BIOMIMETIC METAMORPHIC FRAMEWORK FOR SECURITY IN RESOURCE-CONSTRAINED WIRELESS NETWORKS

by

Raghav Vemagal Sampangi

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
March 2015

To my Parents,

and

the Divine Force within us all.

Table of Contents

List of Tables

List of Figures

Abstract

Contemporary mobile devices are being increasingly integrated with technologies such as RFID (radio frequency identification) and WBAN (wireless body area networks) that enable object identification and sensing in the Internet of Things (IoT). These technologies facilitate many applications such as efficient management of users' health, remote monitoring and asset tracking. Communication in these applications is mainly wireless, making it critical to ensure security. Important elements of security include data encryption, key management and authentication. Although sophisticated cryptography is the straightforward solution to achieve security goals, some IoT entities are limited in their ability to perform the necessary computations, owing to trade-offs between managing available resources and keeping them cost-effective.

The primary objective of this thesis is to propose a new metamorphic multi-algorithm framework for security in resource-constrained wireless networks. Our proposals draw inspiration from biological/natural systems and chaos theory to achieve security through unpredictability. While the state-of-the-art is focused on employing standard cryptographic techniques and customizing them for lightweight applications, our framework facilitates security through a dynamic, context-dependent choice of one of the algorithms for key management and authentication. Three new algorithms, which emphasize dynamic key management and unpredictability, are proposed as part of our framework. These are standalone algorithms that could be employed in resource-constrained networks either independently for generating keys and authentication parameters, or as part of the framework to increase the overall uncertainty, and consequently, security.

We analyze and assess each constituent algorithm and the possible framework configurations to verify that the choice of algorithms are at random, the keys generated remain unpredictable, and the resource utilization is low. We use a proof of concept implementation to generate keys and analyze them to assess algorithm choices, use Sörensen's Similarity Index (*SSI*) and NIST (National Institute of Standards and Technology) Statistical Test Suite to assess key sequence unpredictability and randomness, respectively, and use hardware implementation to assess resource utilization. Results encourage use of our framework and individual algorithms in resource-constrained applications, and its generic design implies that it can be extended and adapted for use in other non-resource-constrained application environments as well.

List of Abbreviations and Symbols Used

$+$	Addition operation
AND	Logical AND operation
$E()$	Encryption Function
$E_K()$	Encryption, with Key, K
F_O	Optional data field (HiveSec1 algorithm)
ID	Identifier or Identification number
K_G	Group-wise key (WBANs)
K_O	Outer-envelope key (HiveSec1 algorithm)
K_S	Encryption key (HiveSec1 algorithm)
K_T	Data transfer key (Butterfly1 algorithm)
K_i	Data encryption key (Butterfly1 algorithm)
Key_x	Generic key, interchangeably used
L_x	Location coordinate (latitude or longitude)
M	Message
$M1$	Message transmitted by initiator (HiveSec1 algorithm)
$M2$	Message transmitted by responder (HiveSec1 algorithm)
MC	Message Code
P	Periodicity of the pseudorandom number generator
SSN	Session sequence number
S_A	Actual seed
S_C	Chosen child seed
S_L	Left parent seed
S_R	Right parent seed
$Seed_G$	Seed to generate group-wise key (WBANs)
VER	1-bit version code
$\bullet$	Bitwise logical AND operation
δ	New session duration

δ_0	Default session duration
ϵ_{I_E}	Initiator error response
ϵ_{R_E}	Responder error response
η_t	Tag number
$\oplus$	Bitwise logical XOR operation
$\|$	Concatenation operation
$\phi()$	Butterfly seed transformation function
θ_i	Message signature (Butterfly1 algorithm)
ε_E	Threshold to classify a message as an attack
$\vee$	Bitwise logical OR operation
asv	Authentication Synchronization Vector (GeM1 and GeM2 algorithms)
c_i	Ciphertext
$cipher_i$	Ciphertext, interchangeably used
em	Encrypted message (GeM1 and GeM2 algorithms)
$f()$	Combination function
$g()$	Pseudorandom Number Generation Function
$genLimit$	Generation Limit, pattern deciding parent key updates
$h()$	Hash Function
i	Message sequence number, Butterfly1 algorithm
j	Butterfly seed state identifier
m_i	Message or encrypted message, interchangeably used
$msign_I$	Initiator message signature
$msign_R$	Responder message signature
n	Key size or length in bits
n_{Alg}	Number of available algorithms in the framework
n_δ	Session identifier
$numX$	First pseudorandom sequence generated by pseudorandom number generator

$pattern_{asv}$	Pattern used to generate authentication-synchronization vector
s	Pseudorandom number generator state, derived from t_s (HiveSec1)
s_e	Pseudorandom number generator state additive component
s_{O1}	Pseudorandom number generator state, derived from t_{s0} (HiveSec1)
s_{init}	Initial pseudorandom number generator seed
t_x	Timestamp
ACK	Acknowledgement
ASIC	Application Specific Integrated Circuit
BCU	Body Central Unit, or the WBAN hub
BSU	Body Sensor Unit, or a WBAN sensor
CLB	Configurable Logic Block
CRC	Cyclic Redundancy Check
DNA	Deoxyribonucleic Acid
EPC	Electronic Product Code
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things

LSB	Least Significant Bit
Main Key	A main key, or encryption key, is the continuously changing key, used for encrypting messages during a communication session.
MSB	Most Significant Bit
NFC	Near-Field Communication
NIST	National Institute of Standards and Technology
PAR	Place and Route
PRNG	Pseudorandom Number Generator
PS	Personal Server
RF	Radio Frequency
RFID	Radio Frequency Identification
RNG	Random Number Generator
Session Key	An optional session key is used to encrypt a message containing a component encrypted using a Main Key and other parameters. Session keys may remain common for a pre-determined session duration.
SSI	Sörensen's Similarity Index
STS	Statistical Test Suite
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
WBAN	Wireless Body Area Network
WSN	Wireless Sensor Network

XOR	Bitwise logical Exclusive-Or (XOR) operation
XST	Xilinx Synthesis Tool

Acknowledgements

My interest to pursue a Ph.D. was fueled by my interactions with my supervisor, Dr. Srinivas (Srini) Sampalli, whose support, feedback and guidance encouraged me to create what I present in this thesis. I am ever grateful that he gave me a chance to work with him and learn from him not only technological aspects, but humility and other attributes to become a better person as well. I also sincerely thank my committee members, Dr. Nur Zincir-Heywood, Dr. Vlado Keselj and Dr. Robert Beiko, who were gracious in their comments and suggestions, while driving me to further improve my work.

I would also like to thank Dr. Mauro Conti, from University of Padua, Italy, for graciously agreeing to be the external examiner, and for his thought-provoking suggestions and questions, which definitely contributed to making my thesis better.

Special thanks also go to the anonymous reviewers of my papers that were published during this journey. Their suggestions and comments contributed to improving my work a great deal.

My work would not have been possible if not for the support and funding given by The Boeing Company. I sincerely thank everyone on the Boeing team, who gave feedback, and helped me create what I present in this thesis. I would also like to thank Dr. Kirstie Hawkey, coordinator of the Mobile Graphics sub-project of the Boeing project for her words of support and encouragement during my years as a student member of this project.

Every day at the Faculty of Computer Science and our research lab, MYTechLab, has been an honour and a privilege, and I thank Dr. Sampalli, the Faculty and the Boeing project for making it happen. I would also like to thank the Faculty graduate administrators Ms. Menen Teferra and Ms. Vidhya Ramamoorthy for making the whole process easier. Special thanks also to Ms. Alicia Kirk from the Faculty of Graduate Studies, who worked tirelessly to coordinate and help with my thesis defence and submission.

I would like to extend special thanks to Dr. Musfiq Rahman (who is now the

Assistant Professor at Thompson Rivers University, BC), Saurabh Dey, Jayagopal Narayanaswamy and Jeremy Porter who were always around for brainstorming. Thanks also to Darshana Upadhyay from Nirma University for her help and suggestions as I worked on my experiments. I would also like to thank all members of MYTechLab for all the discussions and fun, which contributed to making my Ph.D. journey an enjoyable experience.

Over the years, I have also had the opportunity to work with the computer science student societies and organizing committees, and with several members of the faculty and staff in different activities. It has truly been a privilege working with all of you, for I have learned from each interaction, and I hope I have become a better person as a result. All of this made my student experience rewarding and I thank you for the opportunity.

Well, none of this would have happened if it wasn't for my parents who gave me life. Special thanks to my parents, Dr. Sowbhagyalakshmi and Kamalakar, because their constant emotional, spiritual support and guidance have made me the person I am today, and be able to write this thesis.

Having thanked everyone and everything we can see, talk to and perceive, one cannot ignore thanking the Divine force that resides in us all. Some might call it the laws of nature, some others might call it God, a few others might not acknowledge it at all, but there is some force that is keeping us alive and giving us the opportunity to experience all these marvellous aspects of life. I bow my head in gratitude for this life and this amazing opportunity.

Thank you all for an awesome experience.

Chapter 1

Introduction

1.1 Overview

Ava is an employee at a futuristic technology company. As part of her daily work routine, she uses multiple Radio Frequency Identification (RFID) tags to access different tools and parts of the building in which she works. She also uses her company's RFID-based application to guide her to specific tasks in appropriate locations. Recently, however, she was diagnosed with a medical condition and her doctor, *Ben*, advised her to be under constant monitoring. Understanding that her career was equally important, he advised her to wear wireless body sensors and use her cellular phone as part of a Wireless Body Area Network (WBAN) application to remotely monitor her health. Glad that she could work while her doctor was monitoring her health, she went back to her daily routine.

This situation, a portion of it or something similar, could be part of daily activities for many people around the world. Technology solutions such as RFID and WBANs are only two of the many applications that have contributed to making our activities, and hence our lives, convenient and easier to manage. Many such 'productivity' applications are already based on wireless communication, and using mobile devices such as cellular phones, tablet and laptop computers. Thus, although the infrastructure foundation of organizations would entirely be wired networks, wireless networks and their associated security challenges will continue to attract much research attention. Furthermore, the advent of cloud computing has enabled organizations with little infrastructure to deploy their systems almost entirely on the cloud, making them rely on security offerings by the cloud service providers.

Most of our security needs are met by standard symmetric and asymmetric cryptographic algorithms, which have succeeded in securing most aspects of contemporary communication. These algorithms use increased computations and complexity as a means to offer better security. However, resource constraints imposed by devices such as RFID tags or wireless body sensors, or even smartphones in some cases, increase the challenge to deploy such standard algorithms. Thus, security in resource-constrained wireless networks already are and will continue to be of significant interest in the future.

1.2 Resource-Constrained Wireless Networks

Wireless networks are said to be resource-constrained if their physical design imposes restrictions on their ability to store a large amount of information or to perform complex computations. Essentially, such networks and associated entities would have limited memory and computational power, some of which might be necessitated by the absence of on-chip power sources (in devices such as passive RFID tags) or longevity of the entity (as with wireless sensors or wireless body sensors). In this section, we present an overview of RFID systems as well as WBAN systems, exploring the need for creative and lightweight security solutions in both.

1.2.1 Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) is one of the technologies that is at the forefront of emerging wireless technologies. It is one of the fundamental elements in the "Internet of Things" and has inspired new technologies such as near field communications (NFC). The core feature of RFID is the ability to identify and locate objects uniquely among a set of other objects. This technology is extensively used in the retail industry and has more recently been used in healthcare applications.

The entity central to an RFID system (illustrated in Figure 1.1) is an electronic circuit known as RFID tag. Tags are simple electronic circuits whose main function is to respond to querying entities called RFID readers with a unique number. RFID readers are devices that send electromagnetic signals to the tags in a manner similar to querying an entity to know an answer. The electromagnetic signals transmitted by readers energize the tags sufficiently long (similar to an on-demand power source) so that they can respond with the unique number. On receiving the tag's response, the reader then forwards it to the backend enterprise server, which uses middleware for data cleaning/extraction from the response and retrieves detailed information about the object represented by the tag. The server then responds to the reader with this information [15].

