A SUITE OF FAST AND EFFICIENT CRYPTOGRAPHIC MECHANISMS FOR WIRELESS ULTRA-LOW POWER DEVICE NETWORKS

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy
in
Computer Science
University of Regina

By
Qian Yu
Regina, Saskatchewan
May, 2014

Copyright 2014: Q. Yu
Qian Yu, candidate for the degree of Doctor of Philosophy in Computer Science, has presented a thesis titled, *A Suite of Fast and Efficient Cryptographic Mechanisms for Wireless Ultra-Low Power Device Networks*, in an oral examination held on April 28, 2014. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner: *Dr. Dan Wu, University of Windsor*

Supervisor: Dr. Chang-Nian Zhang, Department of Computer Science

Committee Member: Dr. Yang Zhao, Department of Mathematics & Statistics

Committee Member: **Dr. Lisa Fan, Department of Computer Science**

Committee Member: Dr. Boting Yang, Department of Computer Science

Chair of Defense: Dr. Yuchao Zhu, Department of Political Science

*Via video conference*  
**Not present at defense**
Abstract

Mobile computing technology is reaching every corner of our lives. Smart phones, tablets, laptop computers are just a few examples of the most known applications. Recent advances in the ultra-low power technologies enabled the development of even smaller, more mobile, autonomous devices. Wireless Sensor Networks (WSNs), Smart Dust, and Radio Frequency Identification (RFID) are several examples of this trend and have been applied to a large number of areas and will be more and more popular for various applications.

Security is a critical factor to many applications due to the impact on privacy, trust and control, and is also important for many applications powered by the ultra-low power devices. Ultra-low power devices are highly constrained in terms of resources, such as they have insufficient computing and storage capabilities. Therefore, it is a challenge to implement security affordable and efficient, and meet the security requirements.

This dissertation presents a suite of cryptographic mechanisms, including a cryptographic hash function, a construction of one-way hash chains, a dynamic access control, and a secure data transmission protocol to offer affordable, efficient but necessary security protection to ultra-low power devices to meet their network security
requirements. More specifically it is to provide data confidentiality, data integrity,
authentication, and access control in their data transmission, for secure data unicast,
secure data broadcast, and secure data multicast.
Acknowledgements

I would like to take this opportunity to express my sincere gratitude to all those who have given me support in the preparation of this dissertation.

First of all, I am grateful to my supervisor, Dr. Chang-Nian Zhang, for his support and guidance through all stages of my studies and research.

I would also like to thank the members of my supervisory committee, Dr. Yang Zhao, Dr. Lisa Fan and Dr. Boting Yang, for their valuable comments and suggestions.

Special thanks go to Mr. Roy Hjelte who proof-read this dissertation for English content. Also thanks to Dr. Hua Li and his research team who provided sensor devices and the assistance on the sensor implementation.

I would also like to thank the Department of Computer Science and the Faculty of Graduate Studies and Research at the University of Regina to support this research work, to provide me with financial support, and to offer me as sessional instructor or teaching assistant for many computer science courses.

My thanks also go to the China Scholarship Council who awarded me the 2012 Chinese Government Award for Outstanding Self-financed Students Abroad.
Post Defense Acknowledgements

I would like to thank the external examiner Dr. Dan Wu for providing insightful comments and suggestions concerning my dissertation. I would also like to thank the defense chair Dr. Yuchao Zhu for presiding on the defense.
Dedication

I would like to dedicate this dissertation to my beloved parents, Jingmei Liu and Changgen Yu, for their selfless love, unreserved encouragement, and unconditional support and care throughout my life. I would also like to dedicate this dissertation and special thanks to my wife Rongrong Yuan for her support and love, and to my daughter Selina Yu for her trust. Without them, achieving this goal would not have been possible.
# Table of Contents

Abstract ........................................................................................................................................ i

Acknowledgements .................................................................................................................. iii

Post Defense Acknowledgements ........................................................................................ iv

Dedication ................................................................................................................................. v

Table of Contents .................................................................................................................. vi

List of Tables ........................................................................................................................... ix

List of Figures .......................................................................................................................... x

Chapter 1  Introduction ........................................................................................................ 1

1.1 Introduction to Wireless Sensor Networks ................................................................ 2

1.2 Security Concerns ......................................................................................................... 5

1.3 Research Motivation ...................................................................................................... 8

1.4 Summary of Contributions .......................................................................................... 12

1.5 Thesis Outline ............................................................................................................... 17

Chapter 2  Background Knowledge ................................................................................ 19

2.1 Basic Concepts of Cryptography ................................................................................ 19

2.2 Data Confidentiality ..................................................................................................... 21

2.2.1 Secret-key and Public-key Cryptographies ............................................................. 21

2.2.2 Stream Ciphers ....................................................................................................... 25

2.3 Cryptographic Hash Function ...................................................................................... 28

2.4 Network Routing ......................................................................................................... 36
2.5 A Simple Model for Network Security ................................................................. 38
2.6 Summary ............................................................................................................. 40

Chapter 3  RC4 Stream Cipher and the Forward and Backward Property of
RC4 States .................................................................................................................. 41
3.1 Introduction to RC4 ............................................................................................ 41
3.2 The Analysis of RC4 ......................................................................................... 44
3.3 The Forward and Backward Property of RC4 States ........................................ 45
3.4 Summary .......................................................................................................... 47

Chapter 4  RC4-BHF, a Cryptographic Hash Function for Ultra-Low Power
Devices ....................................................................................................................... 48
4.1 RC4-BHF: an RC4-Based Hash Function ........................................................... 48
4.2 Use RC4-BHF to Construct Hash Chains ......................................................... 58
4.3 Other Cryptographic Mechanisms .................................................................... 60
4.4 Summary .......................................................................................................... 63

Chapter 5  SDTP, a Secure Data Transmission Protocol for Secure Data Unicast,
Secure Data Broadcast and Secure Data Multicast in Wireless Ultra-Low Power
Device Networks ....................................................................................................... 65
5.1 Notations and Terminologies ............................................................................ 65
5.2 Secure Data Unicast ......................................................................................... 67
5.3 Secure Data Broadcast and To Handle Delayed or Lost Packets ..................... 75
5.4 Secure Data Multicast and Dynamic Access Control ....................................... 80
5.5 Summary .......................................................................................................... 87

Chapter 6  Analysis .................................................................................................. 88
6.1 The Analysis of RC4-BHF ................................................................................ 88
6.2 The Analysis of SDTP ...................................................................................... 100
6.3 Support for Ultra-Low Power Devices ............................................................. 107
6.4 Summary .......................................................................................................... 112
Chapter 7  Conclusions and Future Work.................................................................113

7.1 Conclusions ............................................................................................................113
7.2 Recommendations for Future Research..................................................................117

Bibliography ..................................................................................................................120

Appendix A  Proof of the Reversibility of RC4 States and a Programming Simulation......................................................................................................................131

Appendix B  Source Code...........................................................................................139
List of Tables

Table 6.1: The Maurer Test Value of the Files Randomly Generated by RC4-BHF.......... 93
Table 6.2: The Maurer Test Value of the Files Generated by ANSI X9.17 PRNG.......... 93
Table 6.3: Speed Comparison with RC4-BHF and Some Well-known Hash Functions .. 95
Table 6.4: Selected Hash Functions and the Processors They are Designed for............ 96
Table 6.5: The Comparisons of the Execution Times on an 8-bit Machine (1).......... 106
Table 6.6: The Comparisons of the Execution Times on an 8-bit Machine (2).......... 107
List of Figures

Figure 2.1: Secret-key Cryptography ................................................................. 22
Figure 2.2: Public-key Cryptography ............................................................... 24
Figure 2.3: Stream Cipher ............................................................................. 27
Figure 2.4: Examples of the Use of a Hash Function for Message Authentication .. 31
Figure 2.5: Message Authentication Code ..................................................... 33
Figure 2.6: Digital Signature .......................................................................... 34
Figure 2.7: General Structure of Secure Hash Value ........................................ 35
Figure 2.8: Examples of Unicast, Broadcast, and Multicast ............................ 37
Figure 2.9: A Simple Model for Network Security .......................................... 39
Figure 4.1: The Appending Process ................................................................. 52
Figure 4.2: The Dividing Process ................................................................. 52
Figure 4.3: The Compression Process ............................................................ 53
Figure 4.4: The Output Function .................................................................. 56
Figure 4.5: The Truncation Process (256 bits Hash Value) ............................. 57
Figure 4.6: The Truncation Process (128 bits Hash Value, Odd Number Case) .. 57
Figure 4.7: The Truncation Process (128 bits Hash Value, Even Number Case) .. 57
Figure 4.8: RC4-BHF based Message Authentication Function ..................... 62
Figure 5.1: Fixed-Length Data Packet ........................................................... 67
Figure 5.2: The Operations Executed by the Sender ....................................... 73
Figure 5.3: The Operations Executed by the Receiver ................................... 74
Figure 5.4: An Example of the Data Broadcast through Multi-Hop Connection .. 76
Figure 5.5: An Example of a Hierarchical Multicast Tree ............................. 81
Figure 6.1: The Setup of the Experiment ................................................................. 109
Figure 6.2: The Setup of the Node A ................................................................. 110
Figure 6.3: The Setup of the Node B ................................................................. 111
Figure A.1: Simulate that the Entries of State Vector S are Set Equal to the Values from 0 through 255 in Ascending Order ......................................................... 134
Figure A.2: Simulate the Process of Applying KSA Algorithm ............................. 135
Figure A.3: Simulate the Process of Generating a Random RC4 State ................. 136
Figure A.4: Simulate the Process of Applying PRGA Algorithm ......................... 137
Figure A.5: Simulate the Process of Applying IPRGA Algorithm ....................... 138
Chapter 1

Introduction

Ubiquitous computing presents the notion that computing devices can be everywhere. Current rapid developments show that, in the near future, the wide availability of relatively low-cost devices along with advances in the ultra-low power technologies will enable ultra-low power devices, such as sensor nodes, smart dust, and radio frequency identification, to reach every corner of our daily lives and become commonly deployed in many fields and areas.

Security is a critical factor for many applications due to the impact on privacy, trust and control [1]. It is also very important for many applications powered by ultra-low power devices. Since ultra-low power devices are highly constrained in terms of resources, how to implement security affordably and efficiently without sacrificing the strength of their security requirements is one of the major concerns and challenges [2].

This dissertation presents a suite of cryptographic mechanisms, including a cryptographic hash function, a construction of one-way hash chains, a dynamic access control, and a secure data transmission protocol to offer affordable, efficient and
necessary security protection to ultra-low power devices to meet their network security requirements. To promote simplicity, the wireless sensor networks have been used as an example to present the research work in this dissertation, but it is applicable to other wireless ultra-low power device networks and can be extended to regular networks and mobile applications.

In this chapter, a typical example of the wireless ultra-low power device networks - wireless sensor networks, and its security concerns are introduced, the motivation of the research is presented, the contribution of this research work is summarized, and the outline of the rest of this dissertation is provided.

1.1 Introduction to Wireless Sensor Networks

The wireless sensor is a typical example of the ultra-low power devices. The wireless sensor network is the wireless network consisting of spatially distributed autonomous devices, sensors, to cooperatively monitor natural or manmade environments such as temperature, sound, light, vibration, pressure, motion, traffic, pollutants and radiation, at a level of granularity [3], which is to serve as an interface to the real world, providing the physical information to a computer system [4]. This is normally achieved by placing hundreds to thousands of sensor nodes in the target area. In order to be mass deployable, the sensor nodes must be inexpensive, expandable, and easy to deploy, use, and maintain [5]. These features make wireless sensor networks very flexible and opened up a wide
range of applications. An overview of the history of the sensor nodes and sensor networks can be found in [6].

Regarding the services offered by wireless sensor networks, they can be classified into three major categories: monitoring, alerting, and provisioning of information on-demand [7]: In the first category, sensor nodes can continuously monitor certain features of their surroundings and timely send such information to the base station. In the second category, sensors can check whether certain physical circumstances are occurring, alerting the users of the system when an alarm is triggered. In the last category, the network can be queried about the actual levels of a certain feature, providing information on-demand.

Most of the applications of wireless sensor networks require the sensor nodes to operate unattended for a long period of time powered by an energy source, usually small batteries. Benefits from recent energy harvesting technologies [8] and inductive charging technologies [9], the power supply is not the main problem to the sensor node as before, but it still can only provide a certain amount of power. The range of radio transmitters of sensors for wireless data transfer is limited which means they can only communicate directly with nodes in close proximity. Most wireless sensor networks establish a routing forest with the base station at its root through multi-hop connections. Sensor nodes are a collection of tiny devices with the task of measuring the physical data of its surroundings, and the base stations are powerful devices in charge of collecting data from the nodes,
forwarding control information from the users, and communicating with the outside world. The base stations are assumed to have sufficient resources for computing, process, storages, and communication [10]. Ad-hoc sensor networks (ASNs) are a special form of wireless sensor networks in that the nodes are highly mobile. In some cases the membership in the network can change, for example new devices can be added, old devices can be removed, and some might be moved to a different location in the network [11].

Each sensor node is typically equipped with the following parts [5]: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, typically including small amount of memory and an 8-bit low-end microprocessor, an electronic circuit for interfacing with the sensors, and an energy source, usually a battery or an embedded form of energy harvesting. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust. The cost of sensor nodes is similarly variable, depending on the complexity of the individual sensor node. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as power supply, computational and processing capability, data storage, communications range and bandwidth [12]. Routing is the most common propagation technique in wireless sensor networks [13] and the most common routing types are unicast, broadcast and multicast. The related knowledge of routing and the most common routing types is introduced in Section 2.4.
1.2 Security Concerns

Originally, wireless sensor networks were developed to study natural environments and monitor animal habitats. The data gathered by the sensor nodes for that purpose were public and there was no value for an attacker in tampering with the operation of the network, therefore there was no need for any form of security. However, with the increasing number of applications for wireless sensor networks, the security concerns are becoming the bottle neck for widespread deployment [5]. Good introductions to the security issues of wireless sensor networks can be found in [14, 15, 16]. The following three examples are given which may give some ideas why security is required to many applications of the wireless sensor networks.

**Example 1:** Traditional meters are metering devices used on utility mains to measure utility consumption, such as the consumption of electricity, natural gas, or water. The smart meters are new applications today, which are a special form of sensor nodes. In addition to the traditional meters, smart meters enable two-way real-time communication between the meter and the central system. For example, the smart meter can monitor the utility consumption in real time, provides identification and track of utility outages, and responses to abnormal situation (e.g., water or gas leaking, power short circuit). Smart meters have been installed in many districts nowadays, and will be used more widely in the near future. For example, the City of Regina has applied a simple version of smart water meters for many years, and SaskPower and SaskEnergy are planning to install
around 800,000 power and natural gas smart meters in Saskatchewan in the next five years. These organizations are also doing research on how to apply smart grid, a form of wireless sensor networks, to the utility infrastructure, which allow the utility companies to monitor and to manage the infrastructure in real time. Any leakage or stoppage in gas, power, or water will be picked up by the sensors and reported to a monitoring facility. This information is used to generate an up to the minute status of the utility infrastructure. This type of monitoring can be achieved by mixing thousands of maintenance free, grain of rice sized, and inexpensive sensor nodes into the utility infrastructure. Once in place, the sensor nodes will form a network and relay their measurement data to base stations. Both the smart meters and smart grids have security concerns. Without appropriate security protection in place, it would be easy for a third party to obtain the measurement data, and it would also be easy for an attacker to send fake information to fool the utility companies to shut down the service for a single family, a building, or even shut down the main line if the measurement data indicate a major leak or some other problem.

Example 2: Wireless sensor networks can be used for military purposes, for example to detect and gain information about enemy movements, explosions, and other phenomena of interest. The information is relayed to mobile command posts where it helps the commanders to make decisions about troop movements, calls for support, etc. The sensor nodes can either be dropped off by an aircraft over enemy territory after which they lie stationary on the ground, or they are carried by each soldier and vehicle
and therefore form mobile wireless sensor networks. In this scenario it is of utmost importance to prevent the enemy from listening to the transmitted data.

**Example 3:** Recently there is a trend to deploy wireless sensor networks to health services, such as hospitals and care centers. In these applications, sensors are being used extensively to monitor the condition of the patients and to manage the medical devices. The information collected by sensor nodes is relayed to base stations to make decision, and sensor nodes take appropriate action once receiving the instruction from the base stations. Firstly, the data collect from patients are private and should be protected. Secondly, the instruction sent from base station to medical devices is critical and should also be protected, identified, and authenticated. Many security experts are warning that these wireless sensor networks could be the next target for attackers. For example, a hacker could send a fake instruction to an insulin pump that could change the delivery of insulin to the patient, or even force the device to dispense fatal insulin doses; another hacker could send a fake instruction to force a pacemaker to deliver a high voltage shock directly to a person’s heart to kill a pacemaker-equipped person. [17] identified that “these attacks are now becoming a major concern… In a word in which communication networks and medical devices can dictate life or death, these systems, if compromised, pose a significant threat to the public and private sector.”

From these examples, we confirm that a certain level of security is needed for many applications of the wireless sensor networks. However, achieving this goal is not easy.
Wireless sensor networks are especially vulnerable against attacks and usually they are not tamper-resistant due to the cost constraints [2]. Additionally, any internal or external device can access the data exchange because the communication channel is public. Cost and size constraints on sensor nodes result in insufficient computational and processing capabilities, limited memory and low bandwidth [12], but in order to implement security, certain resource and process are required. It is impractical to apply conventional security mechanisms which are designed for powerful digital systems to wireless sensor networks. Therefore, there is a strong demand for the development and implementation of an affordable and efficient security solution for wireless sensor networks to meet their security goals, such as confidentiality, integrity, authentication, and access control.

1.3 Research Motivation

A cryptographic hash function is indispensable to achieve data integrity and authentication and can also be used to construct many cryptographic mechanisms and in a wide variety of security applications. The cryptographic hash function is introduced in Section 2.3.

So far, the best known and widely deployed cryptographic hash functions are the MD4-family hash functions, such as MD4, MD5, SHA-0, SHA-1, SHA-256, and RIPEMD [18]. Their designs are based more or less on the MD4 algorithm which was invented by Ron Rivest in 1990. Recent analyses have demonstrated that most members
of the MD4-family are vulnerable and there are attacks that allow finding random collisions faster than expected. For example, Biham et al. presented collisions for SHA-0 [19] and Wang et al. reported collisions for MD4, MD5, SHA-0, SHA-1, HAVAL-128 and RIPEMD [20, 21, 22]. The weaknesses may compromise the security of the applications in which these hash functions are used. Because these widely-deployed hash functions were broken, it will be beneficial to design new hash functions with different internal structures.

