analysis the randomness of the RC4 state generation for KSA* and PRGA*. Simulation results show that the generation of the new RC4 state for KSA* and PRGA* maintains the same randomness as the KSA and PRGA.

Based on the analysis above, we believe that the proposed protocol uses RC4 in a way which makes effective use of RC4's strengths and reduces its weaknesses. The proposed protocol has achieved four main security properties required for designing a general secure communication scheme as below. Furthermore, the proposed protocol achieves semantic security and defends against lots of popular attack effectively, such as replay attacks and modified packet attacks.

- Data Confidentiality: The proposed protocol achieves data confidentiality through encrypting the data through the RC4 stream cipher.
- Data Authentication: The proposed protocol achieves data authentication through the MAC check.
- Data Integrity: The proposed protocol achieves data integrity through data authentication, which is a stronger way to check data integrity.
- Data Freshness: The proposed protocol achieves data freshness through the monotonically increasing value of the sequence counter for each data packet.

In the respect of the performance, the proposed protocol runs simply and efficiently because of the benefits of RC4 and some new features.

Using forward and backward property of RC4 states is one of the new features. By using it, the proposed protocol does not require key initialization frequently. This translates directly to some simplification and likely power saving, as well as improved performance over other original RC4 based protocols. For example, if the length of the data segment of the packet is 256 bytes, since KSA is unnecessary to be applied for every

new packet, the encryption/decryption process of the proposed security protocol requires about half time than the time required by WEP.

Furthermore, all operations in the proposed protocol are simple swap, XOR, and addition and subtraction operations. Due to the simple implementation complexity, the proposed protocol is less expensive in terms of power and CPU resources. Considering the likely hardware support for RC4, it should be easy to selectively generate RC4 states from a base key or a pre-generated RC4 state. Hence, the proposed protocol should be compatible with existing hardware.

The proposed RC4-based security protocol has been implemented. It consists of two major parts. One is on sender side, and another is on receiver side. Our tests show that the proposed protocol works well to deal with the delayed or lost packet. Compared with the ordinary RC4 based security protocol, the proposed one are almost twice faster than the original one. Our tests also show that the proposed protocol is secure and has the ability to against a number of possible attacks including acknowledge attack, replay attack, and modified packet attack. We have applied the proposed protocol to multicasting environment where the numbers are dynamically changed. The experiment works correctly and smoothly. In the next step, we are going to implement the proposed protocol on a wireless sensor network which consists of a base station and a number of wireless sensor nodes. The results will be reported when the experiment is completed.

5 Conclusion

Wireless sensor networks and some of other resource-constrained applications are being deployed today and become more and more important. Security is a critical factor of these applications so a simple, efficient, and robust security protocol for secure data transmission on those devices is in large and urgent demand. This paper focuses on the designs of a lightweight security protocol for communications on wireless sensor networks which is based on the simple structure of the RC4 stream cipher and its forward and backward property.

Due to its simplicity, efficiency and high security, the proposed protocol is comparable with well-known security protocols. It is very simple to implement and efficient in both hardware and software, and is compatible with different levels of security. It can be widely used in many network applications. Furthermore, the proposed RC4-based hash function is a relatively independent part and can be served as one of the key components for other security applications.

References

[1] J. P. Kaps, G. Gaubatz, and B. Sunar. Cryptography on a Speck of Dust. IEEE Computer

Magazine, Feb. 2007, pp38-44.
[2] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. Wireless Networks, 2002, pp521-534.
[3] I. Mantin. Analysis of the Stream Cipher RC4. Master's thesis, The Weizmann Institute of Science, 2001.
[4] W. Stallings. Cryptography and Network Security, Principles and Practice. Prentice Hall, 2003.
[5] S. Mitchell and K. Srinivasan. State Based Key Hop Protocol: A Lightweight Security Protocol for Wireless Networks. In proceedings of the 1st ACM international workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, 2004, pp112-118.
[6] A. Stubblefield, J. loannidis, and A.D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. AT&T Labs Technical Report, Aug. 2001
[7] S. Fluhrer, I. Mantin, and A. Shamir. Weakness in the Key Scheduling Algorithm of RC4. Proceedings, Workshop in Selected Areas of Cryptography, 2001.

Appendix

Theorem 1: For any r, $n \geq r \geq 0$, we have:
$IPRGA^r (S_n, i_n, j_n) = (S_{n-r}, i_{n-r}, j_{n-r})$
Proof. The proof is conducted by induction on integer r
Base Case:
It is true when r=0: $IPRGA^0 (S_n,i_n,j_n) = (S_n,i_n,j_n)$
Inductive Step:
Assume that for any $r \leq k$ we have: $IPRGA^k (S_n, i_n, j_n) = (S_{n-k}, i_{n-k}, j_{n-k})$
Consider case of r = k+1
$IPRGA^{k+1} (S_n, i_n, j_n) = IPRGA (IPRGA^k (S_n, i_n, j_n)) = IPRGA (S_{n-k}, i_{n-k}, j_{n-k})$
Let $IPRGA (S_{n-k}, i_{n-k}, j_{n-k}) = (S^*, i^*, j^*)$
According to PRGA and our notation, we have
$PRGA (S_{n-k-1}, i_{n-k-1}, j_{n-k-1}) = (S_{n-k}, i_{n-k}, j_{n-k})$ where $i_{n-k} = i_{n-k-1}+1$ and $j_{n-k} = j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]$
$S_{n-k}[m] = S_{n-k-1}[m]$ where $m \neq i_{n-k-1}$ and $m \neq j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]$
$S_{n-k}[i_{n-k-1} + 1] = S_{n-k-1}[j_{n-k-1} + S_{n-k-1}[i_{n-k-1} + 1]]$
$S_{n-k}[j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]] = S_{n-k-1}[i_{n-k-1} + 1]$
According to $IPRGA (S_{n-k}, i_{n-k}, j_{n-k}) = (S^*, i^*, j^*)$,
we have $S^*[m] = S_{n-k}[m] = S_{n-k-1}[m]$ where $m \neq i_{n-k-1}$ and $m \neq j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]$
and $S^*[i_{n-k-1}+1] = S_{n-k}[j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]] = S_{n-k-1}[i_{n-k-1} + 1]$
$S^*[j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]] = S_{n-k}[i_{n-k-1}+1] = S_{n-k-1}[j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1]$
Thus $S^* = S_{n-k-1}$, $j^* = j_{n-k} - S^*[i_{n-k}] = j_{n-k-1} + S_{n-k-1}[i_{n-k-1}+1] - S_{n-k-1}[i_{n-k-1}+1] = j_{n-k-1}$, $i^* = i_{n-k}-1 = i_{n-k-1}$
Therefore, we have $(S^*, i^*, j^*) = (S_{n-k-1}, i_{n-k-1}, j_{n-k-1})$
That is, for any $n \geq r \geq 0$, we have: $IPRGA^r (S_n,i_n,j_n) = (S_{n-r},i_{n-r},j_{n-r})$ □