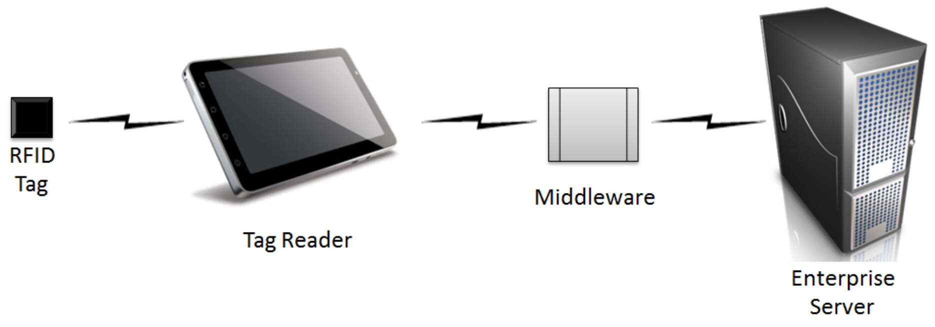


Figure 1.1: A Typical RFID System

RFID systems are classified into different types depending on the types of tags that are employed in said systems. RFID tags themselves are broadly categorized as passive tags, semi-passive tags, and active tags, based on the absence or presence of on-chip power sources [15, 16]. Passive tags do not have an on-chip power source, which limits them from performing complex computations and from initiating communication with readers. Passive tags, therefore, are dependent on electromagnetic signals sent by readers to energize them, and they respond with the data in the brief time when they are energized. Active tags, on the other hand, have an on-chip power source, and can, thus, either initiate the communication with a reader themselves, or can respond to a request from the reader. Semi-passive tags have an on-chip power source, however still require the energy from the reader for broadcasting their message.

Passive RFID tags perform one basic function — respond to any query by any reader, and possibly perform tasks as instructed by the reader. Such a functionality, coupled with the lack of resources for performing sophisticated authentication of the incoming instructions, makes passive tags vulnerable to a range of attacks, such as replay attack, tag killing, tag over-writing, etc. Furthermore, readers with high signal strength, also referred to as "rogue" readers, can read information from any tag, even if separated by a large distance. This further increases the vulnerability of the tags, increasing doubts in their widespread acceptance. When it comes to security in an RFID system, therefore, this inability makes RFID tags the weakest link.

Another classification of RFID tags is based on the computational capabilities [17]. Tags belonging to “full-fledged” class are able to perform complex cryptography including public key cryptography, hashes and complex symmetric cryptography; while tags in “simple” class have the ability to perform simpler computations such as pseudorandom number generation (PRNG) and hashing. “Lightweight” tags can perform pseudorandom number generation and operations such as CRC (Cyclic Redundancy Check), but do not support either complex cryptography or hashing. “Ultra-lightweight” tags are severely limited in their computational abilities and are capable of only simple bitwise logical operations such as AND, OR, Exclusive-OR (XOR), and so on.

Resource limitations, thus, not only limit a tag’s ability to perform complex computations, but also limit their ability to verify the authenticity of the other communicating entity (i.e. reader and/or server), which in all likelihood could instruct the tag to make changes to its internal state and if rogue, could in doing so render the tag useless. Although RFID readers and servers might possess the ability to perform complex cryptographic operations to be able to verify each other, the limits in the abilities of a tag require newer (lightweight) security proposals to be able to verify all entities involved, and to keep the system safe from unwarranted use.

1.2.2 Wireless Body Area Networks (WBAN)

Sensor networks are comprised of sensors (or detecting elements) that *sense* the environment in which they are placed, record data about the environment, and relay the recorded data to a central monitoring station or base station, where data is aggregated. A simple example for this would be the monitors connected to patients in critical care units in hospitals, which monitor their parameters such as blood pressure, oxygen saturation, heart function, among others. Typically, such a system would have wired *leads*, which record the specific parameter, connected to the monitor (inclusive of the signal processor) where recorded data is processed and displayed. This is an example for a *wired sensor network*.

Wired sensor networks, however, limit the deployment zone for the application of interest. Nevertheless, advances in wireless technologies have given rise to wireless sensors, which can be deployed in remote locations, and data monitored at a central

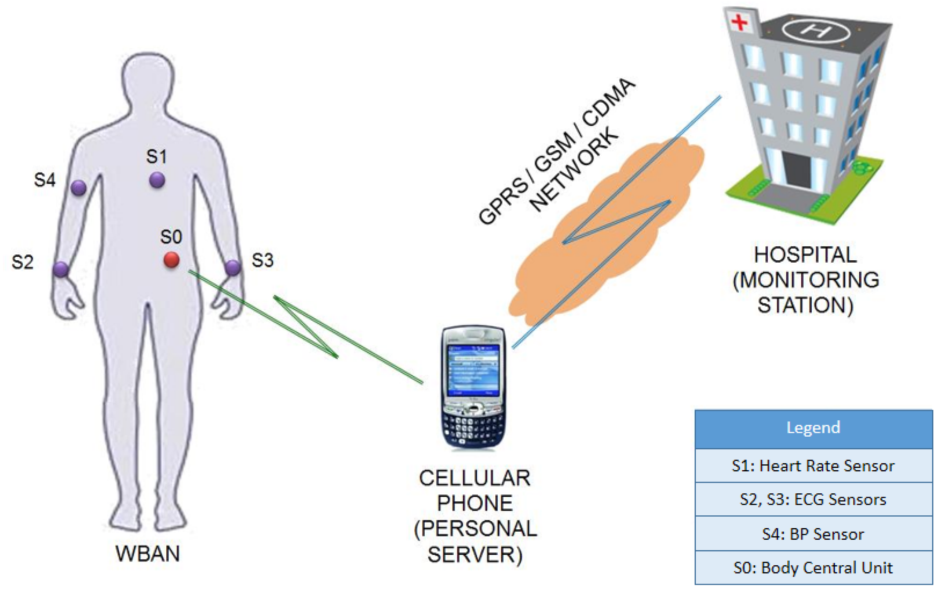


Figure 1.2: A Typical WBAN System [10]

location. A *Wireless Sensor Network* (WSN) is a set up in which a group of sensors are equipped with independent power sources and processing circuits intended to 'sense' or record data about specific parameters in the environment where they are deployed. WSNs, thus, enable deploying sensors in environments that can be considered out of human reach, for example, in a battlefield to help in detecting the presence and movement of humans or any other animals, thereby contributing to real-time tracking of a battlefield [18].

One noble application area for WSNs is healthcare. The remote monitoring feature of WSNs can be used to monitor health of persons remotely, and this has led to the birth of Wireless Body Area Networks (WBANs). In WBANs, sensors are used to record and relay health data to hospitals or other monitoring stations, from where healthcare professionals can remotely monitor the health parameters of patients or other individuals. In a remote healthcare setting, this enables doctors to monitor patients continuously, and keep tabs on events that could aggravate and potentially prove fatal to the patient at a future time. Another particularly useful application of WBANs is in the military, where commanding officers could monitor the health of soldiers in the field.

In WBANs, a sparse network of sensors (referred to as 'Body Sensor Units', BSUs) are deployed either directly on the human body, inside the body or embedded in everyday clothes, to record and transmit health data. BSUs record and transmit data to a Body Central Unit (BCU) or the “WBAN hub” [19], which aggregates data sent by all BSUs and relays the aggregation to a hospital monitoring station. However, a BCU does not have sufficient infrastructure to relay the data directly to a monitoring station, and therefore, uses a Personal Server (PS) as an intermediary. A personal server could be either fixed or mobile, typical examples of which include cellular phones, personal digital assistants (PDAs), desktop or laptop computers [20]. A typical set up of wireless body area networks is as illustrated in Figure 1.2, with BSUs — S1 (heart rate, measured in beats per minute, bpm), S2 (blood pressure in millimeters of Mercury, mmHg), and S3 (blood glucose level in milligrams per decilitre, mg/dl), and S_0 being the BCU, using cellular phone as a personal server.

The fact that WBAN sensors reside on the human body implies that the on-chip power sources or batteries have to have longer lives, so as to not cause inconvenience to the user in case of battery replacement. This is particularly true in case of applications where long-term monitoring of a person is required (for example, remote monitoring of high-risk individuals [21] with ischaemic heart disease [22] or with suspected pregnancy-induced hypertension [23]). This further implies that the design of BSUs must be optimized in their resource utilization, thereby making WBANs resource-constrained. WBAN sensors have several constraints, including restrictions on size and power consumption, electromagnetic power, latency, reliability, but most important of all, security and privacy of user health data [24][25].

1.2.3 A Look Back at Resource-Constrained Wireless Networks

We have presented two of the many systems that have severe resource restrictions imposed due to a multitude of reasons. Such systems, although able to perform their functions, need to be protected against unwarranted accesses and attacks. For example, an RFID application to grant role-based access to resources has to be protected to preserve unauthorized accesses to resources in an organization, while WBAN applications need to be protected against data modification and unauthorized data accesses. Therefore, research on strengthening data privacy and security of resource-constrained wireless networks has assumed focus in the recent years.

Some of the existing research [26, 27] has focused on using pre-shared secrets and simple bitwise operations such as XOR (exclusive-OR) to perform computations for authentication, while most others [28, 29, 30] use entities such as enterprise servers (RFID) or monitoring stations (WBAN) to generate and send encryption/authentication parameters to tags or sensors, respectively, to accomplish security. Although such systems rely on the quality of the backend infrastructure (servers or monitoring stations) being secure, one can never ignore the presence of 'rogue' entities in the communication path between the servers and tags/sensors, who could monitor and potentially hijack the conversation or manipulate data or in the worst case, bring down the system.

1.3 Thesis Contributions

We consider Kerckhoffs's principle as one of the motivating factors for our work, according to which we would need to assume that all information about an cryptographic algorithm is available for everyone's perusal [31]. This, in addition to standards and specifications [32, 33, 19] publishing the details of implementations of algorithms to be employed by systems claiming conformance, makes keys and associated authentication parameters more important to the overall security of the system. Typical key management protocols require entities exchanging either key derivation parameters [34] or exchanging key materials themselves [8] to agree on the keys to be used for encryption/decryption. However, the wireless communication medium leaves transmitted packets open to be 'sniffed' by attackers, who could employ additional computing resource to crack these keys. This leads us to believe that design of systems with a metamorphic structure, i.e. changing at random instances, would increase the security in a system. Basically, we are referring to reconfigurable security architectures, with continually changing (context-dependent) internal system states. In this thesis, we present a new metamorphic modular security framework, which envelopes three algorithms offering solutions to multiple security goals, such as key management, authentication and integrity verification.

The key management algorithms proposed in this thesis are based on the fundamental concept of eliminating the need for actual key exchange. We believe in generating new keys at both the sender and the receiver instead of actually exchanging keys; a concept justified by the Diffie-Hellman Key Exchange algorithm [34]. This is because generating keys with only a few parameters exchanged, removes the dependency on a secure communication channel to ensure security. This way, the two communicating entities may still be secure despite the channel being compromised, although this is undesirable. The entities maintaining the state of key generation independent also ensures that they can verify each other's authenticity, while increasing the overall unpredictability associated with the system, hence making it more secure.

The focus of this thesis is also simplicity, which is necessitated by the resource restrictions imposed by the application environments we have considered. For this purpose, we draw inspirations from natural/biological concepts, and propose mechanisms that mimic some of them. This has enabled us to design 'lightweight' security proposals, reducing the resources required for implementation, but increasing the overall security of the system.