On the other hand, the emerging ultra-low power technologies set new challenges for cryptographic algorithms and protocols because their computing, storage and power resources are limited. The traditional cryptographic algorithms are not suited to this environment. Most hash functions were designed specifically for implementation on 32-bit or 64-bit machines, but an ultra-low power device, such as a sensor node is typically equipped with an 8-bit low-end microprocessor with a simple bit-wise operated instruction set. It is very inefficient or even not possible to apply traditional hash functions on these devices. How to achieve data integrity and authentication in wireless sensor networks is a challenge and it could prohibit wide deployment of wireless sensor networks for the applications which requires data integrity and/or authentication.

There has been a constant flow of new design ideas and new analysis techniques to design new hash functions. One of the ideas is to use a stream cipher to construct a hash function. RC4 is a very simple and elegant stream cipher that can be implemented using
relatively modest computing resources. More importantly, RC4 has been tested for many years and resisted different attempts to break it. The efficiency, simplicity and strength of RC4 make it a good cryptographic primitive to construct a lightweight hash function.

Chang et al proposed a hash function called RC4-Hash [23]. The compression function of RC4-Hash applies the key scheduling algorithm that is one of the main components of RC4. It claimed that the generic attacks which are effective against MD4-family hash functions fail to work. However, RC4-Hash has table lookup and complex reordering which are very costly. More importantly, Indesteege et al [24] have found that the RC4-Hash is not collision resistant. In 2010, we have proposed a new hash function based on RC4 [25, 26]. Unfortunately, collisions also have been found for this hash function. Although vulnerabilities were found on both these hash functions, due to the significant benefits of RC4, there are still ongoing researches and many researchers in crypto community believe that RC4 is a good cryptographic tool to build a secure and fast hash function.

SPINS [27] is a secure data transmission scheme designed for wireless sensor networks which contains two parts, SNEP provides data confidentiality and two-party authentication through RC5 block cipher, and \( \mu \)TESLA achieves broadcast authentication through a delayed disclosure of symmetric keys. TinySec [28] is a generic security package that can be integrated into wireless sensor networks applications. TinySec achieves data confidentiality through the block cipher and achieves data integrity and
authentication through a message authentication code. The default block cipher for TinySec is Skipjack and the default message authentication code is CBC-MAC, which is also based on the block cipher.

Both SPINS and TinySec use block ciphers. As far as we know, there is not any block cipher that their operations are byte-orientated which is suitable for 8-bit machine. In addition, stream ciphers are almost always faster and use far less code than block ciphers [29, 30] and most of the block ciphers require storing certain substitution-boxes in the memory which is an additional memory cost. For these reason, we believe to use stream ciphers in wireless sensor networks could be a valuable alternative to block ciphers which lead to a significant performance gain in terms of reducing the computing requirement, storage overhead, processing time and power consumption.

The State Based Key Hop protocol (SBKH) [31] uses RC4 stream cipher to design a data transmission protocol for wireless sensor networks. However, SBKH requires strong synchronization and maintains two to four RC4 states for each sensor node. Because the strong synchronization, SBKH cannot be applied to secure data broadcast and multicast, which are two basic types of network routing for wireless sensor networks. Moreover, the resynchronization process is relatively costly which requires more resource usage. In addition, it is also costly to maintain several RC4 states for each sensor node.

My previous work [32] proposed a secure data unicast scheme based on RC4. However this scheme only achieves data cryptography based on RC4 algorithm and still
uses a traditional message authentication function to achieve data integrity and authentication. Moreover, the access control which is another requirement for network security has not been covered. Two-way transmission is a form of transmission in which both parties involved transmit information and is very useful or required for many applications, but it has not been discussed. It mentioned that the secure data unicast can be extended to support broadcast and multicast, but the detail has not been provided.

1.4 Summary of Contributions

The objective of this research work is to develop an affordable and efficient security protection for wireless sensor and other ultra-low power device networks to meet their network security requirements, which is to provide data confidentiality, data integrity, authentication, and access control to achieve secure data unicast, secure data broadcast, and secure data multicast. The primary contributions of this research work are highlighted in the following.

(1) RC4-BHF: A New and Fast Hash Function for Ultra-Low Power Devices

This dissertation proposes a new and fast cryptographic hash function, which is called RC4-BHF, for wireless sensor and other ultra-low power device networks.

RC4-BHF is a new attempt to use the RC4 stream cipher to design a hash function. Since vulnerabilities have been found in a number of well-studied hash functions, it is beneficial to propose a hash function with different internal structures. RC4-BHF is such
a new hash function with different internal structures. Moreover, RC4-BHF can be applied to wireless sensor and other ultra-low power device networks, which are normally equipped with an 8-bit low-end microprocessor and most other hash functions cannot be implemented efficiently or are not applicable. RC4-BHF can run much faster compared to the existing hash functions and is exceptionally fast on 8-bit processors.

The initial version and the revised version of RC4-BHF were presented in [26] and [33] representatively, and a complete presentation of RC4-BHF with detailed analysis has been provided in [34] during the course of this research. An application of RC4-BHF which was implemented in a real sensor environment was presented in [35].

(2) **Use RC4-BHF to Construct Security Mechanisms for Ultra-Low Power Devices**

This dissertation introduces how to use RC4-BHF to construct one-way hash chains, such as a one-way RC4 base key chain and a one-way offset number chain, which can be used to generate, distribute, and self-authenticate the base keys and the offset numbers in the proposed secure data transmission protocol to achieve secure data unicast, secure data broadcast, and secure data multicast. As one-way hash chains are very efficient to generate and to verify, it is an ideal approach to be applied on ultra-low power devices.

How to use RC4-BHF to construct one-way hash chains was presented in [36] during the course of this research.

This dissertation also discusses how to use RC4-BHF to construct other cryptographic mechanisms and security applications. Benefits from the high efficiency
and simplicity of RC4-BHF, we can construct many efficient and simple cryptographic mechanisms or security applications based on RC4-BHF. This dissertation shows a couple examples which use RC4-BHF to construct a message authentication function and a pseudorandom number generator. The pseudorandom number generator can be used to generate pseudorandom numbers in the proposed secure data transmission protocol for secure data unicast, secure data broadcast, and secure data multicast.

(3) Secure Data Unicast for Ultra-Low Power Devices

This dissertation presents an enhanced secure data unicast for wireless sensor and other ultra-low power device networks.

The proposed secure data unicast is very efficient to provide secure data transmission between two parties, and can achieve all necessary network security protection, including data confidentiality, data integrity, authentication, and access control. The operations required by the proposed secure data unicast are simple, the speed is very fast, the security is high, and the overhead is low. By using the offset notated in Section 5.1, the insufficient key schedule issue which is a significant weakness of RC4 algorithm has been avoided. It does not require frequent key renewal which leads to further simplification and time saving. It also offers a two-way secure data unicast without adding memory overhead.

The enhanced secure data unicast has been proposed in [37] and [38] during the course of this research.
(4) Secure Data Broadcast for Ultra-Low Power Devices, Including How to Manage the Delayed or Lost Data Packets

This dissertation presents how to achieve secure data broadcast for wireless sensor and other ultra-low power device networks. The issues of delayed or lost data packets are discussed.

By using the feature of the forward and backward property of RC4 states, we found how to eliminate the requirement of the strong synchronization on the states, which is a limitation of a stream cipher. This finding helps to solve the issues of delayed or lost data packets. Through the combination of the proposed secure data unicast and how to handle the issues of delayed or lost data packets, secure data broadcast can be achieved. The proposed secure data broadcast can provide an efficient way to deliver data from a source to all the other participants in the network, which is very efficient to deliver data and is very useful for many network applications. It also achieves all necessary protection for network security, including data confidentiality, data integrity, authentication, and access control.

The proposed secure data broadcast for wireless sensor and other ultra-low power device networks, including how to handle issues of delayed or lost data packets, has been presented in [39].
Secure Data Multicast for Ultra-Low Power Devices, Including a Dynamic Access Control to Support Frequent Membership Changes

This dissertation presents how to achieve secure data multicast for wireless sensor and other ultra-low power device networks. In order to achieve the proposed secure data multicast, a dynamic access control to support frequent key changes is proposed.

The proposed access control supports the frequent key changes in a dynamic membership environment, such as frequent member joining or leaving. For example, each time when a new member joins, an existing member leaves, or in some cases a member moves from one location to another in the network, all participants may need to obtain a new session key. In addition, when a member joins, for most applications it does not allow the new member to be able to retrieve the previous session keys from the new session key; also when a member leaves, most applications do not allow the member who left to retrieve the new session key or future session keys from the previous session keys. The proposed dynamic access control can handle all the above situations. Moreover, the session keys are self-authenticated, this means it is very easy and efficient to authenticate subsequent keys by an authenticated key, which is normally the current session key or a previous session key.

Through the combination of the proposed dynamic access control and the proposed secure data broadcast, secure data multicast can be achieved. The proposed secure data multicast can provide an efficient and secure way to deliver data from a source to certain
target participants in the network, this is very efficient to deliver data and is also very useful for many network applications. It can also achieve the necessary protection for network security, including data confidentiality, data integrity, authentication, and access control.

The proposed secure data multicast for wireless sensor and other ultra-low power device networks, including a dynamic access control to support frequent membership changes, has been presented in [40].

It is worth noting that the proposed cryptographic hash functions and its applications, and the proposed secure data unicast, secure data broadcast and secure data multicast are all based on the RC4 algorithm. This directly simplifies the implementation, saves storage space, and improves efficiency, no matter on hardware or software. Benefits from the speed and simplicity of the RC4 algorithm and its byte-orientated operations, the proposed research work is very simple, fast, and low overhead, and is an ideal network security solution for wireless sensor and other ultra-low power device networks.

1.5 Thesis Outline

The rest of this dissertation is organized as follows. The basic concepts of cryptography, data confidentiality, cryptographic hash function and network routing, as well as a simple network security model are introduced in Chapter 2. The RC4 stream cipher is introduced and analyzed, and the forward and backward property of RC4 states
is reviewed in Chapter 3. Chapter 4 presents the proposed cryptographic hash function RC4-BHF, and discusses how to use RC4-BHF to construct one-way hash chains and other cryptographic mechanisms and security applications. Chapter 5 illustrates the proposed secure data transmission protocol, called SDTP, which provides secure data unicast, secure data broadcast, and secure data multicast. SDTP includes how to manage the delayed or lost data packets and how to achieve the frequent key changes in a dynamic membership environment. The analysis for this research work is provided in Chapter 6, and Chapter 7 concludes this dissertation and recommends future research work.
Chapter 2

Background Knowledge

This chapter\(^1\) introduces the necessary background knowledge. The basic concepts of cryptography are reviewed, the basic knowledge of data confidentiality and cryptographic hash function are studied, the basic network routing is introduced, and a simple network security model is presented in this chapter.

2.1 Basic Concepts of Cryptography

The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. Cryptography is related to issues such as privacy, authenticity, and trust, and is the study of how to obscure what you write in order to make it unintelligible to those who should not read it. Today, cryptography entails more than encryption and decryption, but privacy is still one of the central concerns.

\(^1\) All definitions in this chapter are cited from the following references: [30, 32, 41, 42, 43, 44, 45, 46]. Detailed citations are not given for each definition.
Necessary security protection is required to ensure adequate security of the systems and the data transfer. [47] summarized the following five categories\(^1\) that today’s network security should consider.

- **Access Control:** *Access control* is the ability to protect against unauthorized access. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be limited to authorized individuals.

- **Data Confidentiality:** *Data confidentiality* is the protection to keep the contents of information hidden from all but authorized ones. It is also called privacy or secrecy. To achieve this, data encryption and decryption are normally used.

- **Data Integrity:** *Data integrity* protects from unauthorized data alteration. It assures the data received is exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).

- **Authentication:** *Authentication* is concerned with assuring that a communication is authentic. In other words, authenticity provides assurance of the identity of the communicating parties.

- **Nonrepudiation:** *Nonrepudiation* protects against either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

\(^1\) There is no universal agreement about many terms used in security literature. For example, the term authentication is sometimes used to refer to both the verification of identity and the data integrity.
This is not an essential requirement for the wireless ultra-low power device networks, and therefore, nonrepudiation is not covered in this dissertation.

2.2 Data Confidentiality

Data confidentiality involves two transformations of information: *encryption* and *decryption*. Encryption is the transformation of data into a form that is nearly impossible to read without the appropriate information, such as a key. The purpose of encryption is to ensure privacy by keeping information hidden from anyone whom it is not intended for. Decryption is the reverse process of encryption. It is the transformation of encrypted data, called *ciphertext*, back into its original form, called *plaintext*. Encryption and decryption generally require the use of some secret information, referred to as a *key*. Plaintext can be a text file, a digital image file, or a stream of digitized audio/video, etc.

2.2.1 Secret-key and Public-key Cryptographies

A cryptographic algorithm, also called a cipher, is a mathematical function used for encryption and decryption. A *cryptosystem* (abbreviation for cryptographic system) is a cryptographic algorithm, plus all possible plaintexts, ciphertexts, and keys. There are two major types of cryptographies: *secret-key* (also referred to as symmetric-key) *cryptography* and *public-key* (also called asymmetric-key) *cryptography*. 
Secret-key Cryptography

In secret-key cryptography, the sender and receiver use the same secret key. The sender uses the secret key to encrypt a message, and the receiver uses the same key to decrypt the message. Figure 2.1 depicts this process.

![Secret-key Cryptography Diagram](image)

A two-party communication that applies the secret-key cryptography can be described as follows:

1. The sender and receiver agree on a secret key.
2. The sender encrypts the plaintext with the secret key, and sends the ciphertext to the receiver.
3. The receiver receives the ciphertext and decrypts using the same key.
Block cipher is one of the two main types of the secret-key cryptography. In block cipher, the plaintext is divided into blocks of fixed length which are then encrypted into blocks of ciphertext of the same length. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. The fixed length of every block is called the block size and the length of the key is called the key size. Examples of the block ciphers include DES (Data Encryption Standard [48]) and AES (Advanced Encryption Standard [49, 50]).

The advantages of secret-key cryptography are presented in the below [44]:

- Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than those of the common secret-key schemes.
- Key sizes for public-key schemes are typically much longer than those required for secret-key schemes.
- Secret-key ciphers can be employed as primitives to construct various cryptographic mechanisms.
- Secret-key ciphers can be composed to produce stronger ciphers.

Public-key Cryptography

The main challenge posed by secret-key cryptography is that the sender and receiver must agree on a secret key. The issues of generation, transmission, and storage of keys are known collectively as key management. The cryptosystems need to deal with key management, which means that all keys in a secret-key cryptosystem must remain secret.
In order to address the key management problem of secret-key cryptography, Whitfield Diffie and Martin Hellman introduced the concept of public-key cryptography in 1976 [51]. In their system, each member generates a pair of keys, one of which is called the *public key*, and the other of which is called the *private key*. The public key is published, while the private key is kept secret. In the system, it is no longer necessary to trust a secure channel to share the secret key. The only requirement is that public keys must be associated with their users in a trusted manner. For instance, the key could be authenticated by locating it in a trusted directory. Figure 2.2 presents the public-key encryption and decryption process. Examples of the public key cryptography include Diffie-Hellman [52], RSA [53], Digital Signature Standard [54], and Elliptic Curve Cryptography [55, 56].

![Figure 2.2: Public-key Cryptography](image)
A two-party communication applying public-key cryptography is described below:

(1) When a sender wishes to send a secret message to the receiver, the sender looks up the receiver’s public key in a trusted directory, uses the key to encrypt the message, and sends the message to the receiver.

(2) The receiver then uses its private key to decrypt the message and read it.

The advantages of public-key cryptography are listed below [44]:

- In a two-party communication, there is only one private key, and this key is generated and used by the same party. No key distribution process is needed.
- In order to achieve pairwise privacy in a large network, a secret-key cipher would require a quadratic number of keys (one key per pair), while a public-key cipher would require a linear number of keys (one key per user).
- Depending on the mode of operation, the keys of a public-key scheme may remain unchanged for a long time.

2.2.2 Stream Ciphers

The only encryption scheme that is theoretically secure is the Vernam cipher, or what is called the One Time Pad scheme (OTP). Using this scheme requires a key that is as long as the message, and the ciphertext is produced by XORing the plaintext with the key. An obvious drawback of the OTP is that the large key length increases the difficulty of key distribution and storage. This drawback provides motivation to design stream ciphers
in which the key stream is pseudorandomly generated from a smaller secret key, so that the key stream appears random to an adversary.

Stream cipher is another main type of the two of the secret-key cryptography. They encrypt individual bits or bytes of a plaintext message one at a time, using a simple time-dependent encryption transformation. Stream ciphers are also called state ciphers as the encryption depends not only on the key and plaintext, but also on the current state. Examples of the stream ciphers include RC4, A5/1, A5/2, and FISH. RC4 is introduced in Chapter 3.

If properly designed and used, a stream cipher can be as secure as a block cipher with comparable key length [30]. The primary advantage of a stream cipher is that stream ciphers are almost always faster and use far less code than block ciphers, and have less complex hardware circuitry [29, 30]. Stream ciphers are more appropriate, and in some cases mandatory (e.g., telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be applied in networks with transmission error [57]. For applications that deal with blocks of data, either a block cipher or a stream cipher can be used in virtually any application. Because of their significant advantages, stream ciphers are widely used today, and we can expect many more proposals in the near future.
Figure 2.3: Stream Cipher

Figure 2.3 shows how a stream cipher works. In this structure, a secret key, called base key is input into a stream cipher that produces a stream of 8-bit numbers that are apparently random. A pseudorandom stream is unpredictable without knowledge of the input key. The output of the stream cipher, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-or (XOR) operation. For example, if a byte generated by the generator is 00111010 and a plaintext byte is 11101100, then the resulting ciphertext byte is:

\[
\begin{align*}
00111010 \ (plaintext) \\
\oplus 11101100 \ (key \ stream) \\
\hline
11010110 \ (ciphertext)
\end{align*}
\]

Decryption requires the use of the same pseudorandom sequence:

\[
\begin{align*}
11010110 \ (ciphertext) \\
\oplus 11101100 \ (key \ stream) \\
\hline
00111010 \ (plaintext)
\end{align*}
\]
For any of the stream ciphers, it is vulnerable when using the same pseudorandom sequence to encrypt more than one message. For the keystream $k$, plaintexts $m_1$ and $m_2$, and the corresponding ciphertexts $c_1$ and $c_2$, they have the following relationship:

\[
    c_1 \oplus c_2 = (k \oplus m_1) \oplus (k \oplus m_2) = (k \oplus k) \oplus (m_1 \oplus m_2) = m_1 \oplus m_2
\]

The relationship contains significant information about the plaintexts. Using a single stream or changing keys after every session are two main approaches to overcome this problem.

The following lists important design considerations for a stream cipher [58]:

- The encryption sequence should have a large period without repetition.
- The keystream should approximate the properties of a true random number stream as closely as possible.
- The output of the pseudorandom number generator is conditioned on the value of the input key. To guard against the brute-force attack, the key needs to be sufficiently long.

### 2.3 Cryptographic Hash Function

A hash function $H$ is a transformation that accepts a variable-length block of data $M$ as input and produces a fixed-size hash value $h = H(M)$. The kind of hash function needed for security applications is referred to as a cryptographic hash function. A
A cryptographic hash function is an algorithm for which it is computationally infeasible to find either a data object that maps to a pre-specified hash result (the one-way property) or two data objects that map to the same hash result (the collision-free property). Because of these characteristics, hash functions are often used to determine whether or not data has been changed. In general terms, the principal application of a cryptographic hash function is to verify the integrity of a message, which refers to data integrity or message authentication. When a cryptographic hash function is used in message authentication, the output of the hash function is often referred to as message digest, hash value or hash code. A change to any bit or bits in $M$ results, with high probability, in a change to the hash value. Examples of cryptographic hash functions are MD5 (Message Digest, [59]) and SHA-1 (Secure Hash Algorithm, [60]).

The following summarizes the requirements for a cryptographic hash function [30]:

1. Variable input size: $H$ can be applied to a block of data of any size.

2. Fixed output size: $H$ produces a fixed-length output.

3. Efficiency: $H(x)$ is relatively easy to compute for any given $x$, making both hardware and software implementations practical.

4. Pseudorandomness: the output of $H$ meets standard tests for pseudorandomness. A good hash function has the property that the results of applying the function to a large set of inputs to produce outputs that are evenly distributed and apparently random.
(5) *Preimage resistant* (one-way property): for any given hash value \( h \), it is computationally infeasible to find \( x \) such that \( H(x) = h \).

(6) *Second preimage resistant* (weak collision resistant): for any given block \( x \), it is computationally infeasible to find \( y \neq x \) with \( H(y) = H(x) \).

(7) *Collision resistant* (strong collision resistant): it is computationally infeasible to find any pair \( (x, y) \) such that \( H(x) = H(y) \).

Figure 2.4 [30] illustrates a variety of ways in which a hash value can be used to provide message authentication. When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, because less computation is required.

a) The message plus concatenated hash value is encrypted using secret-key cryptography. The hash value provides the structure required to achieve message authentication. Because encryption is applied to the entire message plus hash value, confidentiality is also provided.

b) Only the hash value is encrypted, using secret-key cryptography. This reduces the processing burden for those applications that do not require confidentiality.

c) It is possible to use a hash function without encryption for message authentication. The technique assumes that A and B share a common secret value S.

d) Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash value.
Figure 2.4: Examples of the Use of a Hash Function for Message Authentication
Message authentication can also be achieved using a Message Authentication Code (MAC), also known as a Keyed Hash Function. A message authentication function takes a secret key and a data block as inputs and produces an authentication tag, referred to as MAC value or MAC checksum.

In fact, Figure 2.4b is a message authentication code. In practice, the specific message authentication algorithms are designed generally more efficient than the combination of hashing and encrypting. However, when data cryptography and message authentication are both required, Figure 2.4a is commonly used. Suppose a message authentication code is used to authenticate messages from a sender to a receiver. Only the receiver can authenticate the message is truly from the sender, as only the sender and the receiver share the secret key, which is used by the message authentication function to generate the MAC value. The process is illustrated in Figure 2.5 and described below.

(1) The sender and receiver agree on a secret key for authentication.

(2) The sender applies a MAC function to the message with the secret key, and creates a MAC value.

(3) The sender sends the MAC value to the receiver with the message. The message may or may not be encrypted, depending on the applications.

(4) The receiver applies the same MAC function as the sender to the message with the shared secret key to calculate a MAC value. The receiver compares this MAC value with the MAC value attached to the message. If the two are the same, the receiver
has successfully authenticated the MAC value. Otherwise, someone is trying to impersonate the sender, the message itself has been altered after the sender attaches the MAC value, or an error occurred during transmission.

Another important application, which is similar to the message authentication code, is the *digital signature*. The operation of the digital signature is similar to that of the message authentication code, but the hash value of a message is encrypted with a user’s private key. Anyone can authenticate that the message is truly from the sender because they can use the public key of the sender to verify the digital signature attached to the message. In this case, an attacker who wishes to alter the message would need to know the user’s private key.
The process of a digital signature is illustrated in Figure 2.6, and described below.

1. The sender applies a hash function to the message and creates a hash value. To create a digital signature, the sender encrypts the hash value with its private key.

2. The sender sends the receiver the message with the digital signature attached. The message itself may or may not be encrypted, depending upon the applications.

3. The receiver applies the same hash function as the sender to the message to calculate a hash value. The receiver also decrypts the digital signature using the sender’s public key to obtain another hash value. The receiver compares these two hash values. If the two are the same, the receiver has successfully authenticated the signature.
Other major applications of a cryptographic hash function include one-way password file generation, intrusion and virus detection, pseudorandom function (PRF) and pseudorandom number generator (PRNG).

The following is to review the general structure of a typical secure hash function, which is indicated in Figure 2.7 [30]. This structure was proposed by Merkle [61, 62] and is the general structure of most hash functions in use today. The hash function takes an input message and partitions it into fixed-sized blocks of bits each. If necessary, the final block is padded to bits. The final block also includes the value of the total length of the input to the hash function, which makes hacking more difficult.

![Figure 2.7: General Structure of Secure Hash Value](image)

<table>
<thead>
<tr>
<th>IV</th>
<th>Initial value</th>
<th>L</th>
<th>Number of input blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVi</td>
<td>Chaining variable</td>
<td>n</td>
<td>Length of hash value</td>
</tr>
<tr>
<td>Yi</td>
<td>ith input block</td>
<td>b</td>
<td>Length of input block</td>
</tr>
<tr>
<td>f</td>
<td>Compression algorithm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.7: General Structure of Secure Hash Value
The hash algorithm involves repeated use of a *compression function*, $f$, that takes two inputs (a chaining variable, which is the $n$-bit input from the previous step, and a $b$-bit block) and produces an $n$-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. The hash function can be summarized as:

$$
CV_0 = IV = \text{initial n–bit value} \\
CV_i = f(CV_{i-1}, Y_{i-1}) \quad 1 \leq i \leq L \\
H(M) = CV_L
$$

where the input to the hash function is $M$ consisting of the blocks $Y_0$, $Y_1$, $\cdots$, $Y_{L-1}$. This structure can produce a secure hash function to operate on a message of any length.

### 2.4 Network Routing

*Routing* is the process of selecting paths in a network along which to send network traffic. Routing is performed for many kinds of networks, including wireless sensor networks. *Unicast*, *broadcast*, and *multicast* are the most important three routing types in networks, which are explained in the following, and examples for each of them are depicted in Figure 2.8 [63].

- **Unicast** is a type of transmission in which information is sent from only one node as sender to only one node as receiver. In other words, unicast transmission is between one-to-one nodes and is only involving two nodes.
Figure 2.8: Examples of Unicast, Broadcast, and Multicast
• Broadcast is a type of transmission in which information is sent from just one node as sender but are received by all the other nodes in the networks as receivers. Some examples of the broadcast applications are software updates, network queries, and command dissemination.

• Multicast is a type of transmission in which there may be one or more than one node as sender and the information sent is meant to a group of nodes that have expressed interest in receiving the information. Some examples of the multicast applications are network queries or command targeting specific members in the network.

2.5 A Simple Model for Network Security

A simple model for network security between two parties [30] is depicted in Figure 2.9 in very general terms. A message is to be transferred from one party to another across some sort of network channel. The two parties, who are the principals in this transaction, have to cooperate for the exchange to take place. A logical information channel is established by defining a network routing through the network from source to destination and by the cooperative use of communication protocols by the two principals.

Security aspects are necessary to protect the information transmission from an opponent who may present a threat to confidentiality or authenticity. The techniques for providing security have the following two components:

• A security-related transformation to the information which is to be sent. Examples
include the encryption of the message, which makes the message is unreadable by the opponent, and the addition code based on the contents of the message, which can be used to verify the identity of the sender.

- Some secret information shared by the two principals. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and to unscramble it on reception.

Figure 2.9: A Simple Model for Network Security

A trusted third party may be needed to achieve the secure transmission. For example, a third party may be responsible for distributing the secret information to the two principals, or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.
This general model shows that there are four basic tasks in designing a particular network security service:

- Design an algorithm to perform the security-related transformation. The algorithm should be designed in a way that an opponent cannot defeat its purpose.
- Generate the secret information to be used with the algorithm.
- Develop methods for the distribution and sharing of the secret information.
- Specify a protocol to be used by the two principals that makes use of the security algorithm and the secret information to achieve a particular security service.

2.6 Summary

In this chapter, we learned the basic concepts in cryptography, data confidentiality, cryptographic hash function, network routing and a simple network security model.

Section 2.1 has covered the basic objectives and concepts for network security and Section 2.2 has introduced the data confidentiality, including the comparison of the secret-key cryptography and public-key cryptography, block ciphers, and stream ciphers. The cryptographic hash function, a variety of ways in which a hash value can be used to provide message authentication, message authentication code, digital signature, and the general structure of secure hash value have been reviewed in Section 2.3. The basic knowledge of network routing and a simple network security model have been studied in Section 2.4 and Section 2.5 respectively.
Chapter 3

RC4 Stream Cipher and the Forward and
Backward Property of RC4 States

This chapter\(^1\) introduces and analyzes the RC4 stream cipher, and reviews the forward and backward property of RC4 states.

3.1 Introduction to RC4

RC4 is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. A study shows that the period of the cipher is overwhelmingly likely to be greater than \(10^{100}\) \([64]\). Eight to sixteen machine operations are required per output byte, and it can be expected to run very efficiently in both hardware and software \([30]\).

RC4 is probably the most widely used stream cipher nowadays. It is used in SSL/TLS, WEP (Wired Equivalent Privacy), WPA (Wi-Fi Protected Access), and BitTorrent. Moreover, it was integrated into Microsoft Windows, Remote Desktop

---

\(^1\) All definitions in this chapter are cited from the following references: [30, 32, 41, 42, 43, 44]. Detailed citations are not given for each definition.
Protocol, Kerberos, Lotus Notes, Apple AOCE, Oracle Secure SQL, Secure Shell, PDF, Skype, and many other commercial applications. In addition, it was chosen to be part of the Cellular Digital Packet Data specification.

RC4 generates a pseudorandom stream of bits (a keystream). The same as other stream ciphers, the keystream can be used for encryption by combining it with the plaintext using bitwise exclusive-or (XOR) operation, and the decryption is performed the same way. To generate the keystream, the cipher makes use of a secret internal state which consists of two parts: a permutation of all 256 possible bytes (elements $S[0], S[1], \ldots, S[255]$) which is denoted by $S$ and two 8-bit index-pointers which are denoted by $i$ and $j$. At all times, $S$ contains a permutation of all 8-bit binary numbers from 0 through 255. Thus RC4 has a huge internal state of $\log_2(2^{8!} \times (2^8)^2) \approx 1700$ bits [23]. The permutation is initialized with a variable length key, typically between 40 and 256 bytes, using the key-scheduling algorithm (KSA). Once this is completed, the stream of bits is generated using the pseudo-random generation algorithm (PRGA). In $PRGA$, a byte $k$ is generated from $S$ by selecting one of the 256 entries in a systematic fashion. As each value of $k$ is generated, the entries in $S$ are once again permuted.

**Key-Scheduling Algorithm (KSA)**

To begin, the entries of $S$ are set equal to the values from 0 through 255 in ascending order, and a temporary vector $T$ is also created. If the length of the key $K$ is 256 bytes, then $K$ is transferred to $T$ directly. Otherwise, for a key of length keylength
bytes, the first keylength elements of $T$ are copied from $K$ and then $K$ is repeated as many times as necessary to fill out $T$. Next, it uses $T$ to produce the initial permutation of $S$. This involves starting with $S[0]$ and going through to $S[255]$, and, for each $S[i]$, swapping $S[i]$ with another byte in $S$ according to a scheme dictated by $T[i]$.

Algorithm 3.1 is the pseudo code of KSA. Since the only operation on $S$ is a swap and the only effect is a permutation, $S$ contains all the numbers from 0 to 255 all the time.

```
for i from 0 to 255
    S[i] := i
    T[i] := K[i mod keylength]
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + T[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

Algorithm 3.1: The Pseudo Code of KSA

**Pseudo-Random Generation Algorithm (PRGA)**

Stream generation involves starting with $S[0]$ and going through to $S[255]$, and for each $S[i]$, swapping $S[i]$ with another byte in $S$ according to a scheme dictated by the current configuration of $S$. After $S[255]$ is reached, the process continues with starting over again at $S[0]$. For encryption, it is to XOR the value $k$ with the next byte of plaintext. For decryption, it is to XOR the value $k$ with the next byte of ciphertext. Algorithm 3.2 is the pseudo code of PRGA.
\begin{algorithm}
  \textbf{i} := 0 \\
  \textbf{j} := 0 \\
  \textbf{while} GeneratingStreamOutput: \\
  \hspace{1em} \textbf{i} := (\textbf{i} + 1) \mod 256 \\
  \hspace{1em} \textbf{j} := (\textbf{j} + \textbf{S}[\textbf{i}]) \mod 256 \\
  \hspace{1em} \text{swap values of } \textbf{S}[\textbf{i}] \text{ and } \textbf{S}[\textbf{j}] \\
  \hspace{1em} \text{Output } \textbf{k} := \textbf{S}[(\textbf{S}[\textbf{i}] + \textbf{S}[\textbf{j}]) \mod 256] \\
  \textbf{ endwhile}
\end{algorithm}

Algorithm 3.2: The Pseudo Code of \textit{PRGA}

3.2 The Analysis of RC4

One of the key benefits of a stream cipher is that stream ciphers are almost always faster and use far less code than block ciphers, and have less complex hardware circuitry [29, 30]. RC4 can be implemented in just a few lines of code. The main factors in RC4's success over a wide range of applications are its speed and simplicity, which is very easy to implement efficiently in both software and hardware.

For the security strength of RC4, a number of studies (e.g., [29, 44, 58, 65, 66, 67, 68]) have been conducted to propose attack scenarios against RC4, but none of them is practical when the RC4 algorithm has been used correctly and has a reasonable key length. So far the most significant weakness of RC4 comes from the insufficient key schedule: the first bytes of output reveal information about the key. This can be corrected by simply discarding the initial portion of the output stream [69].
A serious vulnerability was reported by [70] which is related to the wrong use of RC4. It demonstrates that the Wired Equivalent Privacy (WEP) standard [71], which was intended to provide confidentiality on 802.11 wireless LAN networks, is vulnerable to a particular attack approach. In essence, the problem is not in RC4 itself but the nonrandom or related keys are used as input to RC4. This particular problem does not appear to be applicable to other applications using RC4 and can be remedied in WEP by changing the way how keys are generated. This problem points out the difficulty in designing a secure system that involves both cryptographic algorithm and protocols that make use of these algorithms.

Noteworthy, RC4 is the only common cipher which is immune to the 2011 BEAST attack on TLS 1.0, which exploits a known weakness in the way cipher block chaining mode is used with all of the other ciphers supported by TLS 1.0, which are all block ciphers [72].

3.3 The Forward and Backward Property of RC4 States

An RC4 state is denoted as \((S, i, j)\), which includes a 256-byte state vector \(S\) and two 8-bit index-pointers \(i\) and \(j\). Each element of the state vector and each index pointer are 8 bits long, which makes an RC4 state 258 bytes in total.

In our previous work we showed an important feature of RC4, that is, any RC4 state is reversible [32, 73]. This means if \((S^*, i^*, j^*) = \text{PRGA}^k(S, i, j)\), it has \((S, i, j) = \text{PRGA}^{-k}(S^*, i^*, j^*)\).
$IPRGA^k(S^*,i^*,j^*)$ where $PRGA^k$ denotes applying $k$ times $PRGA$, $IPRGA^k$ denotes applying $k$ times $IPRGA$, and $IPRGA$ is the reverse algorithm of $PRGA$. Algorithm 3.3 is the pseudo code of $IPRGA$. A stream key is generated in each loop with $PRGA$, but by default there is no stream key generation with $IPRGA$. The key stream is only generated by $PRGA$.

```
while ShiftingBackward:
    swap values of S[i] and S[j]
    j := (j - S[i]) mod 256
    i := (i - 1) mod 256
endwhile
```

Algorithm 3.3: The Pseudo Code of $IPRGA$

This feature shows that any previous RC4 state can be recovered from a later RC4 state through applying certain times $IPRGA$. Therefore, any RC4 state can be shifted forward by $PRGA$ and shifted backward by $IPRGA$ to another RC4 state, which is called the forward and backward property of RC4 states.

The feature that any RC4 state is reversible is proved in Theorem 1, and a programming simulation is also conducted. Both are provided in Appendix A.

This feature of RC4 is very important for real-time decryption in network environment. When using a stream cipher, if the cryptographic algorithm does not support the backward feature, the receiver must decrypt the message in order, but it is not always possible with current network technologies as a data packet may be delivered late
or even gets lost in the network. By using RC4, because of this feature, the receiver can decrypt the packet it received in real time without waiting, and the order of the received packets does not matter. If the packets are received in an arbitrary order, it always can move the RC4 state forward or backward to calculate the right key stream to decrypt.

3.4 Summary

In this chapter, we introduced and analyzed the RC4 stream cipher, and reviewed the forward and backward property of RC4 states. The mathematic proof that any RC4 state is reversible and a programming simulation to confirm the forward and backward property of RC4 states are provided in Appendix A.
Chapter 4

RC4-BHF, a Cryptographic Hash Function for Ultra-Low Power Devices

This chapter proposes a cryptographic hash function, called RC4-BHF, which is designed for 8-bit machines, and to be both fast and secure. RC4-BHF plays a fundamental role in data integrity, authentication, access control, and random number generation for the proposed secure data transmission protocol, which is introduced in Chapter 5. How to use RC4-BHF to construct hash chains, such as the one-way RC4 base key chain, and other cryptographic mechanisms are also discussed.

4.1 RC4-BHF: an RC4-Based Hash Function

In this section, RC4-BHF, an RC4-based hash function is presented. RC4-BHF followed the general structure of a typical secure hash function which was introduced in Section 2.3. The input of RC4-BHF is a message of arbitrary length, and a non-negative integer which is called offset number. RC4-BHF has two options for the input of the arbitrary length, which are the input of maximum length of 65536 bits (option I-1) or the
input of maximum length of $2^{64}$ bits (option I-2). The output of RC4-BHF is a fixed-length hash value which could be 256-bit (option O-1) or 128-bit (option O-2) long. The internal state of RC4-BHF is a 256 bytes permutation state. N is an arbitrary non-negative integer and we supposed the length of the input of RC4-BHF is N-bit. N may be zero, it does not need to be a multiple of eight, and it can be arbitrarily large.