1.4 Outline

The rest of this thesis is organized as follows — we present a detailed discussion on security in general and as applied to resource-constrained wireless networks (such as RFID and WBAN systems), in addition to summarizing the associated specifications and standards, in Chapter 2, followed by a summary of the scope of our work, motivation and objectives in Chapter 3. We present our proposal of the new reconfigurable security framework in Chapter 4, followed by a description of its constituent algorithms in Chapter 5. We summarize the framework and the algorithms, and discuss their application — individually, as well as collectively, in Chapter 6. In Chapter 7, we present the results obtained from our analyses and discuss their implications in Chapter 8, before discussing concluding remarks in Chapter 9. Detailed results of the NIST randomness assessment, simulation results from hardware assessment, details of the hardware complexity assessment, list of publications from our work and copyright permissions to include published material in this thesis are presented in Appendices A, B, C, D and E, respectively.

Chapter 2

Background and Related Work

2.1 Overview

Our work, as discussed previously, proposes an overall reconfigurable security framework for resource-constrained wireless networks, with an objective to accomplish multiple security objectives with simple logical operations. In this chapter, we briefly discuss some of the general information security principles. We then present a brief discussion on the aspects of security in resource-constrained wireless networks, specifically, as applied to RFID systems and WBAN applications, before summarizing various key management schemes and authentication protocols used in these resource-constrained wireless networks. Although our proposal is a 'security' framework including all aspects of information security such as key management, encryption and authentication (referred to collectively as an encryption scheme), the focus is mainly key management and authentication — key management, since a publicly available encryption algorithm is only as good as the key, and authentication, which is primarily to be established (using generated keys and message signatures) among communicating entities to ensure secure (trust-based) communication. The proposals in our work are designed to work with any symmetric encryption algorithm.

2.2 Information Security Principles

Securing information communicated between a sender and a receiver involves encrypting the information to convert the message into a form unrecognizable to an unauthorized entity, using either a symmetric key or an asymmetric key encryption algorithm [35, 1]. Symmetric key algorithms work by using a single *secret key* for both encryption and decryption of the message. The sender and receiver come to an agreement over the key used for encryption through a key agreement/management process, where the keys used could be pre-shared, exchanged and agreed upon prior

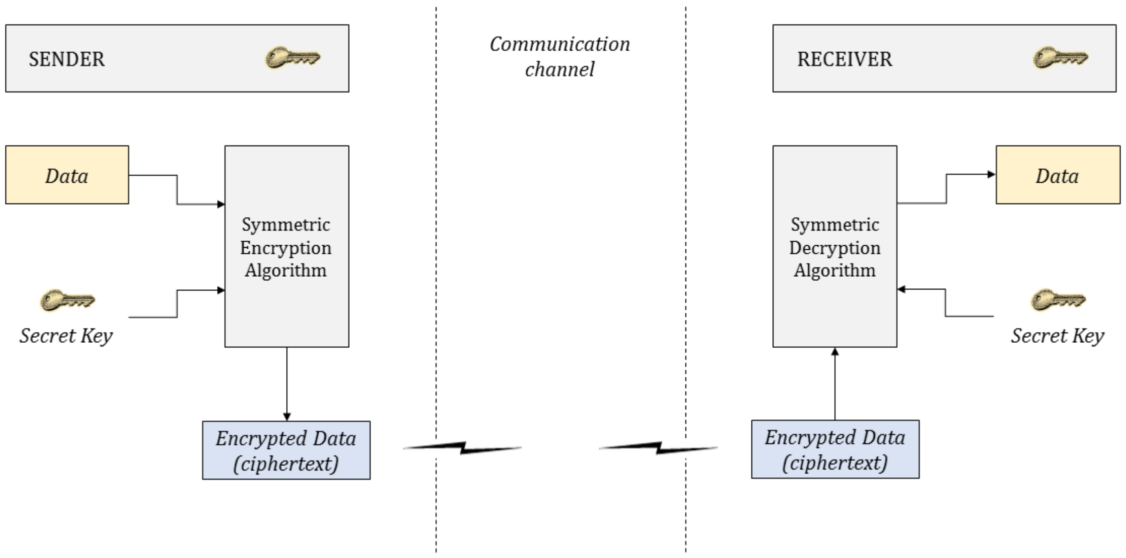

SENDER
Communication channel
RECEIVER
Data
Symmetric Encryption Algorithm
Secret Key
Encrypted Data (ciphertext)
Symmetric Decryption Algorithm
Data
Secret Key
Encrypted Data (ciphertext)

Figure 2.1: Working of a Symmetric cryptosystem

to the communication, or generated independently at the sender and receiver. The encryption itself is accomplished by confusion and diffusion, applying substitution and permutation operations on the data, in addition to specific processing using the key. Figure 2.1 illustrates the typical working of a symmetric cryptosystem, with the sender and receiver using a pre-shared key for encryption and decryption.

Asymmetric key algorithms, on the other hand, accomplish security by using two different (mathematically related) keys for encryption and decryption. These algorithms involve each entity having a publicly available/accessible key called the *public key* and a secret *private key*, known only to that entity. When an entity, *Alice*, wants to communicate with another entity *Bob*, she uses Bob's public key to encrypt the information, which Bob can access by decrypting the ciphertext using his private key. Asymmetric key cryptography depends on the presence of a trusted certificate authority that 'certifies' public-private key pairs, which essentially forms the public key infrastructure (PKI). The presence of two keys, especially when generated by trusted certificate authorities, facilitates user authentication and generation of digital message signatures, which help in ascertaining the integrity of the communicated message. In contrast to symmetric cryptosystems, encryption in asymmetric cryptosystems accomplishes security using complex mathematical operations and relatively large prime numbers as keys. The working of a typical asymmetric cryptosystem is illustrated in

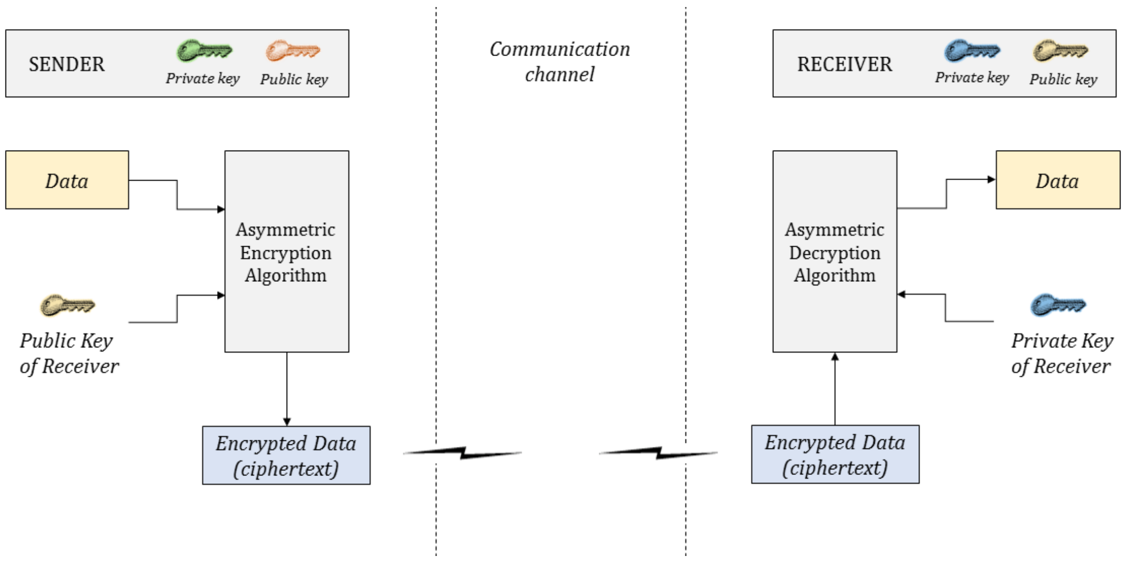


Figure 2.2: Working of an Asymmetric cryptosystem

Figure 2.2.

Historically, security of information would be accomplished by hiding both the encryption algorithm and the key used, a well-known example being the Enigma cipher machine, used by the German military prior to and during World War II [36]. However, with most present day encryption algorithms being published, and some standardized by recognized agencies, the most important aspect of the encryption process remains retaining the keys secret. In this context, the keys include the (shared) secret key in symmetric cryptosystems and the private key in asymmetric cryptosystems. Thus, key agreement between communicating entities is essential to ensuring security in any scenario, since the security of a cryptosystem/encryption algorithm that is publicly available will depend on the security of the keys used (Kerckhoff's principle [31]).

Once a message is encrypted using the key, the next objective becomes ascertaining that the received message is the same as the sent message (*message integrity*). To accomplish this, a straightforward way is to use integrity check algorithms, i.e. message digests (MDs) and message authentication codes (MACs). As discussed earlier, the presence of two keys, typically verified by a certificate authority, makes message integrity verification almost an implicit functionality in asymmetric cryptosystems. However, it becomes a critical (additional) component in symmetric cryptosystems.

Mathematical one-way hash functions have been a popular means of “signing” messages in symmetric cryptosystems, since they help in generating unique digests of each message, with very low probability of collisions. Collision in this case refers to a scenario when two inputs produce the same output. The design of these hash algorithms mandates low collisions and their design is based on the avalanche effect, i.e. a one bit change in the input must lead to a significant change in the corresponding output. Thus, MDs help accomplish verification of message integrity in most applications. Several MD algorithms, including MD5, SHA-3 (Secure Hashing Algorithm), RIPEMD, are popularly used [35]. However, some applications might necessitate the use of mechanisms to ensure that the messages are “authenticated”, i.e. providing the means of detecting any changes to the transmitted message, since the entities use a pre-shared secret key to generate the MAC. The general principles of Hashed-MAC (HMAC) algorithms are based on concatenating the key with the message and hashing this pair. Their ability to sign messages and any other components, with or without a key, enables hash (or keyed hash) algorithms to be a popular choice for entity authentication as well, particularly in resource-constrained networks.

Thus, the overall objective of cryptosystems is to ensure end-to-end security of the information. A good encryption scheme, including algorithms for key management, encryption, authentication and message signature generation, must be able to satisfy the generic security goals summarized in Table 2.1.

Having discussed generic security principles in this section, we explore security as applied to resource-constrained wireless networks in the next section.

2.3 Security in Resource-Constrained Wireless Networks

The term *resource-constrained wireless networks* encompasses a wide variety of technologies and applications that are disparate, and have varying requirements. Such requirements could include resource (memory and computational ability) requirements, data storage requirements, but have a common security requirement. Since much of the resource-constrained wireless networks, such as RFID systems, WBAN systems, Vehicular Ad-hoc Networks (VANETs) and the like, are created as autonomous systems to facilitate one independent activity in our lives, they are all in some way related to peoples’ personal data. This places an emphasis on protecting data being

Table 2.1: Security Goals [1]

Security Goal	Description
Confidentiality	Eavesdropping on communicated data packets must not be fruitful to an adversary
Integrity	The message sent by the sender must be the same as the message received by the receiver
Authentication	Validation of the sender by the receiver (vice-versa)
Non-repudiation	A communicating entity cannot later deny that a message sent originated at its location
Forward security	Adversary should not be able to derive future keys, given a knowledge of a contiguous set of previously used keys
Backward security	Adversary should not be able to generate previous keys, given a knowledge of a contiguous set of future keys

communicated in such systems, thereby preserving the privacy of the individual(s) in question. In this section, we explore such requirements, following which we discuss some of the existing work in two resource-constrained systems, namely, RFID systems and WBAN applications.

2.3.1 Privacy and Security in RFID Systems

RFID tags are central to RFID systems. These simple electronic circuits often have no other functionality than to respond to reader (or interrogator) queries. They are severely resource-constrained and typically store unique identifiers (IDs), which help in identifying objects uniquely among a set of other RFID tagged similar/dissimilar objects [15]. Their hardware simplicity implies that they are resource-constrained and any bit of improvement in efficiency or re-use in circuits would be beneficial. Security in passive RFID tag-based systems is, therefore, always a critical factor to ensure the overall success of the application. This makes it challenging to deploy complex algorithms, which require a high degree of computation for achieving security. On the other end of the scale, simple algorithms may prove easier for an adversary to crack.