```
Input: M_k and STATE_k
Output: Updated Internal State STATE_{M_k}

j := 0
for i from 0 to 255
    j := (j + S[i] + M_k[i mod 64]) mod 256
    swap values of S[i] and S[j]
endfor
```

Algorithm 4.1: The Pseudo Code of KSA*

```
Input: Integer len and Internal State STATE_{M_k}
Output: Updated Internal State STATE_{k}

for i from 0 to (len – 1)
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
endfor
```

Algorithm 4.2: The Pseudo Code of PRGA*
The following steps are performed by RC4-BHF to compute the hash value of an input message: (1) Append Padding Bits and Length, and Divide the Padded Message, (2) Compression, (3) Output, and (4) Truncation. The output step and truncation step are optional. The pseudo codes of KSA and PRGA are given in Algorithm 3.1 and Algorithm 3.2 respectively in Chapter 3, and the pseudo codes of KSA* and PRGA* are given in Algorithm 4.1 and Algorithm 4.2 respectively in this section.

In KSA*, $M_k$ is used to produce output $STATE_{Mk}$ from $STATE_k$. This involves starting with $S[0]$ and going through to $S[255]$, and, for each $S[i]$, swapping $S[i]$ with another byte in $S$ according to a scheme dictated by $M_k[i]$.

For PRGA*, it produces $STATE_k$ from $STATE_{Mk}$. This involves starting with $S[0]$ and going through to $S[len]$, and, for each $S[i]$, swapping $S[i]$ with another byte in $S$ according to a scheme dictated by the current configuration of $S$.

Essentially KSA* and PRGA* are originally from KSA and PRGA but with the modification. Compared to the original KSA and PRGA, KSA* does not initiate the RC4 state at the beginning, and PRGA* does not generate the key stream.

**Step 1: Append Padding Bits and Length, and Divide the Padded Message**

The input to this step is the input message of arbitrary length, and the output is one or multiple of 512-bit message blocks. The message is appended with padding bits and length so that the length in bits is congruent to 0, modulo 512. That is, the message is extended so that the length is an exact multiple of 512 bits. Padding is always performed,
even if the length of the message is already congruent to 0, modulo 512. There are two
default options to append the padding bits and length, depending on the length of the
input message is always shorter or equal to 65536 bits, or it may be longer than that. The
appending process is illustrated in Figure 4.1.

**Option I-1: No input message is longer than 65536 bits**

In this option, appending padding bits is performed as follows: a single "1" bit is
appended to the input message, and then "0" bits are appended so that the length in bits of
the padded message becomes congruent to 496, modulo 512. In all, at least 1 bit and at
most 512 bits are appended. A 16-bit representation of N (the length of the input message,
before the padding bits were added) which is denoted as L is appended to the result of the
previous padding bits process. At this point the resulting message (after appending with
padding bits and N) has a length that is an exact multiple of 512 bits.

**Option I-2: Input messages may be longer than 65536 bits**

In this option, appending padding bits is performed as follows: a single "1" bit is
appended to the input message, and then "0" bits are appended so that the length in bits of
the padded message becomes congruent to 448, modulo 512. In all, at least 1 bit and at
most 512 bits are appended. A 64-bit representation of N (the length of the input message,
before the padding bits were added) which is denoted as L is appended to the result of the
previous padding bits process. In the unlikely event that N is greater than $2^{64}$, then only
the low-order 64 bits of N are used. These bits are appended as eight bytes and appended
low-order byte first in accordance with the previous conventions. At this point the resulting message (after appending with padding bits and N) has a length that is an exact multiple of 512 bits.

After the message appending with padding bits and L, the appended message is divided into one or multiple of 512-bit message blocks, notated by $M_1, M_2, \ldots, M_n$. The dividing process is illustrated in Figure 4.2.

- $M$ is the N-bit input message
- $||$ is the concatenation
- $L$ is a 16-bit or 64-bit data segment to indicate the length of the input message $M$
- The number of $0 \nu$ is the least non-negative integer that satisfy $N + 16$ (or 64) + $1 + \nu \equiv 0 \text{ mod } 512$

**Figure 4.1: The Appending Process**

**Figure 4.2: The Dividing Process**
Figure 4.3: The Compression Process
**Step 2: Compression**

Figure 4.3 illustrates the compression process. The first 512-bit message block $M_1$ and offset number (offset is a non-negative integer) are inputted to initialize the internal state $S$ as follows. Note that the first 512-bit message block $M_1$ functions as the inputted key to the KSA algorithm.

$$\text{STATE}_{M_1} = \text{PRGA}^*(\text{offset, KSA}(M_1))$$

Then the algorithm $\text{PRGA}^*$ updates the internal state $S$ depending on $\text{len}_1$ as below:

$$\text{STATE}_1 = \text{PRGA}^*(\text{len}_1, \text{STATE}_{M_1})$$

where $\text{len}_1 = \{ M_1 \text{ mod } 32 \text{ or } 256 \text{ if } (M_1 \text{ mod } 32 \text{ or } 256) \neq 0 \text{ offset} \text{ if } (M_1 \text{ mod } 32 \text{ or } 256) = 0$.

For $k > 1$ ($k = 2, 3 \ldots n$), the internal states $S$ are updated step by step as follows:

$$\text{STATE}_{M_k} = \text{KSA}^*(M_k, \text{STATE}_{k-1})$$

$$\text{STATE}_k = \text{PRGA}^*(\text{len}_k, \text{STATE}_{M_k})$$

where $\text{len}_k = \{ M_k \text{ mod } 32 \text{ or } 256 \text{ if } (M_1 \text{ mod } 32 \text{ or } 256) \neq 0 \text{ offset} \text{ if } (M_1 \text{ mod } 32 \text{ or } 256) = 0$.

In the compression process, there are two default options about modulo, one is to modulo 32, and another is to modulo 256. Choosing which option is the matter of performance and implementation. In the compression process, how many times $\text{PRGA}^*$ runs is controlled by $\text{len}_k$. Regarding how to calculate “$M_k \text{ mod } 32$” or “$M_k \text{ mod } 256$”, the default method is very simple, which only need to consider the 5 or 8 least significant bits of $M_k$ as an integer number. An alternative enhanced method is provided in the
following to calculate “\(M_k \mod 32\)” or “\(M_k \mod 256\)”: every 8 digits binary can convert to a non-negative integer between 0 and 255, or every 16 digits binary can convert to a non-negative integer between 0 and 65535. Since the length of \(M_k\) is 512 bits, \(M_k\) can be divided into 64 1-byte blocks (or 32 2-bytes blocks), then add them up, modulo 32 or 256 to calculate an integer as the result of “\(M_k \mod 32\)” or “\(M_k \mod 256\)”. For example, dividing \(M_k\) into 64 1-byte blocks, such as 00000000 00000001 00000010 \(\ldots\), the result of \((M_k \mod 32)\) is \((0 + 1 + 2 + \cdots) \mod 32\) and the result of \((M_k \mod 256)\) is \((0 + 1 + 2 + \cdots) \mod 256\).

**Step 3: Output**

The output of the compression step is a 256-byte state \(\text{STATE}_n\) and it can be the final hash value. However in order to increase the security strength and to reduce the size of the final hash value, the output step and truncation step are suggested. This means the output step and truncation step are optional, and after completing the compression step we can call the output function and then truncation function by inputting \(\text{STATE}_n\) to produce a 128-bit or 256-bit final hash value.

The output process is performed as follows: Use the final output of the compression step (256-byte \(\text{STATE}_n\)) as the key to KSA, followed by PRGA operation to generate a 512 bytes output. Discard the first 256 bytes and only keeps the last 256 bytes in the calculation. To XOR \(\text{STATE}_n\) with the last 256 bytes of the PRGA output to generate a 256-byte output which is denoted as \(O_{256}\). The output function is illustrated in Figure 4.4.
Step 4: Truncation

In order to reduce the size of the final hash value, many approaches can be used in designing a hash function. We prepared two default options for the truncation process in RC4-BHF. Option O-1 is to select the least significant bit of each byte of $O_{256}$ as the hash value, and this 256-bit long value is the final hash function output. Option O-2 is to select the least significant bit of each odd number or even number byte of $O_{256}$ as the final hash value (odd number case or even number case), and the final hash function output is 128-bit long. Choosing which option is the matter of performance and implementation. Figure 4.5, Figure 4.6 and Figure 4.7 illustrated these options.
Figure 4.5: The Truncation Process (256 bits Hash Value)

Figure 4.6: The Truncation Process (128 bits Hash Value, Odd Number Case)

Figure 4.7: The Truncation Process (128 bits Hash Value, Even Number Case)
4.2 Use RC4-BHF to Construct Hash Chains

A one-way hash chain is an important cryptographic mechanism which can be used widely in many security applications. For example, a hash chain is a method to produce many one-time keys or session keys from a single key in the key management, and a hash function can be applied successively to additional pieces of data in order to record the chronology of data's existence for non-repudiation [74]. As one-way hash chains are very efficient to verify, they became the mechanism to design security applications for wireless sensor and other ultra-low power device networks, as their low-end microprocessors can compute a one-way function within milliseconds, but would require tens of seconds or up to minutes, or even impossible to generate and verify a traditional digital signature [75]. The proposed construction for one-way hash chains is very efficient and is designed for ultra-low power devices.

By using the proposed secure data transmission protocol which will be introduced in Chapter 5, an RC4 base key (hereinafter referred to as a base key) is required as an input to generate a key stream for data encryption or decryption. A new base key will be needed when requested. Therefore it needs to find a way to generate a bunch of base keys, and it would be beneficial if the new base key can be verified easily by the current key or a previous key. A one-way hash chain is an ideal mechanism to provide these.

A one-way function $F: K_j = F(K_{j+1})$ can be used to generate a one-way key chain $(K_n, K_{n-1}, \cdots, K_1, K_0)$. Through this function, anybody can compute forward (e.g.,
computing $K_0, \ldots, K_j$ from a given $K_{j+1}$) but nobody can compute backward (e.g., computing $K_{j+1}$ for only given $K_0, \ldots, K_j$) in the key chain. Because of the forward and backward property of RC4 states, RC4 cannot be used directly to generate such a base key chain, but RC4-BHF is a one-way function which can be used.

The process of the generation and distribution of the base key chain, as well as of the self-authentication of the keys, are illustrated in the following. The base key chain is generated by a sender or trusted third party, and a new base key, which is picked up in sequence from the one-way base key chain, is distributed to receivers when a new base key is needed. The sender or trusted third party first randomly generate a 256-byte key as the last key $K_n$. The rest of the base keys in the one-way base key chain are generated by successively applying RC4-BHF $F: K_j = F(K_{j+1})$. In order the RC4-BHF to generate a 256-byte output as a new base key, the truncation process of RC4-BHF will not be applied. In this approach, the base key chain is generated in the order of $K_n \rightarrow K_{n-1} \rightarrow \cdots \rightarrow K_1 \rightarrow K_0$. We assume $n$ is large enough so the key chain is sufficiently long in relation to the duration of the data transmission. The distribution order of the base keys is from $K_0, K_1, K_2, \ldots$ to $K_n$.

In the one-way base key chain generated as above, the keys are self-authenticated. A receiver can very easily and efficiently authenticate a subsequent key in the one-way base key chain by only using a previous authenticated key. For example, as soon as the receiver receives a new base key $K_i$, it can authenticate the new base key by its current
base key $K_{i-1}$. The new base key is verified once the two keys match by applying RC4-BHF $F$: $K_{i-1} = F(K_i)$. That is, a receiver can use its current key or any of the previous keys which are in the key chain to verify a new disclosed key.

By using the proposed secure data transmission protocol, an offset number is also required and will need to be updated when requested. Similar to the one-way RC4 base key chain, we can also use RC4-BHF to construct a one-way offset number chain.

### 4.3 Other Cryptographic Mechanisms

Perhaps the most versatile cryptographic algorithm is the cryptographic hash function, which can be used in a wide variety of security applications and network security protocols [30]. We can construct many cryptographic mechanisms based on RC4-BHF. Benefits from the high efficiency and simplicity of RC4-BHF, the cryptographic mechanisms based on RC4-BHF can also be efficient and simple, compared to using other hash functions. This section is to illustrate a couple examples of constructing cryptographic mechanisms based on RC4-BHF.

One of the examples is to construct a message authentication function based on RC4-BHF. We have learned the cryptographic hash function and message authentication function in Section 2.3. While message authentication functions are similar to cryptographic hash functions, they are based upon a hash function with the extension that requires using a secret key.
There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The one that has received the most support is HMAC [76]. HMAC treats the hash function as a “black box” and any hash function can be embedded into HMAC. RC4-BHF can be embedded into HMAC directly to construct a message authentication function. Although it is portable, there is a disadvantage for the HMAC approach because the execution time is twice as long: HMAC involves two executions of the underlying hash function for each output block. An idea to construct a message authentication function by using modified RC4-BHF is proposed, which are introduced in the following.

In the proposed message authentication function, the other definitions are the same as in the RC4-BHF, except adding one more definition which is secret key K. The idea is only need to modify the compression process to input the secret key K, but keep the structure of the compression process without change, which is illustrated in Figure 4.8. The recommended length for the secret key is equal or longer than 128 bytes. The other three steps are kept the same as in RC4-BHF. Because of the characteristics of the underlying function RC4-BHF, the proposed message authentication function is designed for 8-bit machines, and is very efficient and secure.
Another example of constructing cryptographic mechanism based on RC4-BHF is to use RC4-BHF to construct a pseudorandom number generator. According to the analysis which will be presented in Section 6.1, RC4-BHF meets very high standard tests for pseudorandomness, which makes it a good primitive to construct a pseudorandom number generator (PRNG). There are many approaches (e.g., [77, 78, 79, 80]) that have been proposed to construct a PRNG through a cryptographic hash function. We can
directly use these approaches combined with RC4-BHF to construct a PRNG, or introduce a new approach based on RC4-BHF. Because of the characteristics of the underlying function RC4-BHF, the PRNG based on RC4-BHF should be very efficient and secure, and is suitable for 8-bit machines.

RC4-BHF can also be used as an important part to construct many other cryptographic mechanisms and security applications, such as digital signature, one-way password file generation, and intrusion and virus detection. Because of the limited space in this dissertation, no more examples are given for each possible application. Benefits from the high efficiency and simplicity of RC4-BHF, and the structure which is very suitable for 8-bit machines, we can use RC4-BHF to construct the cryptographic mechanisms and security applications more simply and efficiently.

4.4 Summary

A new hash function, called RC4-BHF, has been proposed in this chapter. RC4-BHF is designed for 8-bit machines, and to be both fast and secure. RC4-BHF plays a fundamental role in data integrity, authentication, access control, and random number generation for the proposed secure data transmission protocol, which is introduced in Chapter 5.

RC4-BHF has been proposed in the first section of this chapter. How to use RC4-BHF to construct one-way hash chains, such as RC4 base key chain which is
applicable for the proposed secure data transmission protocol has been discussed in the second section. RC4-BHF can be used to construct many other cryptographic mechanisms, and a couple of examples have been illustrated in the third section of this chapter.
Chapter 5

SDTP, a Secure Data Transmission Protocol for Secure Data Unicast, Secure Data Broadcast and Secure Data Multicast in Wireless Ultra-Low Power Device Networks

This chapter proposes a secure data transmission protocol (SDTP) which is to provide necessary protection of network security, includes data cryptography, data integrity, authentication, and access control in wireless ultra-low power device networks. How SDTP offers secure data unicast, secure data broadcast, and secure data multicast are introduced respectively in each section of this chapter.

5.1 Notations and Terminologies

Firstly, the notations and terminologies which will be used in this chapter are introduced in this section.

- Current RC4 State (CRS): For a stream cipher, the encryption of each digit is dependent on the current state, and the decryption to that digit has to use the same
state. This means once a digit of the plaintext is XORed with a digit of the key stream which is dependent on the current RC4 state for encryption, that digit has to be decrypted by the same digit of the key stream which is dependent on the same RC4 state. In simple terms, the same RC4 state has to be used for both the encryption and decryption on the same digit. Each participant of the data transmission keeps a storage space in its memory to store its current RC4 state.

- **Offset (O):** Offset is an integer to indicate how many times PRGA runs initially, in order to avoid the statistical bias in the initial portion of the keystream, which is called the insufficient key schedule issue [29, 44, 69] to strengthen RC4. This offset number is different than the offset number in RC4-BHF, but it is suggested using the same value for both, to avoid the duplicate cost to generate and share two offset numbers.

- **Sequence Counter (SC):** The sequence counter is a number which is initially zero and is increased by one for each interval. Both the sender and each receiver have a sequence counter in their memories to record what the latest data packet it handled; and every fixed-length data packet also includes a sequence counter to indicate where this data packet in all data packets handled by the sender. The sequence counter of the sender is increased by one for each new fixed-length data packet. The receiver determines the order of the received data packet by comparing the sequence counter of the data packet it received with its own sequence counter.
- Fixed-length Data Packet: In the proposed SDTP the length of the data packets in the data transmission is fixed. A fixed-length data packet contains a position bit, a sequence counter, a data segment, and a hash value checksum. The default lengths for them are 1-bit, 15-bit, 368-bit and 128-bit respectively, which are subject to change depending on the application. The position bit is a one-bit segment in a fixed-length data packet to indicate if it is the last fixed-length data packet of an input plaintext message. “0” represents false, and “1” represents true. Figure 5.1 indicates a fixed-length data packet.

<table>
<thead>
<tr>
<th>PB</th>
<th>SC</th>
<th>Data Segment</th>
<th>HV</th>
</tr>
</thead>
</table>

Figure 5.1: Fixed-Length Data Packet

### 5.2 Secure Data Unicast

Data unicast is the most common type of network routing to provide data transmission from one node to another. Only two participants are involved in a data unicast, which are denoted as a sender and a receiver. This section introduces how to achieve a secure data unicast.

One of the two participants (usually the participant who has more resource) uses RC4-BHF to generate an RC4 base key chain and an offset number chain before the initial data transmission (a very first data transmission) and stores them in its memory.
Before the initial data transmission or when the base key needs to change, that participant prepares and coordinates to share a new base key and a new offset number with another participant, and we call this process the handshake phase. We assume that participant is the sender in this section.