With RFID tags being deployed to assist in identifying objects uniquely, care must be ensured to protect the privacy of this ‘object’. Often, this ‘object’ is a person using a tagged product in their daily life, or an organization representative using a tagged product for protected access to certain resources. Thus, if the responses of an RFID tag do not change with time, it would be very easy to ‘track’ its movements, thereby compromising the user’s privacy. This requires making the tag responses time-variant, which requires that the key refreshes or updates be frequent. Furthermore, since many of the RFID protocols use key management/encryption for authentication, the need to protect such key exchange messages assumes priority. Ideally, tags and other active entities in the communication should be able to generate keys on their own, without having to exchange keys, increasing the security of the system, and protecting the privacy of the user.

2.3.2 Privacy and Security in WBAN Applications

WBAN sensors are placed on (or in) a person’s body and record sensitive health data about that person. Such data, therefore, must not be accessible to anyone other than the person him/her-self, their family and healthcare professionals. Access control, thus, becomes a primary security goal for WBAN systems, and is closely followed by confidentiality, integrity, dependability and authentication [37]. However, there is always a compromise between functionality and security. This is mainly because the system application might be toward monitoring a health condition requiring continuous monitoring of the person’s health parameters, while the sensor’s ability to record data might limit the sophistication of the security solution used to protect the data it records. This is necessitated by the constraints on storage, computation and longevity of the sensor itself.

Furthermore, with the data recorded by the sensors having a direct impact on the health of an individual, the security and functionality requirements need to be carefully balanced so as to secure the data, protect the individual’s privacy and to facilitate longevity of the sensors. Ideally, as with RFID tags, body sensor communications must be secured using time-variant keys and strong encryption algorithms.

2.3.3 Elements of Security in Resource-Constrained Wireless Networks

As discussed in the Section 2.2, conventional systems rely on the following broad techniques for security — either on using shared secret keys and complex substitution/permutation functions as with symmetric cryptosystems, or on longer key-pairs and complex mathematical functions as with asymmetric cryptosystems. However, restrictions in resource-constrained wireless networks limit the size of keys that can be used, and the type of operations that can be performed while ensuring security and longevity. Each fundamental element of security, thus, needs to be customized and adapted for application in resource-constrained wireless networks. In this subsection, we discuss how data encryption, key management and authentication are accomplished in resource-constrained wireless networks.

Data Encryption

Confidentiality is the primary goal of any information security scheme. Encrypting data using either symmetric or asymmetric key algorithms helps accomplish confidentiality, and the various processes employed are typically linked in a manner to satisfy several goals highlighted in Table 2.1. In resource-constrained wireless networks, sophistication in the encryption algorithm used is dependent on the resources available and the security need for specific applications. Some of the solutions for encryption in resource-constrained wireless networks include 'lightweight' solutions based on stream ciphers [38] or adapting AES (Advanced Encryption Standard) to function as a stream cipher for lightweight body sensors [39], optimized block cipher-based encryption algorithms such as SEED [40], Tiny Encryption Algorithm (TEA) [41] or extended TEA [42], authenticated encryption solutions [43], encryption based on chaotic sequences [44], and lightweight AES-based solutions [45]. A few solutions consider a hybrid approach to encryption such as using a combination of AES and Elliptic Curve Cryptography (ECC) [46, 47], while some have even explored adapting asymmetric encryption for resource-constrained applications [48, 49, 50].

In many other approaches [51, 52, 8, 12, 13, 10] however, the Exclusive-OR (XOR) function is used for encryption. The key aspect of XOR that makes it almost an obvious choice for encryption in resource-constrained applications is its involutory nature

(XOR is its own inverse). When XOR is used for encryption, its decryption algorithm is also XOR, thereby facilitating the use of the encryption circuit for decryption as well. Using other logical operations such as AND, OR, etc. requires constructing separate circuits for encryption and decryption, making XOR easier to implement on low resource devices. Furthermore, ensuring random and frequently updated secret keys makes XOR a secure choice for encryption, as observed with stream ciphers [35, 1].

Key Management

Typically, key agreement is a separate phase prior to the communication in resource-constrained wireless networks, or is in a way combined with the message exchange protocol to reduce the overhead. This most common approach involves the trusted enterprise entity or the server performing key updates and exchanging key materials with the resource-constrained entity prior to or during their communication [8, 53, 54, 55, 56, 57], or in some other cases, could involve using pre-shared secrets or initialization vectors in pseudorandom number generators (PRNGs) [58, 59, 60]. As noted by Rahman [61], key management typically involves the following phases:

- *Key Setup*: This is the phase prior to the actual key generation or communication, involving communicating entities agreeing on key materials, initialization vectors, algorithms, etc. In some cases [62], this could involve a separate registration phase.

- *Key Exchange*: This is the actual key exchange and agreement. If using synchronized PRNGs, this might be an implicit process, or in other cases, it might involve an additional phase where keys are exchanged between the entities.

- *Key Refresh*: Entities could communicate for varying lengths of time. Typically, communication for longer durations involve entities refreshing or updating their encryption keys, so as to make it harder for unauthorized entities to crack the keys being used.

- *Key Revocation*: In some cases, ensuring system protection might necessitate "eviction" of entities that are compromised. Key revocation techniques help in updating the system to a new state, with or without changes to the compromised entities.

Although these key management phases [61] were discussed by Rahman in the context of wireless sensor networks (WSNs), these phases broadly guide the design of key management approaches in other resource-constrained wireless networks such as RFID and WBANs. Furthermore, with resource restrictions limiting the sophistication of the encryption algorithms used in RFID tags or WBAN sensors, key management becomes a critical element of security in resource-constrained wireless networks. Application specific requirements might guide the actual key management processes in place, however, in general, keys need to be regularly updated to ensure high unpredictability and therefore, high security.

Authentication

It is essential in communication systems for entities to verify the legitimacy or authenticity of each other, more so in resource-constrained systems as the servers typically have the ability to control the actions of the resource-constrained entities. For example, in RFID systems, a tag would first need to ascertain that the server (or a server-verified reader) is authentic, before performing key updates or other actions. Similarly, the server needs to know that the tag is authentic to ensure that the communication is not controlled by an unauthorized entity. In some systems, only one-way authentication is ensured, for example a server verifying the RFID tag. This could be the case when systems assume the trustworthiness of the server. However, it is essential in all systems for both entities to authenticate each other, or accomplish *mutual authentication*, since it adds to the overall trustworthiness of the communication.

Asymmetric key cryptosystems implicitly facilitate mutual authentication by using trusted certificate authority verified key pairs and digital signatures. While normal encryption requires the sender to encrypt data using the public key of the receiver, digital signatures are signed by the sender by encrypting the message hash using the sender's private key. This enables the receiver to decrypt the message hash using the sender's public key, facilitating entity authentication and message integrity verification. However, this is not possible with symmetric cryptosystems, since the same pre-shared or agreed key is used for both encryption and decryption. This scenario is further complicated in resource-constrained systems due to limitations in their computational abilities [35, 1].

In many symmetric key systems including resource-constrained applications, the key agreement scheme is used in combination with other operations, such as hash, to accomplish authentication [52, 9, 63, 55, 56]. In RFID systems, with much of the sensitive data about the tag being stored on the backend server, key agreement schemes are typically used for authentication of entities. However, this is contrary to the requirements in WBAN systems, where data recorded by each sensor is sensitive and critical, since medical decisions are based on the recorded data.

Cai et al. [64] propose a set of rules for designing and evaluating authentication proposals for RFID systems, considering prevalent security issues and the security objectives that need to be satisfied by such schemes. The rules suggest the protocols — (a) are lightweight (*lightweight rule*), (b) facilitate authentication of each entity by the other (*mutual authentication rule*), (c) facilitate key updates following successful authentication (*key update rule*), (d) ensure synchronization of states to accomplish proper key updates and authentication (*synchronization rule*), (e) should facilitate deployment in an application scenario with a large number of tags (*scalability rule*), (f) ensure only authenticated readers be included in the communication (*confidentiality rule*), (g) introduce randomization in transmitted information (*indistinguishability rule*), and (h) must ensure the information remains valid (*data integrity rule*). R-RAPSE essentially translates various security goals to specific objectives that must be considered when designing authentication schemes. While their proposal is in the context of RFID systems, its generic rules enable its use in designing authentication schemes for all resource-constrained wireless networks.

2.3.4 Standards Governing Proposals for RFID Systems and WBAN Applications

Increased awareness about and emphasis on developing technologies for the Internet of Things (IoT) paradigm have meant that autonomous object identification and personal data management technologies such as RFID and WBANs are increasing in their popularity. Standardization organizations and working groups are working to guide the process of developing products using these technologies, releasing regulations and guidelines for creating each aspect of data management, communication and security. EPCglobal is working on developing a standard for use of RFID technology and sharing data over the Internet [65], while the use of wireless sensors for healthcare and other applications is governed by the specifications released by Task Group 6 of the IEEE 802.15 working group [66] (IEEE 802.15 works towards specifications for wireless personal area networks, WPAN [19]). In this section, we summarize the security considerations specified by these agencies.

EPCglobal RFID Tag Security Specifications

Standard overview

A conglomerate of organizations that develops and maintains standards for supply chain management, EPCglobal [65] works to employ and standardize the use of electronic product codes (EPC) to better manage and uniquely identify physical objects. RFID technology is currently being used as a key component in this effort, particularly due to its non-line of sight operation and reduced costs [15]. EPCglobal categorises RFID tags as [16]:

- *Class-0/Class-1: Identity tags* — passive tags with minimum features including EPC identifier (with optional minimal security);
- *Class-2: Higher-functionality tags* — passive tags that support authenticated access control in addition to EPC identifier;
- *Class-3: Battery-assisted passive tags, or, semi-passive tags* — have all the features of higher-functionality tags, with the presence of an on-chip power source; and,
- *Class-4: Active tags* — possesses all features of semi-passive tags, in addition to being able to initiate communications with readers or interrogators.

The specification and standardization process for RFID application in supply chain management is ongoing, with the current specifications including those for communication in Class 0 and Class 1. Class 0 tags are used in applications where data is written only once and read many times and are governed by the EPCglobal Class 0 tag specifications [67], while Class 1 tags facilitate multiple read and write operations. The functional specifications for Class 1 tags are currently in their second “generation” (Generation 2 version 2, referred to as Gen2v2), with enhancements to include basic security and error correction features, including optional support for cryptographic suites [32]. Cryptographic suites, if used, are suggested to be one of the following — (a) As specified by ISO/IEC (International Standards Organization/ International Electrotechnical Commision) in the standard specification 29167-1:2014 [33], under security services for RFID air interfaces; (b) As specified by GS1 (GS1 is one of the collaborating organizations in EPCglobal); or, (c) As specified/chosen by the tag manufacturer [32]. The specifications for higher classes of tags have not been formalized and are in the early stages of definition [68].

Tag communication is typically 'queried' by an interrogator or RFID reader, with the exception of active tags. Their responses are powered by the electromagnetic signal sent by the reader and are accomplished through backscattering. All communication in an RFID system are with the most-significant bit (MSB) sent first. During the communication, a reader might 'instruct' the tag to perform certain actions, e.g. sleep or kill through access passwords stored in the tag's reserved memory. To verify such commands, the tag first verifies the command using CRC (cyclic redundancy check), following any authentication processes that may be implemented by the application. According to the specification, the maximum size of any tag response is capped at 32 kbits ($32 \times 1024 = 32768$ bits). The specification also enables manufacturers to set an optional "security timeout" feature, which the tag could enable based on failed attempts to kill the tag, or failed authentication, key update and other such processes. Depending on the security needs of the application, the manufacturers can set the timeout value, which is "measured relative to the last rising edge of the last bit" [32] of the response by the reader that caused the tag to enter the timeout phase.

EPCglobal Gen2v2 specification requires a tag to implement random number generators (RNG) or pseudorandom number generators (PRNG) for inventory and password operations, which are required to generate 16 bit random sequences. However, such generators would be in addition to any cryptographically secure RNG/PRNG that may be required by the implementations of the optional cryptographic suite. If the tag implements security, the following optional "security access commands" may be used to secure the communication between the readers and tags — (a) *Authenticate*, to ensure one-way or mutual authentication; (b) *AuthComm*, which allows authenticated communication; (c) *SecureComm*, which employs solutions deployed in the cryptographic suite to accomplish secure (encrypted) communication; (d) *KeyUpdate*, which allows readers (and servers) to change the key stored on the tag; (e) *TagPrivilege*, allowing the reader to access and modify privileges for access passwords or keys.

Cryptographic suites, as discussed earlier, are optional. However, if present, they could be one or more, which defines the security of the communication between a reader and a tag. On authentication, a tag enters the 'secured' state and the communicated messages are encapsulated. To conform to the standards, the manufacturer choosing to implement cryptographic suites is required to ensure that the authenticate and key update processes proceed as specified by the specification or the independent cryptographic suite chosen.

ISO/IEC 29167-1:2014 Security services for RFID air interfaces

The ISO/IEC 29167-1:2014 [33] standard suggests the means for the functionality for tags to be *untraceable* (with a new special mode called, untraceability mode, to hide part or all of its identity), able to *certify authenticity* (by using techniques to generate certificates for authentication), able to *encrypt* data, and ensure *secure access* to data and functions in its memory. One of the key elements of the specification is the implementation of cryptographically secure RNG or PRNG as specified in ISO/IEC 18031 [72], since much of the key generation and other processes in the cryptographic suites are dependent on (pseudo)random numbers. A cryptographic suite, as defined in this specification, is a scheme or a mechanism to apply the cryptographic functions in the algorithm to generate the (ciphertext) output. This specification suggests the use of one (or more) of the following cryptographic suites, depending on the security requirements of the application (Note that as on January 1, 2015, several of these standard specifications are still under various stages of development, with only one of the cryptographic suite specifications being published [73]):

- Advanced Encryption Standard, 128 bit mode of operation (AES-128) [74].
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 10;

- PRESENT-80 [75].
  This specification is complete and published as ISO/IEC 29167 - Part 11 [76];

- ECC-DH (Diffie-Hellman Key Exchange using Elliptic Curve Cryptography, also referred to as ECDH) [77].
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 12;

- Grain-128A (Grain-128 stream cipher with authentication) [78].
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 13;

- AES, used in OFB (output feedback) mode of operation (AES OFB).
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 14;

- XOR (Exclusive-OR).
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 15;

- Elliptic Curve Digital Signature Algorithm - ECDH (ECDSA-ECDH) [79].
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 16;

- cryptoGPS.
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 17;

- RAMON [80, 81, 82].
  This specification is under development and will be as specified in ISO/IEC 29167 - Part 19;

Discussion

The primary application of RFID tags is object identification. The standards being proposed by EPCglobal is to ensure a common data management mechanism so as to make it easier for organizations worldwide to better manage their manufacturing /retail processes. However, developments in information technologies continue to facilitate use of RFID in other applications, including and perhaps most importantly, the Internet of Things. This is primarily aided by the ability of an RFID tag to help identify any object (to which it is tagged) in the world uniquely. Although its benefits are profound, this has the potential for misuse by agencies who wish to 'track' a person using an object that is tagged. This calls for standards organizations such as EPCglobal and ISO/IEC to propose guidelines to maintain the data exchange secure, in order to ensure user privacy. That said, RFID tags are resource-constrained, with limits to the storage, power and size, which restricts the type of security that can be deployed. Accomplishing security through (sophisticated) cryptography "impacts power consumption and processing time for the RFID components and may degrade system performance" [33], which would inevitably mandate increase in resources, thereby increasing the cost. The trade-off between cost, application functionality and security therefore, guides the design of RFID tags, and is primarily one of the main reasons why the security suites are still classified as 'optional' services [32, 33].

IEEE 802.15.6: Wireless Body Area Networks Security Specifications

Standard overview

The IEEE 802.15.6 standard specification provides various guidelines for designing the communication between a node (WBAN sensor) and a hub (WBAN body central node). We summarize the security specifications of the standard in this section. Primarily, the communication between nodes and hubs might be one of the following modes [66, 83]: (a) *Level 0 – unsecured*, with data transmission in "unsecured frames"; (b) *Level 1 – authentication but not encryption*, with facility for verification of message integrity and authenticity, but not confidentiality and privacy protection; or (c) *Level 2 – authentication and encryption*, with messages being secure, authenticated and encrypted. Nodes and hubs are required to store (or generate) a pre-shared master key, which is activated during the security association prior to communication, while they are required to generate "pairwise temporal keys" once per communication session. If the chosen mode is either level 1 or level 2, each message is required to support the specific type of data security.

In the communication modes suggested in the standard specification, the nodes can be in an inactive or "sleep" state, to be 'woken up' by the hub for communication; a function similar to passive/active RFID tags, which are 'woken up' by the signals from the readers.

The security association between nodes and hubs (interchangeably and collectively referred to in this thesis as ‘entities’) includes the entities agreeing on the security association protocol, on the security level and the cipher function to be used. IEEE 802.15.6 suggests the use of either AES-128 or Camellia-128 encryption algorithms, with a possibility for extending the choice by fourteen more algorithms with changes to requirements and standard updates. When using either of the supported encryption algorithms, the entities can use one of the following security association and key agreement protocols:

- *Master key pre-shared association* — The entities use the pre-shared master key to generate the pairwise temporal key on association;

- *Unauthenticated association* — This mode is based on Diffie-Hellman (DH) key exchange that employs elliptic curve public key cryptography (ECC)-based key generation. The node, in this case, does not authenticate the hub, but continues with the security association, by computing the master key using a cipher-based message authentication code (CMAC) function.

- *Public key hidden authentication* — Also based on DH key exchange employing ECC. This is similar in its master key computation to the unauthenticated association process, however, the authentication and key agreement process is preceded by the node secretly sending its public key to the hub.

- *Password authentication association* — Also based on DH key exchange employing ECC. In this mode, the entities store a pre-shared password that is used to authenticate the association and hence, the master key generation. In a manner similar to the public key hidden authentication, the node transmits its public key that is “scrambled” by the password to the hub, which is then used to compute the DH key for communication.

- *Display authenticated association* — Also based on DH key exchange employing ECC. This mode involves the node computing a number called “witness”, using a nonce, the addresses of the node and the hub, and its public key components. This is transferred to the hub and the security association proceeds as with the unauthenticated association, by generating a master key using CMAC. Following generation of the master key, the hub computes the witness and compares its value to the one sent by the node prior to the association. If authentication and association are successful, the entities display a 5-digit decimal number (also computed using CMAC of the entity nonces and the temporary DH key) on their user interfaces.

Following the association phase, the entities compute a pairwise temporal key (PTK) for the remainder of the session, until one of the entities requests disassociation. The entities compute the PTK by extracting a 128 bit value from the CMAC function using the master key to encrypt the entity addresses and their nonces, and the PTK_control field value (extracted from the first frame sent by the initiator during PTK creation). The PTK creation process involves entities computing three keys using the CMAC function — PTK, Key confirmation key (KCK) and a final key, P. While PTK and KCK are generated using the master key, P is generated using KCK and is used to compute a keyed MAC for authentication. Following PTK generation, the hub may generate and distribute a "group temporal key" to be used for multicast transmissions to the sensor nodes. The standard specifies using AES CCM (Counter with Cipher-block-chaining message authentication code) mode [85] for encryption of messages and for authentication, and facilitates the use of other (non-AES) cipher functions, if required.

Discussion

The IEEE 802.15.6 standard specification specifies the norms for designing and securing communications among sensors in WBANs, should the application be recognized to conform to said standard. We can wonder as to whether WBANs can still be considered resource-constrained networks when the sensors are expected to support sophisticated encryption algorithms such as AES and even public key cryptography in the ECC-based Diffie-Hellman key exchange process. Stringent security measures are necessary to protect data, especially when WBANs are used for remote health monitoring, which justifies the standard specification. However, the implementation of such sensors in real-time applications encounters several practical challenges, in addition to the security challenges to be addressed by the protocols suggested in the security specification of IEEE 802.15.6, including but not limited to [21]:

- *Power management*: The standard specifies sleep/inactive stages for power management, and hence, resource optimization. However, in some healthcare applications, it might be challenging to include longer inactive stages for sensors. Examples include remote monitoring of high-risk individuals with ischaemic heart disease [22] or with suspected pregnancy-induced hypertension [23]. When monitoring such individuals, it is expected that their health be monitored in real-time, considering that although there might not be a need for immediate admission to healthcare facilities, regular monitoring and threshold-based alerts help prevent medical emergencies.

- *Sensor data validation*: Validating data recorded by the sensors is critical as errors could lead to false alarms. The responsibility of data validation must be shared by all sensors, the hub and the server at the monitoring station.

- *Interference and collisions*: Although the standard specifies the use of collision avoidance schemes, random access and contention mechanisms used by sensors to transmit data to the hub in practical applications will encounter collisions, especially in a sensor-rich environment.

- *Human-centric challenges*: The key challenge, perhaps, is "human-centric", relating to the adoption of WBAN applications by individuals. First and foremost challenge is the cost. The more the security and functionality, the more will be the cost, owing to the manufacturing and management costs. This is closely followed by ease of use. Perhaps another component adding to the cost, related to ease of use, is the ease of maintenance. Sensors are battery powered and it might be inconvenient for a user if the battery needs to be frequently replaced.

WBANs thus, can still be classified as resource-constrained, considering the various factors to be taken into consideration in the design of applications, and the potential trade-offs resulting therefrom.

2.3.5 Security in Resource-Constrained Wireless Networks: Summary

We have considered two resource-constrained wireless networks with varying security and application needs, both of which have three 'tiers' of devices, which we classify as — (a) *resource-constrained entities* (RFID tags and WBAN sensors), (b) *intermediaries* (RFID readers and WBAN personal servers), and (c) *powerful entities* (backend servers). While powerful entities are assumed to have all resources at their disposal, with no expected constraints, resource-constrained entities are restricted in their computational and storage abilities, as discussed earlier. For much of the communication, the intermediaries merely perform the action of relaying information from the powerful entities to the resource-constrained entities and vice-versa. This in many cases, leads researchers to consider that the entity is secure or assume that it is securely connected with the server [54]. However, with the communication channel being wireless in nature, one cannot assume any of the constituent entities as being ideal or secure. It has to be noted that the communication between these intermediaries and the servers could employ traditional symmetric or asymmetric cryptography, and is beyond the scope of our work.

The various restrictions and trade-offs in designing these systems leave them vulnerable to several attacks. These attacks can be broadly classified into physical layer attacks, network & transport layer attacks and application layer attacks [2, 3, 5], as per the various abstraction layers in a protocol stack as highlighted in the OSI model [86]. We summarize these attacks in Table 2.2 and describe them briefly in the paragraphs that follow.

- *Physical layer attacks*: These attacks involve interference to the physical devices (tags, sensors), to disrupt their communication. Examples include signal jamming or interference, RFID tag killing and sensor tampering.

- *Data link layer attacks*: These attacks involve unauthorized monitoring and manipulation of the communication channel by an adversary. A common attack applicable to RFID and WBANs is the collision attack, where an adversary transmits invalid data packets to collide with valid packets.