In the handshake phase, the sender picks up the next RC4 base key from the one-way base key chain, and the next offset number from the one-way offset number chain, and shares them with the receiver. The detail of the generation, distribution and self-authentication of the base key chain and offset number chain are described in Section 4.2. In the case of the base key and offset number change, the new base key and new offset number can be shared with the receiver through SDTP under the encryption with a portion of the key stream generated from the current base key, and then this base key will expire and the new base key will take its place.

Once the handshake phase is finished, which means the new base key and the new offset number are shared by the two participants of the data transmission, both the sender and receiver use the new base key to initialize the state vector S and the initialized state vector is denoted as $S_0$. Once $S_0$ is ready, both participants run offset number times PRGA from $S_0$ to produce a new RC4 state to generate the key stream for the upcoming encryption and decryption. Both participants set their sequence counter to zero ($SC=0$). We call this process an initialization phase (or a re-initialization phase).
Different from many ordinary RC4 based security protocols (e.g., WEP and WPA 1.0) which are to reinitialize the RC4 state for every new message, in SDTP the re-initialization of the RC4 state performed by the key scheduling algorithm (KSA) is carried out only when the base key changes. Please note it is always using at different portion of the key stream to encrypt or decrypt different data packet. It is not necessary to update the offset number very often and we suggest changing the base key and the offset number at the same time, to minimize the frequency of the handshake phase.

Once both participants finish the initialization (or re-initialization) phase, they are ready for the upcoming secure data transmission. In the following, the process executed by the sender and receiver respectively for secure data unicast are introduced.

Figure 5.2 depicts the three-step operations executed by the sender. After these steps, the sender produces one or multiple encrypted fixed-length data packets from an input plaintext message.

(1) The sender divides the plaintext message into one or a multiple of 368-bit data segments, and the padding bits may need to be appended to the last data segment. The padding bits (denoted as PA) in the last data segment is performed as follows: an end character is appended to the end, and then "0" bits or random bits are followed so that the length of the last data segment becomes 368 bits. How to use RC4-BHF to construct a pseudorandom number generator has been briefly discussed in Section 4.3. After the dividing and padding are finished, the sender assigns a position bit and
a sequence counter to each of the data segments, and then updates its sequence counter.

(2) For each data packet with position bit and sequence counter, the sender calculates the hash value checksum through using RC4-BHF with the inputs of the position bit, sequence counter and unencrypted data segment, and then appends the hash value checksum to form a complete unencrypted fixed-length data packet.

(3) Once the second step is finished for all fixed-length data packets which are associated to the same input plaintext message, the sender produces the encrypted fixed-length data packets through only encrypting the data segment and hash value checksum (position bit and sequence counter are kept unencrypted).

Figure 5.3 depicts the three-step operations executed by the receiver. After these steps, the input plaintext message is recovered from the corresponding one or multiple encrypted fixed-length data packets. The detailed process is introduced below.

(1) For an encrypted fixed-length data packet which the receiver just received, by comparing the sequence counter of the received data packet with its own sequence counter, the receiver can calculate the right RC4 state from its current RC4 state to decrypt the encrypted data segment and hash value checksum of that received packet, based on the forward and backward property of RC4 states, and then the receiver updates its own sequence counter.
(2) Once the first step is finished, the receiver verifies the data packet by re-computing the hash value checksum and comparing it with the hash value checksum decrypted in the first step. If the two hash value checksums are the same, the receiver has successfully authenticated the packet. If the two do not match, either someone is trying to impersonate the sender, the message itself has been altered, or an error occurred during transmission.

(3) After all fixed-length data packets which are associated with one input plaintext message are decrypted and their hash value checksums are verified, the receiver restores them to produce an input plaintext message.

The approach depicted above for one-way secure data unicast can be extended to achieve two-way data transmission, which is a form of transmission in which both parties are involved to transmit data. In order to achieve the two-way data transmission, the RC4 state which is ready to generate the key stream for the upcoming encryption and decryption, after finish the handshake phase will run to forward or backward, and each direction is for one of the two participants to use.

For example, once the handshake phase is finished, both the sender and receiver use the new base key to initialize the state vector S to get $S_0$. Once $S_0$ is ready, both participants run offset number times PRGA from $S_0$ to produce a new RC4 state to generate the key stream and both participants set their sequence counter to zero. From this new RC4 state, one participant run forward by PRGA to generate the key stream to
encrypt, and the sequence counter which is positive is increased by one for each interval (e.g., 1, 2, 3, 4…); another participant run backward by IPRGA to generate the key stream to encrypt, and the sequence counter which is negative is decreased by one for each interval (e.g., -1, -2, -3, -4…). Through this setup, both participants can still keep only one RC4 state and only one sequence counter in their memories, and the sequence counter can be negative as well as positive. Once receiving a data packet, a participant can compare the sequence counter of the received data packet with its own, and calculate the right RC4 state from its current RC4 state to decrypt. Please note the participant updates its own sequence counter accordingly after encrypting or decrypting a data packet.
Figure 5.2: The Operations Executed by the Sender
Figure 5.3: The Operations Executed by the Receiver
5.3 Secure Data Broadcast and To Handle Delayed or Lost Packets

Data broadcast is one of the most common types of network routing to provide efficient delivery of data from a source to all the other participants in the network. In a large network environment, broadcast is a very useful service to deliver data amongst all network participants. Example applications of network broadcast are software updates, network queries, and command dissemination. This section introduces how SDTP achieves secure data broadcast.

As mentioned in Section 1.3, there are some research proposals that try to use RC4 to achieve secure data transmission, such as the State Based Key Hop protocol [31]. Because of the strong synchronization requirement, they cannot be used for broadcast and multicast. A network protocol which supports data broadcast should not require strong synchronization, which means it should be able to handle the delayed or lost packets. In a network environment, data packets received from the network may be out-of-order, which means a receiver receives data packets from the network is not always in sequence. For example, in an anisomerous communication, either from a server to a client or from a base station to a sensor node, is where the sender has high capability and the receiver has low capability, if the sender focuses solely on processing and sending data, the receiver may not be able to handle the packets in a very short period of time. Moreover, data packets in the network are often out-of-order because the delivery paths may be different.
even for the same participants but with different data packets. This poses a significant
difficulty to apply a state-based stream cipher in these applications.

SDTP can handle the issues of delayed or lost packets. Once a receiver receives a
data packet, it does not matter if it matches the packet orders or not, the receiver can
compare the sequence counter of the received data packet with its own, and calculate the
right RC4 state from its current RC4 state to decrypt that data packet.

Figure 5.4: An Example of the Data Broadcast through Multi-Hop Connection

Figure 5.4 is an example of a data broadcast through multi-hop connection. The
bigger node is a base station, and the smaller node is a sensor node. There are one base
station BS and eight sensor nodes SN₁, SN₂, SN₃, SN₄, SN₅, SN₆, SN₇ and SN₈.

In the network, a receiver may receive data packets in an arbitrary order. We assume
base station BS sends four data packets P₁, P₂, P₃, and P₄ to all sensor nodes and the
sending order is P₁ → P₂ → P₃ → P₄. Sensor nodes SN₁, SN₃, SN₅, SN₆, SN₇ and SN₈
receive the packets in the order of the sending order, but SN₂ receives in the order of
$P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3$, and SN$_4$ receives in the order of $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2$. The following shows how SDTP handles the delayed packets to support data broadcast.

1. The base station BS sends four data packets P$_1$, P$_2$, P$_3$ and P$_4$ to all sensor nodes. The sending order is $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$.

2. The sensor nodes SN$_1$, SN$_3$, SN$_5$, SN$_6$, SN$_7$ and SN$_8$ receive the four data packets in the same order as the sending order, and then they can decrypt the packets as usual.

3. The sensor node SN$_2$ receives packets in the order $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_3$. SN$_2$ decrypts P$_1$ and P$_2$ as usual first. Since P$_4$ is received prior to P$_3$, SN$_2$ compares the sequence counter of P$_4$ with its own sequence counter, and calculate the right RC4 state from its current RC4 state to decrypt P$_4$. Once P$_3$ is received, SN$_2$ compares the sequence counter of P$_3$ with its own sequence counter, and calculates the right RC4 state from its current RC4 state to decrypt P$_3$. Please note SN$_2$ updates its own sequence counter after finishing each data packet.

4. The sensor node SN$_4$ receives packets in the order $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2$. SN$_4$ decrypts P$_1$ as usual first. Once P$_3$ is received, SN$_4$ compares the sequence counter of P$_3$ with its own sequence counter, and calculate the right RC4 state from its current RC4 state to decrypt P$_3$. In the following SN$_4$ decrypts P$_4$ as usual, once received. Upon receiving P$_2$, SN$_4$ compares the sequence counter of P$_2$ with its own, and calculate the right RC4 state from its current RC4 state to decrypt P$_2$. Please note SN$_4$ updates its own sequence counter after finishing each data packet.
In the case of packet lose, a list for potential missing packets (PMPL) is maintained by the base station to record the information of the potential lost packets for all sensor nodes. The information is reported by the sensor nodes. In order for the base station to minimize the process overhead, the sequence counters of the missing packets are needed and the resending order is based on the sequence counters stored in PMPL. For example, P_2, P_4, P_5, and P_1 have been reported missing by four different sensor nodes and the information is recorded in PMPL, it does not matter the reporting order, the base station will wait until a predefined time to process all missing packets in PMPL at once, by ascending order. The processing order for the example will be \( P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_5 \).

In the network, packets may get lost so sometimes a receiver may not receive all packets. Assume base station BS sends 4 data packets P_1, P_2, P_3 and P_4 to all sensor nodes and the sending order is \( P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \). S_4 does not receive P_2, and S_N2 does not receive P_3. The following shows how SDTP handles the lost packets to support secure data broadcast.

(1) The base station BS sends four data packets P_1, P_2, P_3 and P_4 to all sensor nodes.

The sending order is \( P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \).

(2) The sensor nodes SN_1, SN_3, SN_5, SN_6, SN_7 and SN_8 receive the four data packets in the same order as the sending order, and then they can decrypt the packets as usual.

(3) The sensor node SN_2 receives packets in the order of \( P_1 \rightarrow P_2 \rightarrow P_4 \) and for some reason it does not receive P_3. SN_2 decrypts P_1 and P_2 as usual first. Once P_4 received,
SN₂ compares the sequence counter of P₄ with its own sequence counter, since P₄ is received prior to P₃, it calculates the right RC4 state from its current RC4 state to decrypt P₄ and then to update its own sequence counter. In the meantime, SN₂ records this to its own list that it does not receive P₃, and wait for a predefined time to report to BS to add P₃ to the PMPL about SN₂ does not receive P₃ if P₃ is still not received by SN₂ at that time.

(4) The sensor node SN₄ receives packets in the order of P₁ → P₃ → P₄ and for some reason it does not receive P₂. SN₄ decrypts P₁ as usual first. Once P₃ is received, SN₄ compares the sequence counter of P₃ with its own sequence counter, since P₃ is received prior to P₂, it calculates the right RC4 state from its current RC4 state to decrypt P₃ and then to update its own sequence counter. In the meantime, SN₄ records this to its own list that it does not receive P₂, and waits for a predefined time to report to BS to add P₂ to the PMPL about SN₄ does not receive P₂, if P₂ has still not received by SN₄ at that time. Once P₄ is received, SN₄ decrypts it as usual.

(5) The base station BS receives the requests from SN₂ and SN₄ that SN₂ does not receive P₃ and SN₄ does not receive P₂, and records the requests in PMPL. At a certain time, BS processes all requests in PMPL to retrieve the lost packets. For each lost packet, it compares the sequence counter of the lost packet with its own sequence counter, and calculates the right RC4 state from its current RC4 state to
encrypt the data packet. Once \( P_2 \) and \( P_3 \) are ready, BS sends them in the order of \( P_2 \rightarrow P_3 \) to SN\(_2\) and SN\(_4\) specifically, indicates that it is for resending.

(6) Once the sensor node SN\(_2\) receives the lost packet \( P_3 \), it compares the sequence counter of \( P_3 \) with its own sequence counter, and calculates the right RC4 state from its current RC4 state to decrypt \( P_3 \). The same with SN\(_4\), once it receives lost packet \( P_2 \), it compares the sequence counter of \( P_2 \) with its own sequence counter, and calculates the right RC4 state from its current RC4 state to decrypt \( P_2 \).

In summary, SDTP does not require a strong synchronization and is applicable for secure data broadcast.

### 5.4 Secure Data Multicast and Dynamic Access Control

Data multicast is one of the most common types of network routing to provide efficient delivery of data from a source to certain participants in a network. In a large network environment, multicast is a very useful service to deliver data to certain network participants. Example applications of data multicast are network queries or command target specific participants in the network. This section introduces how SDTP achieves secure data multicast.

Data multicast has to achieve dynamic access control, more specifically it should be able to handle frequent key changes. Compared to the data unicast and data broadcast, data multicast may require more frequent key changes because of the member movement,
such as member joining or member leaving. Each time, when a new member joins, an existing member leaves, or in some cases a member moves from one location to another in the network, all multicast members may need to have a new session key and a new offset number. When a member joins, in most cases applications do not allow the new member to be able to retrieve the previous multicast data; and when a member leaves, applications do not allow the member who left to retrieve the new multicast data. As introduced in Section 1.1, the ad-hoc sensor network is an example of such wireless sensor networks in that the nodes are highly mobile.

Figure 5.5: An Example of a Hierarchical Multicast Tree
For a secure data multicast, participant nodes physically nearby are grouped together and we call them subgroup in the multicast tree. The idea of the subgroup is to partition a large multicast group into a hierarchy of subgroups. There are three different types of participates in the multicast tree: Base Station (BS), Power Sensor Node (PSN) and regular Sensor Node (SN). PSN is physically the same as SN but PSN can pass data on to other nodes, and SN is the terminal node which can only receive data. As data transmitter, PSN receives data packets from the BS or another PSN, and passes the data packets on to its children PSNs or SNs in its subgroup. SNs only receive data from a PSN.

Figure 5.5 is an example of a hierarchical multicast tree. The BS of the multicast group is the root of the multicast tree, and subgroup 0 is composed of BS, PSN₁, PSN₂, and PSN₃. PSN₄ resides in two subgroups. Subgroup 4 is called PSN₄’s child subgroup, whereby PSN₄ serves as the subgroup server, while subgroup 3 is PSN₄’s parent subgroup, in which PSN₄ is PSN₃’s client. Each PSN has only one parent subgroup and one or more child subgroups. Each subgroup has one subgroup server PSN, and may have one or more SNs. For simplicity, we assume that BS has no change in the tree, a PSN can only join to the tree, and a SN can join, leave or move in the tree.

The generation, transmission, and storage of keys, which are known as key management, are important, especially for data multicast as it involves member movement more often. Some studies [41, 81, 82] show that most practical network security attacks are achieved through finding security vulnerability in key management,
not through attacking cryptographic algorithms. It means that a proper mechanism for access control is an important factor for network security.

The following introduces how the session keys are generated, stored, transmitted and authenticated in the multicast tree for data multicast in the case of the tree initializes, a member leaves, a new member joins, or a member moves.

All PSNs and SNs are authorized by the BS. The BS maintains a member management list, which contains the device id, individual key, and routing information for every PSN and SN in the multicast tree. BS has to receive the above information from each SN or PSN before placing them into the multicast tree. The device id and individual key are only shared between the corresponding sensor node (PSN or SN) and the BS, without sharing with anyone else.

**When the tree initializes**

1. BS generates an RC4 base key chain and an offset number chain. The number of base keys and offset numbers should be large enough in the base key chain and offset number chain, which are sufficiently long in relation to the duration of the entire multicast lifetime. The detailed description of the one-way RC4 base key chain and the one-way offset number chain are in Section 4.2.

2. BS picks up the corresponding RC4 base key from the RC4 base key chain and the corresponding offset number from the offset number chain, and uses the new base key to initialize the state vector S. Once $S_0$ is ready, BS runs offset number times
PRGA from $S_0$ to produce a new RC4 state and sets its sequence counter to zero (SC=0). BS can thereafter generate the key stream from this RC4 state for future data multicast.

(3) For each SN or PSN, BS obtains its device id and individual key from the member management list, encrypts the base key and offset number with the individual key, and sends them to the corresponding SN or PSN.

(4) Each SN or PSN gets the corresponding base key and offset number through the decryption with its own individual key. The SN or PSN uses the received base key to initialize the state vector $S$. Once $S_0$ is ready, the PRGA runs the number of times equal to the offset number from $S_0$ to produce a new RC4 state and sets its sequence counter to zero (SC=0). The SN or PSN can thereafter generate the key stream from this RC4 state for future data multicast.

**When a member leaves**

(1) BS picks up a corresponding new offset number from the offset number chain and records its current sequence counter value, runs the PRGA the number of times equal to the new offset number from its current RC4 state to generate a new RC4 state. BS resets its sequence counter to zero (SC=0) and thereafter it can generate the key stream from this new RC4 state for future data multicast.

(2) For each SN or PSN which is still in the multicast tree, BS obtains its device id and individual key from the member management list, encrypts the new offset number
and recorded sequence counter value with the individual key, and sends them to the corresponding SN or PSN. The BS resets its SC to zero (SC=0).

(3) Each SN or PSN gets the new offset number and sequence counter value through the decryption with its own individual key. The SN or PSN can self-authenticate the new offset number easily as explained in Section 4.2. The SN or PSN compares the received sequence counter with its own sequence counter, and calculates the right RC4 state from its current RC4 state, and runs the PRGA the number of times equal to the new offset number from that RC4 state to produce a new RC4 state for the upcoming key stream generation for future multicast. The SN or PSN resets its SC to zero (SC=0).

**When a new member joins**

(1) BS picks up a corresponding new offset number from the offset number chain and records its current sequence counter value, and runs the PRGA the number of times equal to the new offset number from its current RC4 state to generate a new RC4 state. BS resets its sequence counter to zero (SC=0) and thereafter it can generate the key stream from this new RC4 state for future data multicast.

(2) For old SNs and PSNs, BS encrypts the new offset number and recorded sequence counter value with the old session key stream, and sends to all old multicast members.
(3) For each old SN or PSN, it gets the new offset number and sequence counter value through the decryption with the old session key stream. The SN or PSN can self-authenticate the new offset number. The SN or PSN compares the received sequence counter with its own sequence counter, and calculates the right RC4 state from its current RC4 state, and runs the PRGA the number of times equal to the new offset number from that RC4 state to produce a new RC4 state for the upcoming key stream generation for future multicast. The SN or PSN resets its SC to zero (SC=0).

(4) For the new joined SN, BS obtains its device id and individual key from the member management list, encrypts the new RC4 state which is the RC4 state right after runs the PRGA the number of times equal to the new offset number with the individual key, and sends to the new joined SN. The new joined SN gets the RC4 state through the decryption with its own individual key and sets its sequence counter to zero (SC=0). The new joined SN thereafter can generate the key stream from this RC4 state for future data multicast.

**When a member moves**

For some applications, a member may move from one location to another in the multicast tree. If a SN moves from one subgroup to another in the multicast tree, the key stream for multicast will not change, but the SN needs to contact BS to update its routing information in the member management list.
In summary, SDTP supports dynamic access control and is applicable for secure data multicast. SDTP can handle how session keys are generated, stored, transmitted and authenticated in the multicast tree for data multicast in the case where the tree initializes, a member leaves, a new member joins, or a member moves. This is also applicable when several new members join, several members leave, or several members move at the same time.

5.5 Summary

This chapter proposed a secure data transmission protocol, called SDTP, which provides necessary protection for network security, including data cryptography, data integrity, authentication, and access control to achieve secure data transmission in wireless ultra-low power device networks. In the first section of this chapter, the notations and terminologies for SDTP have been introduced. In the following, how SDTP achieves secure data unicast, secure data broadcast and secure data multicast have been presented in each section respectively.
Chapter 6

Analysis

This chapter analyzes the proposed hash function RC4-BHF and the proposed secure data transmission protocol SDTP, and discusses their support for wireless ultra-low power device networks.

6.1 The Analysis of RC4-BHF

This section analyzes the proposed hash function RC4-BHF. We have introduced the basic knowledge of cryptographic hash function and its requirements and have learned the general structure of a typical secure hash function in Section 2.3. RC4-BHF followed the general structure of a typical secure hash function. RC4-BHF has been implemented on real sensor nodes. The implementation is introduced in Section 6.3, and the source code of RC4-BHF in C programming language is provided in Appendix B.

In order to have an intuitive understanding of RC4-BHF, in the following some random ASCII input and the corresponding 256 bits RC4-BHF hash value in hexadecimal
are listed. The random ASCII inputs are from [59] and [83], which have been used to describe MD4 and MD5. The offset number in the following examples is set to 100.

The following demonstrates a 43-byte ASCII input and the corresponding RC4-BHF output:

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>The quick brown fox jumps over the lazy dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>AD EE 57 71 EA 73 02 A7 94 33 EF 29 45 84 0B 22 6F 22 48 BE 2E BA 31 E7 4D CF DF 36 32 14 6B 64</td>
</tr>
</tbody>
</table>

Even a small change in the message will, with overwhelming probability, result in a completely different hash value, e.g. changing d to c in the input message:

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>The quick brown fox jumps over the lazy cog</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>4E 5C 03 30 AC 17 27 C5 1F DC 71 8F 65 6F 7E C0 31 5A DB 7B 1B 65 CB E6 D0 B1 F9 47 7F 01 A5 11</td>
</tr>
</tbody>
</table>

The hash value of the zero-length string is:

| The ASCII input to RC4-BHF | 0C C3 6F 5D 52 46 CC 9C 6C A9 2E 30 E7 C2 BA F5 18 1D A0 EE 6C 26 1E 21 1B 44 AA 91 A5 10 D3 DD |

Other random RC4-BHF test vectors are listed below:

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>62 6D 9E 85 A8 66 FF 86 AB 6C B6 E0 6D 2F 8C ED 47 31 9F C1 AB F6 FD E5 D0 0B CF 01 7A 05 A6 4A</td>
</tr>
<tr>
<td>The ASCII input to RC4-BHF</td>
<td>abc</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>AA 9E 74 C8 D0 30 74 F0 09 75 FC 01 54 4B 29 24 FA 95 EC 7F 42 34 A7 AF D3 E3 39 8F B4 41 D6 68</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>message digest</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>3E 52 6B 6D 80 CC 9F BD 3C 91 FE 93 C5 EA E3 54 A4 F1 2A 55 E6 24 01 EA 82 D7 22 07 71 E5 A5 E3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>abcdefghijklmnopqrstuvwxyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>2D 28 05 EF 09 22 B1 A9 D3 28 96 F6 CB 80 BB 50 F2 98 9B 07 84 8F BF 37 06 88 96 88 FB 08 99 A3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>B9 7F BA 40 23 3F ED DF 8D 83 88 38 99 9D 42 35 9C 70 71 B1 45 D7 63 27 C6 05 6E FF C2 F8 6C 93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The ASCII input to RC4-BHF</th>
<th>1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890</th>
</tr>
</thead>
<tbody>
<tr>
<td>The corresponding RC4-BHF hash value</td>
<td>6B FD 36 93 C1 7A 8F DF 1F 42 7B 4A 82 FB AA 79 0C EB 52 BE 6E 67 D1 8D 14 98 C4 8D D2 7C A8 BE</td>
</tr>
</tbody>
</table>

In the following, how RC4-BHF meets the requirements for a cryptographic hash function and how it can resist the common attacks against cryptographic hash function are discussed.

1) **Variable input size.** The input of RC4-BHF is of arbitrary length, by default not greater than $2^{64}$ bits. There are two default options for the input of the arbitrary length, which are the input of maximum length of 65536 bits (option I-1) or the input of maximum length of $2^{64}$ bits (option I-2). The option I-1 is for the application which the
input message is always shorter or equal to 65536 bits, and the option I-2 is for the application which the input message may be longer than 65536 bits, but shorter than $2^{64}$ bits. It is unlikely that any input message will be greater than $2^{64}$, but the range of the allowed input size can be adjusted by only changing the default bitlength to indicate the length of the input message $L$.

2). Fixed output size. The output of the RC4-BHF is fixed-length. The output of the compression process $\text{STATE}_a$ is fixed to be 256 bytes and it can be the final hash value. The output and truncation steps are optional but are suggested, and the output of the Option O-1 is fixed to be 256 bits and the output of the Option O-2 is fixed to be 128 bits.

3). Pseudorandomness. A hash function needs to meet the standard tests for pseudorandomness. Two standard tests have been conducted for pseudorandomness against RC4-BHF, which are Matlab’s runs test for randomness runstest(x) [84] and Maurer’s universal statistical test [85].

For preparing the test data, firstly three groups of data have been prepared, and each group contains 1024 randomly generated strings of arbitrary length. Secondly, we run RC4-BHF, and each of the 3072 randomly generated strings is the input to RC4-BHF, and obtained 3072 256-bit RC4-BHF hash values (Option O-1), in 3 groups, and each group contains 1024 of them. Thirdly, for each group of the hash value, two kinds of files have been prepared. One is for each byte of the 256-bit hash value as a test unit, and therefore in the file it includes $32 \times 1024 = 32768$ test unit and each test unit is 1-byte.
Another is for each of the 256-bit hash value as a test unit, and therefore in the file it includes 1024 test unit and each test unit is 32-byte. Six files have been obtained: TestData-1.data, TestData-3.data, and TestData-5.data are for each test unit which is 1-byte, and TestData-2.data, TestData-4.data, and TestData-6.data are for each test unit which is 32-byte. We also merged each of the three files with the same size test units and obtained two bigger files, based on the file with 1-byte test unit (TestData-7.data) and the file with 32-byte test unit (TestData-8.data).

For conducting the tests, firstly we run the Matlab’s runs test for randomness runstest(x) against each of the eight files, and all eight files passed the randomness test. Thereafter, we run the Maurer’s universal statistical test against each of the files the test unit is 32-byte: TestData-2.data, TestData-4.data, TestData-6.data and TestData-8.data. The test tool for Maurer’s universal statistical test is included in the Crypto++ Library [86] and the version of the Crypto++ Library we used is 5.6.2. The value range of the Maurer test value is from 0 to 1, and the best test result of the Maurer test value is 1. The test value closest to 1, the more random the test data is, which means the test data meet a very high standard of randomness. In reverse, the test value closest to 0, the less random the test data is. The Maurer test values for the four files TestData-2.data, TestData-4.data, TestData-6.data and TestData-8.data are listed in Table 6.1.
Table 6.1: The Maurer Test Value of the Files Randomly Generated by RC4-BHF

<table>
<thead>
<tr>
<th>Test Data File Name</th>
<th>Maurer test value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestData-2.data</td>
<td>1</td>
</tr>
<tr>
<td>TestData-4.data</td>
<td>0.998923</td>
</tr>
<tr>
<td>TestData-6.data</td>
<td>0.999566</td>
</tr>
<tr>
<td>TestData-8.data</td>
<td>0.999537</td>
</tr>
</tbody>
</table>

We also used a pseudorandom number generator ANSI X9.17 [87] which was approved by the American National Standards Institute and the Federal Information Processing Standards to randomly generate 10 files of random numbers and then use the Maurer’s universal statistical test against these 10 files. The test result is listed in Table 6.2.

Table 6.2: The Maurer Test Value of the Files Generated by ANSI X9.17 PRNG

<table>
<thead>
<tr>
<th>File Name</th>
<th>Maurer test value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X917-1.data</td>
<td>0.99992</td>
</tr>
<tr>
<td>X917-2.data</td>
<td>0.997975</td>
</tr>
<tr>
<td>X917-3.data</td>
<td>1</td>
</tr>
<tr>
<td>X917-4.data</td>
<td>0.999578</td>
</tr>
<tr>
<td>X917-5.data</td>
<td>0.997869</td>
</tr>
<tr>
<td>X917-6.data</td>
<td>0.999612</td>
</tr>
<tr>
<td>X917-7.data</td>
<td>1</td>
</tr>
<tr>
<td>X917-8.data</td>
<td>0.998504</td>
</tr>
<tr>
<td>X917-9.data</td>
<td>0.999604</td>
</tr>
<tr>
<td>X917-10.data</td>
<td>0.999848</td>
</tr>
</tbody>
</table>
From the above experimental results, we can conclude that the output of RC4-BHF meets very high standard tests for pseudorandomness. The pseudorandomness of the output RC4-BHF generated is at the same level as the output generated by ANSI X9.17 PRNG. ANSI X9.17 PRNG is one of the strongest (cryptographically speaking) PRNGs and a number of applications employ this technique, including financial security applications [30].

4). **Efficiency.** RC4-BHF is designed to be exceptionally fast, especially on 8-bit processors. RC4-BHF is based on KSA, PRGA, KSA*, and PRGA* and these algorithms are remarkably simple and very efficient. The operations of these algorithms are byte-oriented, which includes exclusive-or operation, byte swap operation, byte comparison operation, modulo operation, and add operation.

Table 6.3 is the speed comparison we conducted to compare the speed with RC4-BHF and some well-known hash functions on 32-bit processors. The tool we used is the benchmark function provided by Crypto++ Library version 5.6.2. The benchmark function of Crypto++ Library version 5.6.2 were coded in C++ and we compiled it with Microsoft Visual Studio Professional 2012, Update 2, and ran on an Intel Core 2 Duo 2.80 GHz processor and 4.0 GB installed memory (RAM) under Windows 7 in 32-bit mode. From the comparison result, we can confirm that RC4-BHF is faster than the listed hash functions on 32-bit processors. Some hash functions are designed for 32-bit or 64-bit processors, but may still be able to run on 8-bit processors with slow performance.
RC4-BHF is designed for 8-bit processors, and should be run a lot faster than these hash functions on 8-bit processors.

Table 6.3: Speed Comparison with RC4-BHF and Some Well-known Hash Functions

<table>
<thead>
<tr>
<th>Hash Function</th>
<th>Speed (MiB/Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>104</td>
</tr>
<tr>
<td>SHA-1</td>
<td>38</td>
</tr>
<tr>
<td>SHA-256</td>
<td>159</td>
</tr>
<tr>
<td>SHA-512</td>
<td>124</td>
</tr>
<tr>
<td>SHA-3-224</td>
<td>8</td>
</tr>
<tr>
<td>SHA-3-256</td>
<td>8</td>
</tr>
<tr>
<td>SHA-3-384</td>
<td>6</td>
</tr>
<tr>
<td>SHA-3-512</td>
<td>4</td>
</tr>
<tr>
<td>Whirlpool</td>
<td>76</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>26</td>
</tr>
<tr>
<td>RIPEMD-320</td>
<td>25</td>
</tr>
<tr>
<td>RIPEMD-128</td>
<td>46</td>
</tr>
<tr>
<td>RIPEMD-256</td>
<td>48</td>
</tr>
<tr>
<td>RC4-BHF</td>
<td>213</td>
</tr>
</tbody>
</table>

As illustrated in Chapter 1, the ultra-low power device, such as sensor node is typically only equipped with 8-bit low-end microprocessor and small amount of memory. Most of the well-known hash functions are designed to run on 32-bit or 64-bit processor so they are not well suited for ultra-low power devices. We listed the selected hash functions and the processors they are designed for in Table 6.4. If memory allows, a small number of hash functions which are designed for 32-bit or 64-bit processors may still be
able to run on 8-bit processors, however, in that case, it would require significantly longer computing time and more power and memory usage. RC4-BHF is designed to run very fast on 8-bit processors and it works very efficiently in the ultra-low power devices.

Table 6.4: Selected Hash Functions and the Processors They are Designed for

<table>
<thead>
<tr>
<th>Hash Function</th>
<th>The Processor It is Designed For</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD4 and MD5</td>
<td>32 bits</td>
</tr>
<tr>
<td>SHA-0</td>
<td>32 bits</td>
</tr>
<tr>
<td>SHA-1</td>
<td>32 bits</td>
</tr>
<tr>
<td>SHA-2</td>
<td></td>
</tr>
<tr>
<td>SHA-224</td>
<td>32 bits</td>
</tr>
<tr>
<td>SHA-256</td>
<td></td>
</tr>
<tr>
<td>SHA-384</td>
<td>32 bits</td>
</tr>
<tr>
<td>SHA-512</td>
<td></td>
</tr>
<tr>
<td>SHA-512/224</td>
<td>64 bits</td>
</tr>
<tr>
<td>SHA-512/256</td>
<td></td>
</tr>
<tr>
<td>SHA-3</td>
<td>64 bits</td>
</tr>
<tr>
<td>Whirlpool</td>
<td>8 bits and 64 bits</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>32 bits</td>
</tr>
<tr>
<td>RC4-BHF</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

5). Preimage resistant, second preimage resistant and collision resistant. The other three requirements for a cryptographic hash function are the properties: preimage resistant (one-way property), second preimage resistant (weak collision resistant), and collision resistant (strong collision resistant), which have been introduced in Section 2.3.

The property preimage resistant requires that it is easy to generate a hash value given a message, but it is impossible to generate a message given a hash value. The property
second preimage resistant guarantees that it is impossible to find an alternative message with the same hash value as a given message. The property collision resistant guarantees that it is impossible to find any pair of the messages which can generate the same hash value. The designs of the compression process of RC4-BHF, the appending and dividing process, the output process, the truncation process, and the high randomness ensure that RC4-BHF meets these requirements.

All cryptanalytic attacks of hash functions seek to exploit one or more of these three properties to perform some attacks other than an exhaustive search (also called brute-force attack). The following are the three major categories of attacks which match the three requirements and the known attacks are listed in [88, 89, 90]. In cryptanalysis, if it is difficult to find any of the above attacks against a cryptographic hash function, we can claim that this hash function is resistant to these attacks [23].

- Preimage Attack: given a random y, it can use less than the required time for the exhaustive search to find M that \( H(M) = y \)
- Second Preimage Attack: given a message \( M_1 \), it can use less than the required time for the exhaustive search to find \( M_2 \) that \( H(M_1) = H(M_2) \)
- Collision Attack: it can use less than the required time for the exhaustive search to find \( M_1 \neq M_2 \), but \( H(M_1) = H(M_2) \)

For a cryptographic hash function, the compression function is the most important part as the problem of designing a secure hash function reduces to that of designing a
collision-resistant compression function [30]. As the terms are notated in Figure 2.7, the
cryptanalysis of hash functions focuses on the internal structure of the compress function
\( f \) and is based on the attempts to find efficient techniques to produce collisions for a
single execution of \( f \). The attack must also take another input IV into account. KSA and
PRGA have been studied for many years and there are no reported methods to find
successful collisions, therefore the internal structure of RC4-BHF makes it improbable to
produce collisions for a single execution of \( f \). Furthermore, the value of another input
len depends on the internal state and the offset number. It is extremely difficult to
produce a collision for a single execution of \( f \), and in the meantime that collision
matches the value len. To the best of our knowledge, the feasible collision attacks of hash
functions, which are the attacks effective against MD4-family [19, 20, 21, 22] are to work
on the specific internal structures inherited from the symmetric block ciphers, which \( f \)
consists of a series of round functions. These attacks, such as the differential
cryptanalysis, have relatively high possibility to find out the pattern of bit changes from
round to round to produce collisions [18]. However, these attacks are not applicable for
RC4-BHF as RC4-BHF is based on RC4 algorithm, which does not have a similar
structure to execute the round processing many times. [23] also confirmed our conclusion,
indicating that many analysis results of RC4 algorithm can be used to show the security
of an RC4 based hash function against known attacks and more importantly resistance
against attacks proposed by [20, 21, 22].
In RC4-BHF, the output function (Figure 4.4) provides an extra protection as this design prevents an attacker from going backward from the final hash value to the internal state $STATE_n$ to make sure that the attacker cannot find any information about the internal state. This structure also blocks the distinguishing attack and as a result, the hash value cannot be distinguished from a truly random sequence.

For a hash function with $c$-bit hash value, an exhaustive search takes $2^{c/2}$ complexity for a collision attack and $2^c$ complexity for both the preimage attack and second preimage attack [30]. In addition, Kelsey-Schneier [91] has shown a generic attack for second preimage on a classic hash function with complexity much less than $2^c$. Therefore, a wide pipe hash design has been suggested [92]. If the intermediate state size is very large compared to the final hash size, the security of the hash function can be assumed to be strong [92] even though there are some weaknesses in the compress function [23]. The Kelsey-Schneier second preimage attack is not applicable if the intermediate state size is larger than two times of the final hash size [23]. RC4-BHF is a wide pipe hash function and more importantly the intermediate state size of RC4-BHF is 2048 bits, which is very large compared to the final hash size, which is 128 bits or 256 bits. Thus, RC4-BHF is considered to be a strong hash function and the Kelsey-Schneier second-preimage attack is not applicable with RC4-BHF.