- *Network & Transport layer attacks*: These attacks are typically the case of the adversary exploiting the vulnerabilities in the communication between various

Table 2.2: Known Attack Classification [2, 3]

OSI Layer	Possible Attacks [2, 3]	Applicable System(s)
Physical	Jamming	RFID, WBAN
	Tampering	RFID, WBAN
Data link	Collision	RFID, WBAN
Network & Transport	Eavesdropping	RFID, WBAN
	Selective forwarding	RFID, WBAN
	Sybil	WBAN
	Sinkhole	WBAN
	Wormhole	WBAN
	Replay/Spoofing/altering	RFID, WBAN
	Reader impersonation	RFID
	De-synchronization	RFID, WBAN
	Hello Flooding/Flooding	RFID, WBAN
Application	Unauthorized reads	RFID
	Malicious code injection	RFID, WBAN

entities and the lack of strong security mechanisms. These include eavesdropping (adversary monitors all data transfer between verified entities), selective forwarding (adversary monitors and selectively forwards data packets), sybil attack (compromised node fabricates new node identities or impersonates valid existing node identities), sinkhole attack (compromised node circulates false routing information, forcing other nodes to select routes through the compromised nodes; this could lead to selective forwarding), wormhole attack (compromised nodes fake a shorter route to 'hijack' a packet, giving verified nodes an illusion of a shorter communication route), replay/spoofing/data alteration attack (adversary captures packets, spoofs verified entities and replays captured packets, with or without modification, to an authorized entity after some delay), reader impersonation (or, rogue readers), de-synchronization (adversary attempts to disrupt existing communication sessions to de-synchronize entities' states), and hello flooding/flooding (adversary causes resource exhaustion using *hello* packet or other packet flooding).

- *Application layer attacks*: These include attacks such as unauthorized reading of tags and sensor data, and data modification by adversaries on tags and sensors without (write) protection. These may also involve attacks such as buffer overflows, malicious code injection to the various components of the system.

- *Other attacks*: Typically, most of the common attacks on the resource-constrained systems such as RFID and WBANs are not on one layer alone. They usually encompass all the layers and affect the system. Such attacks include covert channels, denial of service, analysis of the patterns in the packet traffic, crypto attacks, side channel attacks, replay attacks, attacks on secrecy and authentication, etc.

Several research proposals have explored a variety of mechanisms to address the security challenges associated with deploying resource-constrained wireless network applications; some are even formalized by the specified standards. Rapid ongoing improvements in technology development could ultimately result in reducing the constraints on such devices, and the focus therefore, has to be on solving the problems keeping in mind the resource-constraints that exist today, but also ensuring that the proposals can be scaled should there be no such constraints in the days to come. In the next section, we elaborate on some of the contemporary proposals for key management and authentication in RFID systems and WBAN applications.

2.4 Related Work

The literature available on security in RFID and WBANs helps us categorize the research on the basis of the specific techniques used to accomplish key management and authentication as follows — (a) Key management based on pseudorandom number generation (similar to "rolling or hopping code [89, 90]"), (b) Pseudonym-based techniques, (c) Hash (or keyed hash)-based techniques, (d) Physical characteristics-based techniques, (e) Biometric techniques, (f) Certificateless techniques, (g) Symmetric encryption-based techniques, (i) Hybrid techniques, and (j) Frameworks.

2.4.1 Pseudorandom number generation-based techniques

Pseudorandom number generators (PRNGs) are a popular choice for cryptographic algorithms for key generation, since they generate unique sequences given different seeds, and can often operate for long periods without repeating sequences. Several PRNGs and their variants have been known to be secure for cryptographic applications [58, 60], and researchers are working towards adapting them for several RFID applications [59]. Due to the constraints of RFID tags, several lightweight PRNG algorithms are also in the works. Two such algorithms are proposed by Martin et al. [93]. Their implementations use non-linear filter functions to ensure dispersion of bits in the resultant random sequence. An essential aspect in design of PRNGs is the feedback polynomial, which decides the tap positions for feedback, in the linear feedback shift register (LFSR) implementations of PRNGs. An interesting approach has been adopted by Melià-Seguí et al. [94], where they use multiple polynomials and a "polynomial selector" to achieve unpredictability in the generated pseudorandom numbers. The polynomial selector is implemented in the manner of a "wheel", with the appropriate polynomial being selected by a true random number (TRN). The TRN is the output of a function based on physical characteristics such as thermal noise, etc. Their approach combines the true randomness provided by physical characteristics of a circuit with deterministic LFSR for security. It must be noted, however, that PRNG-based techniques help accomplish only key generation and management, requiring them to be combined with other techniques such as hash-based or trusted-third party-based approaches for authentication.

2.4.2 Pseudonym-based techniques

Resource-constrained security sometimes relies on the use of pseudonyms, or authorized changing of entity IDs, using either a synchronized PRNG or other techniques for generating pseudonyms. In an approach employing pseudonyms, Molnar et al. [95] introduce the concept of time-limited delegation for RFID tags, where a tag generates different pseudonyms, that enables a trusted server to authenticate the tag. The server delegates the responsibility of tag authentication to an authorized reader by giving it a set of pseudonyms that it can use to verify the tag for a specified amount of time. Their work employs the "tree of secrets" concept to choose (and verify) the appropriate pseudonym. The nodes of a binary secret tree has secret keys in each node, and each tag maintains a counter pointing to a leaf of the tree, which in turn represents a pseudonym. Therefore, a particular pseudonym can be used to represent one tag, depending on its present state and the pseudonym. The work by Juels [53] uses a concept called "pseudonym throttling", which is essentially a challenge-response scheme. In this, a tag stores a short list of pseudonyms, α_i, while the verifier (trusted server) authenticates itself by sending a key, β_i, in response to which, the tag responds with an authentication key, γ_i. β_i and γ_i are unique to each α_i. Being a challenge-response protocol, it relies on the ability of the verifier to update these values following successful mutual authentication. One-time pads in their scheme facilitate pseudonym updates, and these are directed by the verifier on successful authentication. Although pseudonym-based techniques facilitate mutual authentication, the pseudonyms are typically generated by the trusted server, and care needs to be ensured in the transfer of these pseudonyms to the resource-constrained entities over a wireless channel.

2.4.3 Hash (or Keyed Hash)-based techniques

Hashes, especially keyed hashes, in particular are popular ways of accomplishing authentication in symmetric algorithms. Dong et al.'s work on RFID authentication [9] employs the new SHA-3 standard (Keccak algorithm) [96] to compute the message digests using a concatenation of pseudorandom numbers, keys and the tag ID. Pseudorandom numbers are updated with each communication and are sent in the open, along with the hash containing an internally updated key. The key is updated on successful mutual authentication of the entities. The authors discuss various cases of operation, accounting for loss of tag acknowledgement messages and de-synchronization attempts. Hashing algorithms are also employed by Hakeem et al. [63] for authentication in their proposal, where they use timestamps for key generation. Their work relies on two separate timestamps, one each generated at the server/reader and the tag. Their work also employs a linear feedback shift register (LFSR) to update keys. The first part of the protocol depends on each entity authenticating the other based on the difference in timestamps between the previous acknowledged timestamp and the current timestamp, and the XOR value of this timestamp difference with the secret tag key, k_t. Tag authentication by the server involves the tag sending a hash of its ID and the upper half of the secret key, K. Key updates at the server and the tag involves updating two secret keys and the timestamp, where the keys are updated using the previous values as seeds to the LFSR, while the current timestamp becomes the new stored timestamp value at the tag.

2.4.4 Physical Characteristics-based techniques

Physical characteristics of RFID tags, such as electrical characteristics, are factors that can provide true randomness and uniqueness. This, along with deterministic algorithms, can help in achieving a high degree of security. The proposal by Choi et al. [97] capitalizes on the fact that a PUF (Physically Unclonable Function) may result in different outputs for the same input, depending on the nature of physical characteristics under consideration. Their approach pre-computes encrypted challenge-response pairs using the PUF-based encryption function, using random numbers generated by both the server and tag as inputs. They also present a protocol for authentication.

Jung and Jung's work [52] involves a scheme that uses keyed hash algorithms and physically unclonable functions (PUF) for mutual authentication between the server and the tag. The initial set up involves a challenge-response message exchange between the server and the tag, where the tag responds with a random number generated using the PUF output as the seed. Normal communication phases involve exchange of messages containing hash values computed using combined parameters such as the tag and server IDs, and tag and server timestamps. From the description, although it seems as if the tag timestamp, T_t, is generated at the tag, it must be noted that passive RFID tags are not always 'alive' or active, which makes it unreasonable to expect an always online clocking source on the tag. Also employing PUFs for authentication is Akgün et al.'s work [54]. Their scheme uses a temporary PUF value, x, that is computed during each communication phase for authentication. They use random numbers generated by the reader (assumed to be securely connected to the database) and the tag, in addition to the PUF, x, and the message, M_1, for authentication. M_1 is computed through pseudorandom number generation using the keys, $k'_{j,k}$ determined by the tag evaluating PUF with input S (the random seed). The security of this scheme lies in the fact that PUFs are employed to accomplish authentication. However, one thing to be noted is that the fact that a reader and/or server, which instructs the tag to perform specific tasks, such as internal state updates or instructing it to 'sleep' (rendering it ineffective), is never authenticated by the tag.

An alternative approach to using physical characteristics is presented by Huey et al. [51]. Their work uses the received power of the tag during the initial communication as the 'electronic fingerprint' for the tag. This electronic fingerprint is used in combination with the tag's electronic product code to compute a cyclic redundancy check (CRC) output, which is used as the primary authentication component. This is XOR-ed with a changing key, k_i, to generate the parameter, M_i, used by both tag and the server to authenticate the other entity. However, the CRC component remains constant throughout the communication, with only the key being updated. With the messages, M_i being XORs between a constant parameter and a changing key, their protocol remains vulnerable to de-synchronization attacks, and in determining the CRC component through analysis of a contiguous set of communicated messages.

Shi et al.'s work [98] exploits physical characteristics for security and (one-way) authentication in WBANs. Rather than having the sensors depending on cryptography for authentication, their work, BANA, considers using physical layer characteristics unique to the sensors; specifically, the variation in received signal strength (RSS) in the communication channel. The WBAN controller unit authenticates the on-body sensor nodes based on expected variations in received signal strengths of their individual responses and based on a threshold on the response time. The authors claim that attackers would experience "larger fluctuations due to multipath effect and Doppler spread than on-body sensors", making it a feasible authentication scheme. Mutual authentication among sensors or between sensors and the controller unit does seem to impact the limited resources, especially sensor battery life, in the long run, since BANA expects all sensors to compute the average RSS variations and authenticate other entities. This is mainly because authentication is an independent functionality in these sensors, which are required to include separate deployments of key management and encryption algorithms. Although the design of BANA is innovative in using physical channel characteristics for authentication, the need for separate implementation for key management implementation imposes an additional overhead on the resource-constrained sensors.

2.4.5 Biometric techniques

WBANs implicitly support biometric techniques for key generation and authentication, since they record and store physical data about an individual. The unique variation in most health parameters such as electrocardiogram (ECG) signals can be used to uniquely identify an individual, especially when the system is interfaced with a mechanism to extract key features of the recorded signals.

The (improved) fuzzy vault technique [99] uses a fuzzy extractor to extract “uniform randomness” from its input, with the rationale that small changes to the input will not cause the extracted output to be different, as long as the changed input remains “reasonably close to the original” [99] input. Zhang et al. [100] extend this work for application in WBANs, focusing on the “intercommunication and authentication between sensors” [100]. Their key generation is based on extracting feature (F) from the person’s ECG signal and generating a monic polynomial with root, F. Their work (ECG-IJS), like the improved fuzzy vault technique [99] is based on accomplishing security by retaining a subset of the coefficients of the monic polynomial a secret. ECG-IJS works on the assumption that the communicating sensors on the human body that employ this technique have the ability to extract ECG signals or are connected to ECG sensors. Authentication in ECG-IJS is accomplished through verification of the hash of the data, encryption key and the subset of the coefficients. Although fuzzy extraction promises increased security in normal WSN applications, when it comes to applying ECG-IJS in WBANs, one needs to consider the computational overheads added to the resource-constrained data collection sensors by the polynomial arithmetic and the additional overhead of attaching ECG sensors to each sensor on the body that collects data. This is especially true in cases that require continuous remote monitoring of a person’s health, as discussed earlier.