The collision attack against RC4-HASH [24] relies on finding fixed points when the index pointer $j$ is equal to “0” or “1”. Since RC4-BHF does not include the index pointer $j$
in the intermediate state, the collision attack presented by [24] does not work for RC4-BHF. Because offset is inputted, a strong output function is introduced, the index pointers i and j are not included in the intermediate state, and the issue of possible running “0” loop of PRGA is resolved, the collisions attack against the hash function presented by [25, 26] fail to work. In addition, the attacks which seek to attack RC4 algorithm, such as [29, 44, 58, 65, 66, 67, 68] have been reviewed, and we confirmed that none of them are applicable with RC4-BHF. We believe that RC4-BHF meets the requirements of preimage resistant, second preimage resistant and collision resistant and it is secure.

6.2 The Analysis of SDTP

This section analyzes the proposed secure data transmission protocol SDTP, which includes the analysis for security, feasibility, and performance.

Security Analysis

The proposed secure data transmission protocol SDTP is based on RC4 algorithm with improvements, therefore the security analysis can be made in the view of the security analysis of the RC4 algorithm, the improvements, and how to use them. The improvements include using the offset number and using the forward and backward property of RC4 states.
We have discussed the security strength of RC4 in section 3.2 and concluded that the RC4 algorithm is secure when use correctly. We noticed that there is a weakness with RC4, called the insufficient key schedule, which is the first bytes of the output revealing information about the key. This issue has been avoided in SDTP by introducing the offset number. The idea of introducing the offset number in SDTP is to discard the initial portion of the output stream to overcome the weakness of the insufficient key schedule. For the same reason, we also introduced an offset number into RC4-BHF to strength RC4-BHF. To use an offset number just changed the way how to use RC4 algorithm but RC4 algorithm remains the same. It obviously would not reduce the security strength and actually the security strength can be enhanced by avoiding the issue of the insufficient key schedule. In the same manner, the forward and backward property of RC4 states is an intrinsic property of RC4 algorithm. SDTP only uses this property and have no change to the RC4 algorithm. For this reason, it also would not reduce the security strength of the RC4 algorithm. We believe that SDTP uses RC4 in a novel way which makes effective use of the strengths of RC4 and reduces its weaknesses.

For the encrypted data packets which are transferred across the open channel network, each packet includes a position bit, a sequence counter, a data segment, and a hash value checksum. The position bit and sequence counter are unencrypted, so the recipients can use these information to handle the packets upon receiving them. However the hash value checksum is generated through the position bit, sequence counter, and data segment,
which ensures that there is no alteration and error during the data transmission and the encrypted hash value ensures that they are sent by the claimed sender. The data segment and hash value are encrypted to make sure an opponent cannot read the original data, and the data and hash value checksum are sent from the claimed sender.

In the following, we confirm that SDTP satisfies the necessary requirements of the network security which are listed in Section 2.1. Non-repudiation protects against either the sender or receiver from denying a transmitted message later, which is very costly and not necessary for majority of the applications for ultra-low power devices, therefore SDTP does not provide non-repudiation protection.

- **Access Control** protects against unauthorized access and SDTP offers a dynamic access control.

- **Data Confidentiality** keeps the contents of information hidden from all but authorized ones. SDTP achieves data confidentiality through the data encryption by the RC4 algorithm with improvements.

- **Data Integrity** protects from unauthorized data alteration and **Authentication** assures sender’s identity. SDTP achieves data integrity through the hash value checksum, and the authentication through the encrypted hash value checksum.

SDTP can also offer semantic security and can defend against known network security attacks, such as spoofing attacks, modified packet attacks and replay attacks.
• Semantic security makes sure the knowledge of the ciphertext and the length of message does not reveal any additional information on the message that can be feasibly extracted. SDTP offers semantic security through always using at different portion of the key stream to encrypt different messages, even the two messages have the same content, but are just repeated at a different time.

• A spoofing attack is an attacker tries to masquerade as another to falsify data. A modified packet attack is to modify a portion or entire packet to affect the result. Since SDTP uses encrypted hash value checksum, both the spoofing attacks and the modified attacks are not applicable.

• A replay attack is to capture packets sent to a receiver with the intent to later to replay the payloads or exact packets in order to affect the result. Since SDTP uses the monotonically increased sequence counter for each data packet and have encrypted hash value checksum to guarantee the sequence counter without alternation, the replay attacks do not work.

**Feasibility Analysis**

SDTP for secure data unicast and secure data broadcast, including the generation, distribution, and self-authentication of the one-way base key chain and one-way offset number chain through RC4-BHF, has been implemented and simulated in C programming language. The source code of the sender function and receiver function of SDTP has been attached in Appendix B. When simulating the secure data broadcast, one
sender and four receivers are included as participants. We also analyzed SDTP for secure data multicast. The implementation, simulation, and analysis are summarized below.

- Since RC4 algorithm is state based, the same RC4 state has to be used for both the encryption and decryption on the same digit. If a recipient gets out of the synchronization by even a single byte on the state, the decryption will fail. Since each data packet includes a sequence counter and each participant of the data transmission also maintains a sequence counter, SDTP can confirm that each participant of the data transmission uses the same state on the same digit for both the encryption and decryption.

- The process of the generation, distribution, and self-authentication of both the one-way base key chain and one-way offset number chain work as expected.

- In SDTP, the RC4 state maintained by each participant can move forward and backward smoothly and correctly to handle the delayed or lost packets, based on the forward and backward property of RC4 states.

- In SDTP, the inclusion of the offset number with RC4 algorithm works smoothly.

- SDTP does not need to initialize the RC4 state until a request to change the base key.

- The position bit and sequence counter keep unencrypted so the recipients can direct handle the received packets, but the encrypted hash value checksum ensures that there is no alternation or transmission error on the position bit and sequence counter.
• The padding and dividing process conducted by the sender and the restoration process conducted by the receiver work smoothly and as expected.

• It is relatively easy to use RC4-BHF to compute a hash value for any given input, and the verification process can detect either someone is trying to impersonate the sender, the message itself has been altered, or an error occurred during transmission.

• SDTP can achieve the dynamic access control when the participants are mobile. It can handle the cases of frequent member change, such as a new member joins, an existing member leaves, and a member moves in the network.

• Each participant only keeps one RC4 state and one sequence counter in the memory.

**Performance Analysis**

In respect to performance, SDTP runs very efficiently benefiting from RC4 and the forward and backward property of RC4 states. As indicated in Chapter 3, RC4 is remarkably simple and fast and the overhead is low. For example, there is no need to store substitution-boxes in the memory and there is no need to do table lookup. The involved operations for SDTP, such as exclusive-or operation, byte swap operation, byte comparison operation, modulo operation and add operation only require byte manipulations. Furthermore, SDTP uses RC4 more efficiently. By using the forward and backward property of RC4 states, SDTP does not require frequent key re-initialization. This translates directly to simplification and time and power saving, as well as improved performance over the ordinary RC4 based schemes. For example, if the length of data is
256 bytes, since KSA is not required for every new message, the proposed encryption or decryption process requires only about half the operations and half the process time than the ordinary RC4 based scheme.

Table 6.5 and Table 6.6 compare the execution times among AES-based scheme, WEP (ordinary RC4 based scheme), RC5-based scheme, Skipjack-based scheme and SDTP. The comparisons are based on an 8-bit processor. In Table 6.5, \( t_1 \) is the time required for a logic operation or a simple arithmetic operation, \( t_2 \) is the time required for one time memory access. For simplicity, we assume \( t = t_2 = 2t_1 \) in Table 6.6. The result of the comparisons confirmed that SDTP runs much faster than others.

Table 6.5: The Comparisons of the Execution Times on an 8-bit Machine (1)

(\( t_1 \) is the required time for a logic operation or a simple arithmetic operation, and \( t_2 \) is the required time for one time memory access)

<table>
<thead>
<tr>
<th>Data Length</th>
<th>64 bytes</th>
<th>128 bytes</th>
<th>256 bytes</th>
<th>512 bytes</th>
<th>1024 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-based</td>
<td>5824(t_1) + 6272(t_2)</td>
<td>11648(t_1) + 12544(t_2)</td>
<td>23296(t_1) + 25088(t_2)</td>
<td>46592(t_1) + 50176(t_2)</td>
<td>93184(t_1) + 100352(t_2)</td>
</tr>
<tr>
<td>WEP</td>
<td>1024(t_1) + 2816(t_2)</td>
<td>1280(t_1) + 3328(t_2)</td>
<td>1792(t_1) + 4352(t_2)</td>
<td>2816(t_1) + 6400(t_2)</td>
<td>4864(t_1) + 10496(t_2)</td>
</tr>
<tr>
<td>RC5-based</td>
<td>14784(t_1) + 832(t_2)</td>
<td>29568(t_1) + 1664(t_2)</td>
<td>59136(t_1) + 3328(t_2)</td>
<td>118272(t_1) + 6656(t_2)</td>
<td>236544(t_1) + 13312(t_2)</td>
</tr>
<tr>
<td>Skipjack-based</td>
<td>1280(t_1) + 1664(t_2)</td>
<td>2560(t_1) + 3328(t_2)</td>
<td>5120(t_1) + 6656(t_2)</td>
<td>10240(t_1) + 13312(t_2)</td>
<td>20480(t_1) + 26624(t_2)</td>
</tr>
<tr>
<td>SDTP</td>
<td>256(t_1) + 512(t_2)</td>
<td>512(t_1) + 1024(t_2)</td>
<td>1024(t_1) + 2048(t_2)</td>
<td>2048(t_1) + 4096(t_2)</td>
<td>4096(t_1) + 8192(t_2)</td>
</tr>
</tbody>
</table>
Table 6.6: The Comparisons of the Execution Times on an 8-bit Machine (2)

(assume $t = t_2 = 2t_1$)

<table>
<thead>
<tr>
<th>Data Length</th>
<th>64 bytes</th>
<th>128 bytes</th>
<th>256 bytes</th>
<th>512 bytes</th>
<th>1024 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-based</td>
<td>9184t</td>
<td>18368t</td>
<td>36736t</td>
<td>73472t</td>
<td>146944t</td>
</tr>
<tr>
<td>WEP</td>
<td>3328t</td>
<td>3968t</td>
<td>5248t</td>
<td>7808t</td>
<td>12928t</td>
</tr>
<tr>
<td>RC5-based</td>
<td>8224t</td>
<td>16448t</td>
<td>32896t</td>
<td>65792t</td>
<td>131584t</td>
</tr>
<tr>
<td>Skipjack-based</td>
<td>2304t</td>
<td>4608t</td>
<td>9216t</td>
<td>18432t</td>
<td>36864t</td>
</tr>
<tr>
<td>SDTP</td>
<td>640t</td>
<td>1280t</td>
<td>2560t</td>
<td>5120t</td>
<td>10240t</td>
</tr>
</tbody>
</table>

6.3 Support for Ultra-Low Power Devices

The features of the wireless sensor and other ultra-low power device networks, and their security concerns have been introduced in Chapter 1.

One of the challenges to deploy security on the ultra-low power devices is how to offer necessary security protection to the resource constrained devices. It is impractical to apply conventional security mechanisms which are designed for powerful digital systems to these devices. For example, sensor nodes typically equipped with an 8-bit low-end microprocessor to perform bit or byte oriented operations, and embedded with limited ROM, RAM and flash memory for operations and data storages. The security mechanism for wireless sensor networks has to be able to run on an 8-bit low-end microprocessor and the usage of processor and memory should be as low as possible. In addition, the extra
overhead added to the data packets should be as low as possible to save the transmission cost for the radio transceiver to send and receive data.

The efficiency and simplicity of the proposed cryptographic hash function RC4-BHF and the proposed data transmission protocol SDTP make them more suitable than others for wireless sensor and other ultra-low power device networks. For example, RC4-BHF and SDTP are designed to be exceptionally fast and resources saving on an 8-bit machine. RC4-BHF is based on KSA, PRGA, KSA\(^*\), and PRGA\(^*\) and SDTP is based on KSA, PRGA, and RC4-BHF, and these algorithms or functions are remarkably simple and very efficient. The operations of these algorithms and other necessary operations are bit or byte oriented, which includes exclusive-or operation, byte swap operation, byte comparison operation, modulo operation, and add operation. As far as we know, there are few cryptographic mechanisms which are designed for an 8-bit machine. The simplicity of the RC4-BHF and SDTP lead directly to memory space saving, which is used to store the codes, variables, and transformation lookup tables on the device. For example there are many ciphers requiring complex substitution-boxes, which lead to the cryptographic mechanisms and security applications which use these ciphers requiring more processing, but RC4-BHF and SDTP are more efficient. The memory consumption during the process and the memory preservation for variables are also light weight. SDTP only requires adding two-byte overhead by default to each data packet and it saves the transmission cost for the radio transceiver to send and receive data.
Figure 6.1: The Setup of the Experiment

The proposed cryptographic hash function RC4-BHF has been implemented with real life sensor devices\(^1\), and in the next step the proposed secure data transmission protocol SDTP will also be implemented with real sensor nodes. Figure 6.1 shows the experiment environment, consisted of Node A and Node B which are the two participants of the data transmission. The detail of the Node A and Node B are listed in the following.

The Node A is a complete set of a sensor node which consists of sensors, RF transceiver, XBee shield and microcontroller. Figure 6.2 shows the setup of the Node A.

- **Sensors**: TMP36GT9Z temperature sensor [93] and PIR motion sensor [94] have been used in the experiment.

- **Arduino UNO ATmega328P**: This is a microcontroller with 5V operating voltage and 14 digital I/O pins. It has a clock speed of 16MHz, 8-bit microprocessor, 1 KB

\(^1\) The implementation was done with the help of Dr. Hua Li and his research team, and the implementation process is presented in [35]. I thank Dr. Hua Li and his research team to provide the sensor devices and the assistance on the sensor implementation.
EEPROM, 2 KB SRAM. 0.5 KB is pre-used by bootloader out of the total 32KB of the flash memory [95].

- Arduino RF Transceiver: 2.4GHz XBee module which has 100m of range [96].
- Arduino XBee Shield: This shield helps Arduino to communicate via RF transceiver. It mounts directly on Arduino and holds RF transceiver. This acts like a connector between RF transceiver and Arduino Uno.

![Setup of the Node A](image1)

**Figure 6.2: The Setup of the Node A**

The Node B includes an Arduino RF Transceiver for receiving the data, and it is connected with computer via USB cable. The only reason to use a computer in Node B is because it needs a way to output the experiment result. Figure 6.3 shows the setup of the Node B.
The experiment was implemented with one sender node which is Node A, and one receiver node which is Node B. RC4-BHF with the offset number was pre-loaded into Arduino UNO ATmega328P on the sender node, and saved on the computer of the receiver node. In the sender node, the temperature sensor and motion sensor were used. After successfully configuring, the communications between parties were started. Once the sender node received a sensor reading, it calculated the hash value checksum through inputting this reading, and sent the sensor reading along with the calculated hash value checksum to the receiver node. The computer connected to the Arduino RF transceiver on the receiver node received them. The receiver node applied RC4-BHF to generate a hash value through inputting the received sensor reading, and compared this hash value with
the hash value attached to the sensor reading. The two hash values were the same, and the receiver node successfully authenticated the hash value. Therefore the experiment has confirmed that the proposed cryptographic hash function RC4-BHF works as expected on real life sensor devices.

6.4 Summary

This chapter analyzed the proposed cryptographic hash function RC4-BHF and the proposed secure data transmission scheme SDTP, and discussed their support for wireless sensor and other ultra-low power device networks. According to the analysis, we believe RC4-BHF is an ideal hash function and SDTP offers necessary protection of the network security, all for the wireless ultra-low power device networks.
Chapter 7

Conclusions and Future Work

This chapter summarizes and concludes the research work presented in this dissertation and recommends future research work.

7.1 Conclusions

Current developments show that in the near future the wide availability of the relatively low-cost devices along with advances in ultra-low power technologies will enable wireless sensor and other ultra-low power device networks to reach every corner of our daily lives and become commonly deployed in various fields, such as industry, agriculture, national defense, and scientific research. An ultra-low power device is a tiny object which is highly constrained in terms of resources. For example, a regular sensor node only consists of the limited power supply, 8-bit low-end microprocessor with small memory, low-power radio transceiver, and necessary sensors.

Security is a critical factor for wireless sensor and other ultra-low power device networks due to the impact on privacy, trust and control. One of the major bottlenecks to
widely deploy them is the security applications which employ computationally intensive
cryptographic operations. Since ultra-low power devices are highly constrained in terms
of resources, how to find cryptographic mechanisms that work for them is a challenge.

The objective of this research work is to develop an affordable and efficient security
solution for wireless ultra-low power device networks to meet their necessary security
goals, such as data confidentiality, data integrity, authentication, and access control. It is
worth noting that the proposed cryptographic hash functions RC4-BHF and its
applications, and the proposed secure data transmission protocol SDTP for secure data
unicast, secure data broadcast and secure data multicast are all based on the RC4
algorithm. This directly simplifies the implementation, saves the storage spaces, and
improves the efficiency. Benefits from the speed and simplicity of the RC4 algorithm and
its byte-orientated operations, the proposed research work is very simple, fast, and has
low overhead, and is ideal for wireless sensor and other ultra-low power device networks.

The following summarizes and concludes the proposed research work.

This dissertation has proposed a new cryptographic hash function, called RC4-BHF,
which is designed to be both fast and secure, and is applicable to ultra-low power devices.
In conclusion, RC4-BHF has at least the following three advantages. Firstly, the majority
of the current hash functions are designed for 32-bit or 64-bit processors, and are not
applicable or cannot be implemented efficiently on ultra-low power devices which are
normally equipped with 8-bit processors. RC4-BHF is designed for 8-bit processors and it
suits ultra-low power devices very well. Secondly, after the implementation and comparison, we confirm that RC4-BHF is faster than all the well-known hash functions on 32-bit processors, and should be a lot faster on 8-bit processors. Thirdly, the security of RC4-BHF is based on the security of RC4 algorithm, but eliminates the reported issues of RC4 algorithm, such as the insufficient key schedule issue. The internal structure of RC4-BHF is different from the broken hash functions so that people cannot use the existing attack strategies to break it. If applying RC4-BHF to a large set of inputs, it produces outputs that are evenly distributed and random. From the analysis, we believe RC4-BHF is secure. RC4-BHF plays a fundamental role in achieving the data integrity, authentication, access control, and random number generation for the proposed secure data transmission protocol SDTP.

How to use RC4-BHF to construct other cryptographic mechanisms for ultra-low power devices have also been discussed. This dissertation has introduced how to use RC4-BHF to construct a one-way RC4 base key chain and a one-way offset number chain, which can be used to generate, distribute, and self-authenticate the base keys and offset numbers for the proposed secure data transmission protocol. As one-way hash chains are very efficient to generate and to verify, it is ideal for ultra-low power devices. We can also use RC4-BHF to construct many other cryptographic mechanisms. Benefits from the high efficiency and simplicity of RC4-BHF, the cryptographic mechanisms or security applications based on RC4-BHF can also be very efficient and simple. A couple examples
have been briefly introduced which are to use RC4-BHF to construct a message authentication function and a pseudorandom number generator. The pseudorandom number generator can be used to generate pseudorandom numbers in the proposed secure data transmission protocol SDTP.

This dissertation has also presented a secure data transmission protocol, called SDTP, which is an ideal secure data transmission solution for wireless sensor and other ultra-low power device networks to achieve secure data unicast, secure data broadcast, and secure data multicast. SDTP achieves data confidentiality, data integrity, authentication, and access control with simple operations and low overhead, which is based on the simple structure of RC4 and RC4-BHF, and the forward and backward property of RC4 states. SDTP offers semantic security and can defend against known network security attacks, such as spoofing attacks, modified packet attacks and replay attacks. By using the offset, SDTP avoids the insufficient key schedule issue which is a weakness of RC4 algorithm. By using the forward and backward property of RC4 states, SDTP does not require strong synchronization and can handle the delayed or lost data packets. SDTP does not require frequent key renewal and this leads directly to further simplification and power saving.

It is noteworthy that SDTP includes an efficient access control to support the frequent key changes in a dynamic membership environment, such as frequent member joining or member leaving. For example, each time when a new member joins, an existing member leaves, or in some cases a member moves from one location to another
in the network, all members may need to have a new session key. In addition, when a member joins, for most applications it does not allow the new member to be able to retrieve the previous session keys from the new session key; and when a member leaves, the applications do not allow the member who left to retrieve the new session key or future session keys from the previous session keys. The proposed dynamic access control can handle the above situations. Moreover, the session keys are self-authenticated, meaning it is very easy and efficient to authenticate subsequent keys by an authenticated key, which is normally the current session key or a previous session key.

7.2 Recommendations for Future Research

This dissertation presented a security solution for wireless sensor and other ultra-low power device networks to meet their network security requirements, which is to provide data confidentiality, data integrity, authentication, and access control to achieve secure data unicast, secure data broadcast, and secure data multicast. This section provides an overview of possible areas in which further work could be pursued.

**Fully deploying actual wireless sensor networks:** The proposed hash function RC4-BHF has been implemented in real life sensor node. In the next step, the proposed secure data transmission protocol SDTP can be fully deployed in an actual wireless sensor network environment.
Other attempts to enhance RC4: The proposed hash function RC4-BHF and the proposed secure data transmission protocol SDTP are all based on the RC4 stream cipher with improvements. Recently, there are some attempts to try to modify the RC4 algorithm to enhance it, such as RC4A [97], VMPC [98], and RC4+ [99]. These attempts are only proposals and have not been fully studied. We can study these proposals and consider adopting some benefits into the proposed research work.

Other applications: The proposed research work is ideal for wireless sensor and other ultra-low power device networks, but is also applicable to other applications. We can consider how to apply the proposed research work in other applications.

Trust management: In the network of more than two participants, if the participants can trust each other, only a few of the secret information such as session keys need to be shared and used by all participants. Otherwise, each pair needs unique secret information. How to establish the trust among all participants is a concern, and how to solve the problem if no trust can be established could be a new research direction.

Denial-of-service attacks: A denial-of-service attack or distributed denial-of-service attack is an attempt to make a machine or network resource unavailable to provide services. How to prevent these attacks in wireless ultra-low power device networks is a whole topic unto itself and could be another research direction.

Nonrepudiation: Nonrepudiation protects against either sender or receiver from denying a transmitted message. When a message is sent, the receiver can prove that the
alleged sender in fact sent the message, and when a message is received, the sender can prove that the alleged receiver in fact received the message. This is not required for most of the security applications, especially for ultra-low power device applications. However a few applications may still need a mechanism to achieve this, and it could also be a new research direction.


[34] Q. Yu, C. N. Zhang. "RC4-BHF: A New and Fast Cryptographic Hash Function". This paper is in preparation for journal publication.


[89] “Cryptographic Hash Function”,

[90] “Hash Function Security Summary”,


Appendix A

Proof of the Reversibility of RC4 States and a Programming Simulation

This section proves that any RC4 state is reversible, and conducts a programming simulation to confirm the forward and backward property of RC4 states.

Let \((S_0, i_0, j_0)\) be the RC4 state initialized by KSA, where \(i_0 = 0\) and \(j_0 = 0\) is the initial index values of \(i\) and \(j\), and \(S_0\) is the initial state vector for PGRA in RC4. If \(PRGA^r(S_0, i_0, j_0) = (S_n, i_n, j_n)\), it has the following property:

**Theorem 1:** For any \(r, n \geq r \geq 0\), we have:

\[
PRGA^r(S_n, i_n, j_n) = (S_{n-r}, i_{n-r}, j_{n-r})
\]

**Proof.** Our proof is conducted by induction on integer \(r\)

**Base Case:** \(r = 0\). It is true: \(IPRA^0(S_n, i_n, j_n) = (S_n, i_n, j_n)\)

**Inductive Step:**

Assume that for any \(r \leq k\) we have: \(PRGA^k(S_n, i_n, j_n) = (S_{n-k}, i_{n-k}, j_{n-k})\)

Consider case of \(r = k + 1\)

\[
PRGA^{k+1}(S_n, i_n, j_n) = PRGA(IPRA^{k}(S_n, i_n, j_n)) = PRGA(S_{n-k}, i_{n-k}, j_{n-k})
\]

\[131\]
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 \( i_{n-k} = i_{n-k-1} + 1 \)

\[
j_{n-k} = j_{n-k-1} + S_{n-k-1}[i_{n-k-1} + 1]
\]

\[
S_{n-k}[m] = S_{n-k-1}[m] \]

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 \( S^* = S_{n-k-1} \)

\[
j^* = j_{n-k} - S^*[i_{n-k}] = j_{n-k} - S_{n-k-1}[i_{n-k-1} + 1] - S_{n-k-1}[i_{n-k-1} + 1] = j_{n-k-1}
\]

\[
i^* = i_{n-k} - 1 = i_{n-k-1}
\]

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 \( IPRGA'(S_{n}, i_{n}, j_{n}) = (S_{n-r}, i_{n-r}, j_{n-r}) \)

In order to better understand and more intuitive to visually confirm the forward and backward property of RC4 states, a programming simulation was conducted. Figure A.1 to Figure A.5 in the next five pages lists the simulation process.
(1) Figure A.1 simulates that the entries of $S$ are set equal to the values from 0 through 255 in ascending order, at the beginning of applying KSA algorithm.

(2) Figure A.2 simulates the process of applying KSA algorithm. Figure A.2 shows an initialized $S$ vector, which is generated by the KSA algorithm. For simplicity, a base key is not entered, but is randomly generated in the calculation.

(3) Figure A.3 simulates the process of generating a random RC4 state. In order to confirm that any RC4 state is reversible, certain rounds of PRGA is applied to generate an RC4 state as a random RC4 state. In the simulation, 13 times of PRGA is applied to generate an RC4 state as a random RC4 state to start.

(4) Figure A.4 simulates the process of applying PRGA algorithm. In this simulation, 7 times of PRGA is applied, starting from the random RC4 state illustrated in Figure A.3, and generated an RC4 state and is illustrated in Figure A.4.

(5) Figure A.5 simulates the process of applying IPGRA algorithm. In the simulation, 7 times of IPGRA is applied, starting from the RC4 state illustrated in Figure A.4, and generated an RC4 state and is illustrated in Figure A.5.

From the simulation we can find that the RC4 state illustrated in Figure A.3 is the same as the one illustrated in Figure A.5, as expected. It confirmed that any RC4 state can be shifted forward by PRGA algorithm, or can be shifted backward by IPGRA algorithm. IPGRA algorithm can be used to recover any RC4 state to any of its previous RC4 states.
Figure A.1: Simulate that the Entries of State Vector S are Set Equal to the Values from 0 through 255 in Ascending Order
Figure A.2: Simulate the Process of Applying KSA Algorithm
Figure A.3: Simulate the Process of Generating a Random RC4 State
Figure A.4: Simulate the Process of Applying PRGA Algorithm
Figure A.5: Simulate the Process of Applying IPRGA Algorithm
Appendix B

Source Code

This appendix provides selected source code which is written in C programming language for the implementation of the proposed cryptographic hash function RC4-BHF and the proposed secure data transmission protocol SDTP.

// <copyright>Copyright (c) 2014 All Right Reserved</copyright>
// <author>Qian Yu</author>
// <affiliation>Department of Computer Science, University of Regina</affiliation>
// <email>yu209@cs.uregina.ca</email>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <string.h>

#define bool unsigned char
#define true 1
#define false 0
#define bLength 64

//This function swaps the values in two different points in an array.
void swap(unsigned char state[], unsigned int i, unsigned int j)
{
    unsigned int temp = state[i];
    state[i] = state[j];
    state[j] = temp;
}
// This function compares two hash to determine they are equal or not
bool comparehash(unsigned char a[], unsigned char b[])
{
    int i;
    for(i=0; i<16; i++)
    {
        if(a[i] != b[i]) return false;
    }
    return true;
}

// This function calculates the lengths based on the input m[] and offset number.
unsigned int length(unsigned char m[], unsigned int offset)
{
    unsigned int length = 0, idx;
    for(idx=0; idx<64; idx++)
    {
        length = length + (int)m[idx];
    }
    length = length % 256;
    if (length == 0)
    {
        length = offset % 256;
    }
    return length;
}

// Key Scheduling Algorithm (KSA)
void ksa(unsigned char state[], unsigned char key[], unsigned int keylen)
{
    unsigned int i, j = 0;
    for (i=0; i<256; i++)
    {
        state[i] = i;
    }
    for (i=0; i<256; i++)
    {
        j = (j + state[i] + key[i % keylen]) % 256;
        swap(state, i, j);
    }
}
//Key Scheduling Algorithm Star (KSA*)
void ksa_star(unsigned char state[], unsigned char m[])
{
    unsigned int i, j = 0;
    for (i=0; i<256; i++)
    {
        j = (j + state[i] + m[i % 64]) % 256;
        swap(state, i, j);
    }
}

//Pseudo-Random Generator Algorithm (PRGA)
void prga(unsigned char state[], unsigned char output[], unsigned int msglen, unsigned int *i, unsigned int *j)
{
    unsigned int idx;
    for (idx=0; idx<msglen; idx++)
    {
        *i = (*i + 1) % 256;
        *j = (*j + state[*i]) % 256;
        swap(state, *i, *j);
        output[idx] = state[(state[*i] + state[*j]) % 256];
    }
}

//Pseudo-Random Generator Algorithm (PRGA), but without output
void prga_nooutput(unsigned char state[], unsigned int msglen, unsigned int *i, unsigned int *j)
{
    unsigned int idx;
    for (idx=0; idx < msglen; idx++)
    {
        *i = (*i + 1) % 256;
        *j = (*j + state[*i]) % 256;
        swap(state, *i, *j);
    }
}
/Pseudo-Random Generator Algorithm Star (PRGA*)
void prga_star(unsigned char state[], unsigned int length)
{
    unsigned int i, j=0;
    for (i=0; i < length; i++)
    {
        j = (j + state[i]) % 256;
        swap(state, i, j);
    }
}

//Inverse Pseudo-Random Generator Algorithm (IPRGA)
void iprga(unsigned char state[], unsigned int msglen, unsigned int *i, unsigned int *j)
{
    unsigned int idx;
    for (idx=0; idx<msglen; idx++)
    {
        swap(state, *i, *j);
        *j = (*j - state[*i]) % 256;
        *i = (*i - 1) % 256;
    }
}

//RC4-BHF hash function
void hash (unsigned char plaintext[], unsigned char hashvalue[], unsigned int offset)
{
    unsigned char state[256], state2[256], op_stream[256];
    unsigned int msglen = 0, idx, divnum = 0, zerolen, x, y;
    unsigned char **m;

    //msglen: msglen is an integer number to indicate the length of the message.
    msglen = strlen((char*)plaintext);

    //divnum: divnum is an integer number to indicate how many message blocks the
    original input message being divided into.
    divnum = (msglen + 3) / 64;
    if (((msglen + 3) % 64 != 0)
    {
        divnum = divnum + 1;
    }
//zerolen: zerolen is an integer number to indicate how many zeros to be padded
zerolen = 64 * divnum - msglen - 3;

//The following code is for the padding and dividing process of the RC4-BHF
m = (unsigned char **)malloc(divnum*sizeof(unsigned char*));
for (idx=0; idx<divnum; idx++)
{
    m[idx]=(unsigned char *)malloc(64*sizeof(unsigned char));
}
if (((64*divnum)-msglen) == 66)
{
    for(x=0; x<(divnum-2); x++)
    {
        for(y=0; y<64; y++)
        {
            m[x][y] = plaintext[64*x+y];
        }
    }
    for(y=0; y<62; y++)
    {
        m[divnum-2][y] = plaintext[64*(divnum-2)+y];
    }
    m[divnum-2][62]= 0x80;
    m[divnum-2][63]= 0x00;
    for(y=0; y<62; y++)
    {
        m[divnum-1][y] = 0x00;
    }
    m[divnum-1][62]=msglen>>8;
    m[divnum-1][63]=msglen&~(0xFF<<8);
}
else if (((64*divnum)-msglen) == 65)
{
    for(x=0; x<(divnum-2); x++)
    {
        for(y=0; y<64; y++)
        {
            m[x][y] = plaintext[64*x+y];
        }
    }
}
for(y=0; y<63; y++)
{
    m[divnum-2][y] = plaintext[64*(divnum-2)+y];
}
m[divnum-2][63]= 0x80;
for(y=0; y<62; y++)
{
    m[divnum-1][y] = 0x00;
}
m[divnum-1][63]=msglen>>8;
m[divnum-1][64]=msglen&~(0xFF<<8);
}
else
{
    for(x=0; x<(divnum-1); x++)
    {
        for(y=0; y<64; y++)
        {
            m[x][y] = plaintext[64*x+y];
        }
    }
    for(y=0; y<(61-zerolen); y++)
    {
        m[divnum-1][y] = plaintext[64*(divnum-1)+y];
    }
m[divnum-1][61-zerolen] = 0x80;
    for(y=(62-zerolen); y<62; y++)
    {
        m[divnum-1][y] = 0x00;
    }
m[divnum-1][62]=msglen>>8;
m[divnum-1][63]=msglen&~(0xFF<<8);
}

    //The following code is for the compression process of the RC4-BHF
ksa(state, m[0], 64);
prga_star(state, offset);
prga_star(state, length(m[0], offset));
if (divnum > 1)
{
    for (idx=1; idx < divnum; idx++)
    {

{ 
    ksa_star(state, m[idx]);
    prga_star(state, length(m[idx], offset));
}

//The following code is for the truncation process of the RC4-BHF
ksa(state2, state, 256);
x=0;
y=0;
prga_nooutput(state2, 256, &x, &y);
prga(state2, op_stream, 256, &x, &y);
for(idx=0; idx<256; idx++)
{
    op_stream[idx] = op_stream[idx] ^ state[idx];
}

//The following is for the 256-bit output hash value
for (idx=0; idx<32; idx++)
{
    hashvalue[idx] = 0;
}
for (idx=0; idx<256; idx++)
{
    int bucket = idx/8;
    int offset = idx%8;
    unsigned char flag = (op_stream[idx]&0x01)<<(7-offset);
    hashvalue[bucket] = hashvalue[bucket] + flag;
}

//The following is for the 128-bit output hash value (odd number case)
for (idx=0; idx<16; idx++)
{
    hashvalue[idx] = 0;
}
for (idx=0; idx<256; idx=idx+2)
{
    int bucket = idx/16;
    int offset = (idx%16)/2;
    unsigned char flag = (op_stream[idx]&0x01)<<(7-offset);
hashvalue[bucket] = hashvalue[bucket] + flag;
}

//The following is for the 128-bit output hash value (even number case)
for (idx=0; idx<16; idx++)
{
    hashvalue[idx] = 0;
}
for (idx=1; idx<256; idx=idx+2)
{
    int bucket = idx/16;
    int offset = ((idx-1)%16)/2;
    unsigned char flag = (op_stream[idx]&0x01)<<(7-offset);
    hashvalue[bucket] = hashvalue[bucket] + flag;
}

//This function is the sender function of the SDTP
void sender(unsigned char *input, unsigned char *dblock)
{
    unsigned int msglen = 0, divnum = 0, offset = 100, keylen, tmp, idx, x, y , i, j;
    unsigned char hashinput[48];
    unsigned char hashvalue[16];
    unsigned char state[256];
    unsigned char key[]={"ThisIsTheKey"};
    unsigned char keystream[62];
    msglen = strlen(input);
    keylen = sizeof(key) - 1;
    ksa(state,key,keylen);
    divnum = msglen / 46;
    if (msglen % 46 != 0)
    {
        divnum = divnum + 1;
    }
    prga_nooutput(state, offset, &i, &j);
    for(x=0; x<divnum; x++)
    {
        tmp=x;
        if(x==(divnum-1))tmp+=0x8000;
        dblock[bLength*x]=((tmp&0xFF00)>>8);
        dblock[bLength*x+1]=(tmp&0xFF);
for(y=0; y<46; y++)
{
    if((46*x+y)<msglen)
    {
        dblock[bLength*x+2+y] = *input++;
    }
    else if((46*x+y)==msglen)
    {
        dblock[bLength*x+2+y]=0x8;
    }
    else
    {
        dblock[bLength*x+2+y]=0x7F;
    }
}
for(idx=0; idx<48; idx++)
{
    hashinput[idx] = dblock[bLength*x+idx];
}
hash(hashinput, hashvalue, offset);
for(idx=0; idx<16; idx++)
{
    dblock[bLength*x+48+idx]=hashvalue[idx];
}
prga(state, keystream, 62, &i, &j);
for(idx=0;idx<62;idx++)
{
    dblock[bLength*x+2+idx] = keystream[idx] ^ dblock[bLength*x+2+idx];
}

//This function is the receiver function of the SDTP
void receiver(unsigned char *dblock, unsigned char *output)
{
    unsigned int msglen = 0, divnum = 0, offset = 100, keylen, idx, x, y , i, j;
    unsigned char hashinput[48];
    unsigned char recvhash[16];
    unsigned char calhash[16];
    unsigned char state[256];
    unsigned char key[]="ThisIsTheKey";
unsigned char keystream[62];
keylen = sizeof(key) - 1;
space
tate, key, keylen);  
ksa(state, key, keylen);
    prga_nooutput(state, offset, &i, &j);
    for(x=0; ; x++)
    {
        prga(state, keystream, 62, &i, &j);
        for(idx=0; idx<62; idx++)
        {
            dblock[bLength*x+2+idx] = keystream[idx] ^ dblock[bLength*x+2+idx];
        }
        for(idx=0; idx<16; idx++)
        {
            recvhash[idx]=dblock[bLength*x+idx+48];
        }
        for(idx=0; idx<48; idx++)
        {
            hashinput[idx] = dblock[bLength*x+idx];
        }
        hash(hashinput, calhash, offset);
        if (comparehash(calhash,recvhash))
        {
            for(y=0; y<46; y++)
            {
                if(dblock[x*bLength+2+y]! =0x8)*output++=dblock[x*bLength+2+y];
                else break;
            }
        }
        else
        {
            printf("two hash not equal!");
        }
        msglen+=64;
        if(dblock[bLength*x]&0x80)break;
    }
    *output='\0';
}

//To simulate RC4-BHF, to generate a 128-bit hash value
void main()
{

unsigned char plaintext[]={"this is the input message"};
unsigned int offset = 100, idx;
unsigned char hashvalue[16];
hash(plaintext, hashvalue, offset);
for (idx=0; idx<16; idx++)
{
    printf("%02X ", hashvalue[idx]);
}
printf("\n");

//To simulate RC4-BHF, to generate a 256-bit hash value
void main()
{
    unsigned char plaintext[]={"this is the input message"};
    unsigned int offset = 100, idx;
    unsigned char hashvalue[32];
    hash(plaintext, hashvalue, offset);
    for (idx=0; idx<32; idx++)
    {
        printf("%02X ", hashvalue[idx]);
    }
    printf("\n");
}

//To simulate the secure data transmission
void main()
{
    unsigned char plaintext[]={"This is the original input plaintext"};
    unsigned char *m;
    unsigned int msglen = strlen((char*)plaintext);
    unsigned int divnum = 0, idx;
    unsigned char *output=(unsigned char *)malloc(msglen*sizeof(unsigned char));
    divnum = msglen / 46;
    if (msglen % 46 != 0)
    {
        divnum = divnum + 1;
    }
    m = (unsigned char *)malloc(divnum*64*sizeof(unsigned char)+1);

    printf("The original message is:\n%s\n\n", plaintext);
sender(plaintext, m);
receiver(m, output);
printf("The recovered message is:\n\n", output);
}