ECG signals are also used in the work by Venkatasubramanian et al. [101], primarily for generating encryption keys. In their work, sample values of ECG data are taken from a particular interval of the recorded ECG signal and coefficients are extracted using Fast Fourier Transform. Following this, a feature vector is generated using these coefficients and is used to generate the encryption key. The focus of their work is securing the inter-sensor communication within the BAN. Sensors in the network agree upon a common key generated using the ECG signal. While

Venkatasubramanian et al.'s work focuses on inter-sensor communication, Mana et al.'s work [102, 103] uses ECG signals to generate keys for encrypting data communicated among the sensors placed on the body, and between the sensors and the server at the monitoring station, as a way to accomplish end-to-end security.

While the works discussed in this subsection so far employ one biometric data recorded by the sensors, the work by Sampangi et al. [10] considers a WBAN with multiple sensors to record various parameters, such as heart rate, blood glucose level, etc. Their work involves the body central node sending data recorded by the various sensors in a single encrypted frame (with several fields to store data) to the monitoring station. The keys used for encryption are generated at both the sensors and the server by a random choice of one of the data fields from one of five previously acknowledged data frames, termed "reference frames". Reference frames are updated by replacing the oldest reference frame with the latest acknowledged data frame. This emphasizes data freshness, and uses the random choices of the frame fields and subtle variations in data in these fields (if any) to establish adaptive and changing keys for each communication, with the unique biometric data facilitating authentication.

2.4.6 Certificateless techniques

Message digests, digital signatures and third party certificates are common forms of accomplishing authentication among communicating entities. A different approach to accomplish this is a 'certificateless' manner, proposed by Liu et al. [55]. This is accomplished by a "public key generator" (PKG), an entity that computes and distributes public system parameters, including its public key. A signing entity generates its partial public and private keys, while using its identifier, *id*, as the other partial public key and requesting the corresponding partial secret key from the PKG. The certificateless signature generated by the signer includes the hash of the message, the output of an exponential function applied on the public key of the PKG and the signer, its partial secret key, and a random integer. Their authentication protocols assume the presence of a certificate authority (CA) and an environment that supports public key infrastructure. The protocols are designed to work at the application provider (typically the monitoring station) and the WBAN client (personal server). Both entities employ the certificateless signature mechanism to authenticate each other. One aspect to note is that with WBAN data updates being periodic, it is unclear whether this scheme is designed to authenticate entities on each update. This is because mutual authentication using public key infrastructure, regardless of how secure it is, will place an increased load on the already resource-constrained entity, whether it is a personal server or a sensor. While their approach appears to be difficult to be deployed on the sensors for authentication, their application to secure the communication between the personal server and the application provider would be reasonable, considering the current (and expected future) improvements to mobile devices.

2.4.7 Symmetric Encryption-based techniques

While symmetric encryption techniques such as AES are typically used to achieve confidentiality, Pham et al. [104] use it to accomplish a challenge-response mutual authentication scheme for RFID systems. Authentication in their approach is accomplished using pre-shared secret keys, and verification of ciphertexts using the current seed and entity ID. The pre-shared secret key used to encrypt all messages between the server, reader and the tag. The internal seed, s, in the tag and the server is updated on successful authentication, and used to ensure synchronization and future authentication. Perhaps the biggest assumption of their work is in considering that the ability of a passive RFID tag facilitates performing complex computations, including several rounds of encryption, as required in AES, while keeping the production costs low.

The work by Zhu and Khan [8] is a symmetric key management protocol, that supports authentication. It features a common key that is shared among the server and all RFID tags, and another key that is specific to each tag that is updated after successful authentication. Their scheme employs a block-wise key update mechanism, in which authentication is followed by updates to the 32 bit blocks (subkeys) of a 128 bit key, one at a time. The authentication involves the tag sending its encrypted ID combined (XOR) with the updated subkeys (message $m1$), and on successful authentication of the tag, the server responding with a combination of the ID and the updated subkey, encrypted by the new key (message $m2$). One thing to note is that their algorithm is prone to de-synchronization attacks, since the tag update is contingent on server authentication based on $m2$. If an adversary were to block $m2$ and transmit an unrelated $m2'$, the tag would not be able to authenticate the server, causing it to roll back its key update, thereby disrupting future communication.

2.4.8 Hybrid techniques

A key generation and authentication scheme using rotating 'carousel' is proposed by Kuroda et al. [105]. In their work on security in WBANs, communicating entities share the carousel structure, and in them store a value computed using the sensed data and random data. The final key is generated using a hash function on the value chosen at random from the carousel. After an initial verification phase, where the sensor sends the seed to the personal server (referred to as coordinator) for verification, the sensor updates its key using a (synchronized) random carousel rotation, and transmits the data encrypted by this key to the coordinator. The main drawback of this scheme is the one-way authentication of the sensor by the coordinator. While it might be reasonable to expect sensor nodes to be limited in their abilities, it would be essential to protect the actions they might take due to instructions by an unauthorized entity, especially when these sensors/actuators act based on and directly affect the health of the user.

Kovacevic et al. [56] choose a different approach to security in WBANs. Their scheme, referred to as LIRA (Light channel for sensor Initialization and Radio channel for Authentication), uses the visible light channel from a light source to establish secret keys and uses the wireless radio channel for authentication of the communicating entities. They use a multi-touch screen as the light source, employing on-off keying modulation to transmit the secret keys to the body sensors, which detect and recognize the key pattern using light detectors. The (mutual) authentication phase involves the body controller unit and the sensors exchanging pseudorandom numbers, node IDs and a keyed-hash (using the agreed secret key) of these parameters, and verifying each other. While LIRA is an innovative approach employing light channel to exchange secret key materials, this scheme adds an additional burden on the user to remove the sensors from their deployed locations on the body, place it on the light source, generate and exchange keys, and replace them on the body, which could be a significant inconvenience for users. Furthermore, the protocol requires that the secret keys be exchanged in the open, via a light source (e.g. a multi-touch screen tablet computer), which is a potential major vulnerability.

While public key infrastructure (PKI) might seem to be an improbable choice for deployment on the current generation of WBAN sensors without additional strain on available resources, a reasonable approach explored by Liu et al. [55], would be to use PKI between the personal server and the backend monitoring station. Such a combination has also been proposed by Drira et al. [106], with identity-based encryption (IBE) for communication between the mobile personal server and the monitoring station (extra-body communication), and symmetric cryptography employed for communication between the sensors and mobile personal server (intra-body communication). In their work, intra-body communication is enforced as a one-hop communication between individual sensors and the personal server. Authentication and key establishment takes place in two phases — (a) between mobile personal server and backend server, using IBE for authentication and Elliptic Curve Diffie-Hellman (ECDH) technique for key agreement, and (b) authentication of all three entities, and establishment of keys between sensors-mobile node, sensors-backend server and a three party key using ECDH. A similar approach has been adopted by Liu and Kwak [7], employing ECDH for key agreement and AES (Advanced Encryption Standard)-based techniques to accomplish authentication among the entities. These approaches are balanced between completely symmetric cryptography and completely asymmetric cryptography for security in WBANs. However, given that these require all entities including sensors to perform computations such as exponentiation and Elliptic Curve Cryptography point multiplication, we can expect that they will impose additional overhead to the already resource-constrained devices.

2.4.9 Frameworks

Our discussion up to this point has focused on individual algorithms and combinations thereof to accomplish key management and authentication in RFID systems and WBAN applications. In this section, we explore algorithm frameworks, or a collection of algorithms used to accomplish a single or several security goals.

A multi-algorithm encryption framework for active RFID tags has been proposed by Zhou et al. [110], an improvement of which is a generic optimized proposal for reconfigurable security co-processor work by Li et al. [111]. In their work, the control and data logic module chooses one of four encryption algorithms, namely AES, DES/3DES (Data Encryption Standard), RSA (Rivest-Shamir-Adleman) public key cryptosystem, and ECC (Elliptic Curve Cryptography)-based cryptosystem. Their work is deployed in an FPGA (Field Programable Gate Array)-based active RFID tag, where the design allows for reconfigurability and customization. The control/data logic module chooses the applicable encryption algorithm, in addition to the appropriate memory module, initializes the FPGA-based execution unit and performs the encryption. Their use of FPGA-based design is based on the reconfigurability rationale of the work by Jones et al. [112]. Jones et al. argue that a silicon-based implementation is not suggested for the design to be (re-)configurable. However, when we consider low resource devices such as passive RFID tags, one does not have any other option than implementing the custom security algorithms on silicon chips. Reconfigurability in such cases, can be accomplished by using hardware switches that can route data to the appropriate 'path' of the chosen algorithm for processing. This is the rationale we adopt in designing our framework for security.

The specified ISO/IEC 29167-1:2014 standard for RFID security [33] and IEEE 802.15.6 for WBANs [19] provide means for manufacturers to deploy multiple encryption algorithms on the devices as part of the respective security suites. From the available algorithms, the entities can select one for use for a particular session, during security association. When agreeing on the algorithm to be used, however, the choice is typically communicated in plaintext, available for an eavesdropper to learn about the system states. This reduces the overall uncertainty associated with the system. The approach we adopt, however, involves a random choice of one of the available algorithms, based on a previously agreed and synchronized timestamp, which increases the overall unpredictability and thus, the security of the system.

2.5 Summary

In this chapter, our focus was on recalling some of the general principles of information security and exploring how these apply to and are affected by specific requirements of resource-constrained wireless networks. Restrictions on utilization of the available resources on resource-constrained entities make these systems vulnerable to several attacks as explored in Section 2.3.5. Their vulnerabilities stem from the fact that design of such applications are based on trade-offs between application functionality, battery optimizations to ensure longevity (limiting RFID tags and WBAN sensors from staying 'alive' or active for longer times during communication), and security. There is no doubt that sophisticated cryptographic solutions, especially those included in the standard specifications discussed in Section 2.3.4, are extremely effective, as has been observed in other domains, however, the demands (and restrictions) of these systems most of which are dictated by the overall cost and size of the system, make such proposals infeasible. Our rationale behind choosing RFID (passive tag-based) systems and WBAN applications used for remote health monitoring of individuals for our work is that if security proposals are designed to as to increase security while consuming less resources in these systems, they can be scaled to other resource-constrained applications by slight modifications, depending on the needs of the particular system.

The research contributions discussed in this chapter emphasize the need for preserving privacy of individuals using these applications and some of the existing solutions. Though much of these existing proposals are designed for specific applications and the corresponding needs, in our thesis, we propose a generic modular, reconfigurable, multi-algorithm framework that addresses the needs of all resource-constrained systems. The novelty of our proposal is in its reconfigurability, both in terms of providing means to customize the security framework on a resource-constrained system and in its unpredictable choices of the constituent algorithms. To the best of our knowledge, ours is the first multi-algorithm, dynamically reconfigurable framework suggested for key management and authentication in resource-constrained wireless networks, such as RFID and WBANs, although a framework for encryption has been proposed by Zhou et al. in RFID systems [110]. Even though encryption algorithms are implicitly associated with key management and authentication algorithms, our framework is focused on key *generation* rather than exchange, does not depend on the servers to generate these parameters, and is independent of the encryption algorithms used. We discuss the motivation behind our work and the research objectives in the next chapter. Following this, we present our framework in the next chapter, followed by a description of each of its constituent algorithms.

Chapter 3

Research Focus

3.1 Motivation

As discussed in Chapter 2, resource-constrained wireless networks typically have at least one entity without constraints, a.k.a. *the powerful entity*, which performs much of the sophisticated computations and behaves as a *master* node. In RFID systems, the backend server is the powerful entity, while its counterpart in WBANs is the monitoring station or the base station. In such architectures, it is common for the master node to send much of the information such as encryption keys and authentication parameters (we refer to this process as security association) to the resource-constrained entities, such as the RFID tags or the WBAN sensors. In both cases, there is an intermediary entity, a.k.a. *the middle man*, which relays information from the resource-constrained entities to the powerful entity, and vice-versa. The RFID reader dons this role in RFID systems, while the cellular phone (or laptop computer) personal server performs the same function in WBANs.

To summarize, resource-constrained wireless networks have security associations over the air, an intermediary to relay information back and forth on the system, and computationally limited entities such as tags or sensors.

This being the case, interception of such security associations by unauthorized agents could leave the system vulnerable to attacks including, but not limited to — data modification, entity tracking (that also constitutes an intrusion of privacy), or disruption in service, which could ultimately lead to information capture, session hijacking, selective forwarding, etc. These attacks hamper the normal operation of such systems and would lead to financial, informational losses, and potentially, health risks in case of WBANs.

A comprehensive security technique for any system would include all constituent entities having the ability to authenticate their peers, and have sufficient computational abilities to be able to generate encryption parameters independently, rather than having to rely on the ‘powerful entities’ to perform computations for them. In other words, a secure system is comprised of computationally independent entities, responsible for their own security.

3.2 Research Objectives

In our thesis, we propose a new security framework for resource-constrained wireless networks, with a focus on the following objectives:

- *Reconfigurability*: The proposed framework will have the logic to dynamically and automatically choose one of the available algorithms for key generation and mutual authentication;
- *Modularity*: The proposed framework will be composed of modular security mechanisms that work independently as standalone systems, as well as collectively, bound by the proposed framework. This modularity will also facilitate reconfigurability and reduced resource utility due to re-use;
- *Independent (and dynamic) key generation*: The constituent security modules will have independent key generation and management logic, which prevents the need for key exchange messages, and facilitates dynamic key generation. Key generation will be dynamic in that the keys will be generated based on the current conditions (determined by parameters such as timestamp);
- *Mutual authentication*: The key generation logic in the constituent security modules will facilitate generation of authentication parameters that help entities validate the authenticity of each other;
- *Deterministic, with unpredictability*: The proposed framework (and constituent modules) will work in a way to keep the system deterministic (to facilitate mutual authentication), and ensure high unpredictability owing to changing states (to ensure higher security).

3.3 Hypotheses

The main objective in our thesis is to create an environment that facilitates higher security independence to resource-constrained entities, while removing the need for exchange of key generation and authentication parameters. This is in addition to having a modular-reconfigurable framework as a foundation, with high unpredictability to ensure security and being computationally simple to ensure applicability to resource-constrained entities. The state-of-the-art emphasizes separate communication phases for key agreement and/or selection of one of the algorithms from a suite of algorithms, necessitates entities to perform complex computations, or requires the resource-constrained entities to depend on the backend server for generation of encryption keys/authentication parameters. This adds a communication overhead in an already constrained application environment. Our framework (discussed in the next chapter) includes a mechanism to automatically (and depending on the context) choose one of the available algorithms. The algorithms that are proposed as a part of our framework (discussed in Chapter 5) are designed in a way to ensure continuous key updates, synchronized with the server. The design of the framework and the algorithms is simple, in that the operations performed by entities would include bitwise logical operations and simple arithmetic operations.

This leads us to hypothesize that:

H1: *Algorithm choice*: Choice of the algorithm in the proposed framework used will be (pseudo)random and context-dependent, with each constituent algorithm being equi-probable, i.e. having equal probability of being chosen in a large number of trials.

H2: *Dynamic key generation*: Encryption keys generated by the proposed framework and the constituent algorithms will be updated with every communicated message.

H3: *Key unpredictability*: Encryption keys generated by the constituent algorithms (and hence, the framework when it employs them) will be unpredictable to an unauthorized entity.

H4: *Attack detection*: All algorithms in the proposed framework will be able to detect attacks, including replay attacks, data modification, and multiple requests that will constitute Denial-of-Service (DoS) by overwhelming the entity.

H5: *Resource utilization*: The proposed algorithms (and hence, the framework when it employs them) will be efficient in their utilization of the available memory and logical resources on a resource-constrained device.

3.4 Summary

In this chapter, we discussed the motivation for our work, as well as the overall objectives of our research, followed by listing the hypotheses that we are testing in our research. In the following chapters, we discuss our proposed framework in detail and present a description of each of its constituent modules, which form the foundation towards accomplishing the objectives stated in this chapter. We will then revisit our hypotheses in Chapter 6, where we also describe techniques to evaluate them and discuss expected results.

Chapter 4

Proposed Framework

4.1 Overview

The main concept in our work is a metamorphic framework for choosing appropriate algorithms for security in resource-constrained wireless networks. Inspired by the functioning of a chameleon, which changes its colour based on the colour of its surroundings, our framework is aimed at choosing an algorithm at random from the set of available algorithms. Although our framework might not be direct in its adaptation of the mechanism by which a chameleon changes its colour, it relies on a (pseudo)random choice of an algorithm dependent on the context of communication, i.e. based on a combination of the entity ID, the time of the communication (timestamp) and an incrementing number that is specific to the entity. We introduce our framework in this chapter.

4.2 Proposed Framework

Let us consider a conventional scenario with a system having one pre-defined algorithm for each aspect of security. Most systems use such an architecture and this works when all parameters other than encryption keys are pre-defined. In such cases, the uncertainty of the system operation remains limited. However, algorithms such as IPSec [86] (and the ISO/IEC 29167-1:2014 and IEEE 802.15.6 standard specifications [33, 66]) are considerably better in the security that they offer than the former, with the entities choosing one of the pre-agreed algorithms just as the communication session begins. We derive motivation for our proposed framework from this aspect of being able to change algorithms, and dynamically so, while removing the need for explicit agreements between entities, which would normally require an algorithm agreement message exchange prior to the session.

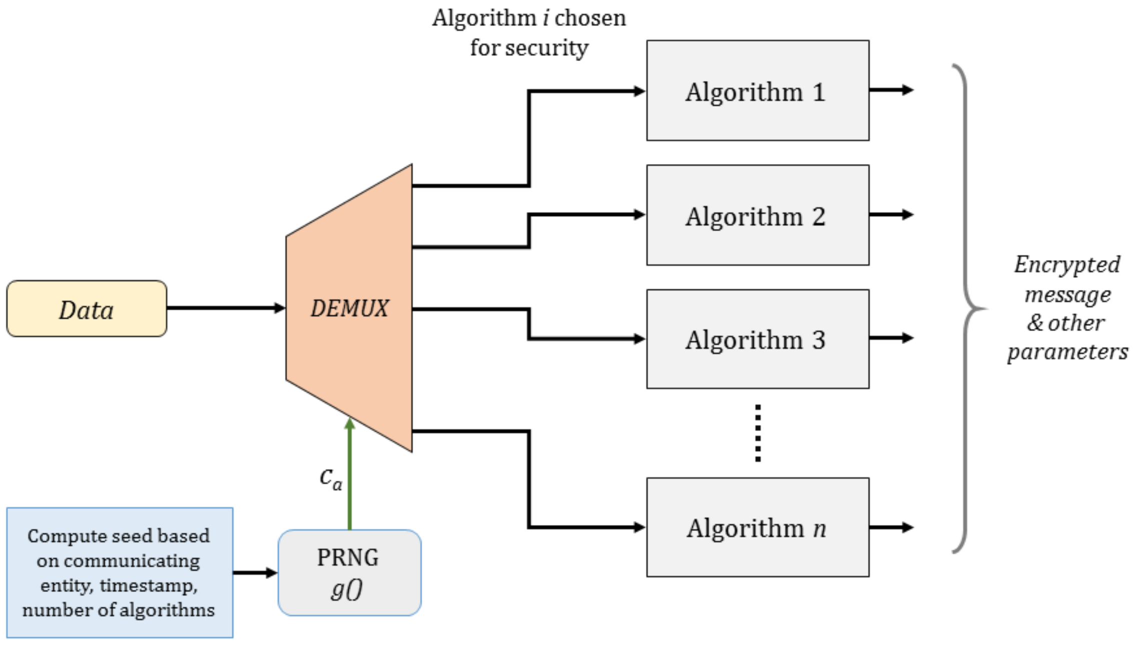

Algorithm i chosen for security
Algorithm 1
Algorithm 2
Algorithm 3
Algorithm n
Data
DEMUX
c_a
Compute seed based on communicating entity, timestamp, number of algorithms
PRNG $g()$
Encrypted message & other parameters

Figure 4.1: Overview of the Multi-Algorithm Framework

Figure 4.1 illustrates the overview of our proposed framework. Imagine a situation in which the system has n_{Alg} encryption schemes, each being a composite of algorithms to accomplish key management, encryption and authentication. Central to our framework is a mechanism to choose one of the available algorithms automatically and in a synchronous manner. We refer to this as the *algorithm choice logic*. This logic uses a unique combination of the ID (identifier) of the resource-constrained entity, the initial deploy-time timestamp (t_0) and an incrementing integer number, r_{ac}, in the range $0...(n_{Alg} - 1)$, to determine which of the n_{Alg} schemes will be chosen to generate keys for data encryption and generate authentication parameters for a particular message transfer. The integer r_{ac} has a modulus of n_{Alg}, i.e. it 'wraps around' on reaching n_{Alg} (Equation (4.1)).

$$r_{ac} = (r_{ac} \ + \ 1) \ mod \ n_{Alg} \tag{4.1}$$

The choice aspect of the algorithm choice logic is accomplished by a pseudorandom number generator, $g()$, that uses a combination of the ID, t_0 and r_{ac} as the seed. This seed, $seed_{ac}$, is generated as summarized by Equation (4.2).

$$seed_{ac} = ID \ \oplus \ t_0 \ \oplus \ r_{ac} \tag{4.2}$$

Here, ID is the identification number associated with the resource-constrained entity (e.g. RFID tag or WBAN sensor); t_0 is the deploy-time timestamp; and, r_{ac} is the incrementing integer number, whose computation is explained by Equation (4.1). $\oplus$ represents the Exclusive-OR (XOR) operation.

The chosen algorithm, C_a, is determined by generating a pseudorandom number using $seed_{ac}$ as the seed for $g()$. The number generated has a modulus of n_{Alg}.

$$C_a = g(seed_{ac})\ mod\ n_{Alg} \tag{4.3}$$

The deploy-time timestamp, t_0, is the timestamp that is stored on the resource-constrained entity just prior to deploying it in its application environment. This is among the first pre-shared attributes, along with the ID and the initial encryption parameters associated with each algorithm. During the course of operation in two of the proposed constituent algorithms, i.e. Butterfly1 and HiveSec1, the initial timestamp, t_0, is updated at random. This will ensure that the system state remains unpredictable to an observer and will be discussed further in the corresponding chapter. We employ the same timestamp in the algorithm chooser logic to re-use the stored and synchronized data, and to capitalize on the added uncertainty it provides. This specific combination of numbers, i.e. the timestamp, t_0, the incrementing number, r_{ac}, and the ID, changes continuously owing to increments in r_{ac} and at random with changes to t_0. This ensures that the n_{Alg} algorithms have a fair chance in being chosen for a specific encryption cycle.

To better explain this scenario, let us consider that there are 3 algorithms ($n_{Alg} = 3$), the ID is 36 (hexadecimal), and $t_0 = 17$ (hexadecimal). Table 4.1 summarizes possibilities of how the seeds change, thus, changing the pseudorandom number that is generated, and hence the algorithm chosen.

The operation of the seed generation and the corresponding pseudorandom number generation are illustrated by Figure 4.2. We can imagine the function of the algorithm choice logic in Figure 4.1 to be realized using an $n - to - 1$ Demultiplexer, with C_a being used as the *select* input to choose one of the n_{Alg} algorithms that will be used to generate keys for message encryption and generate parameters for authentication. Note that a demultiplexer is a logic circuit entity that is used to drive the input to one of the n_{Alg} available output ports [113].

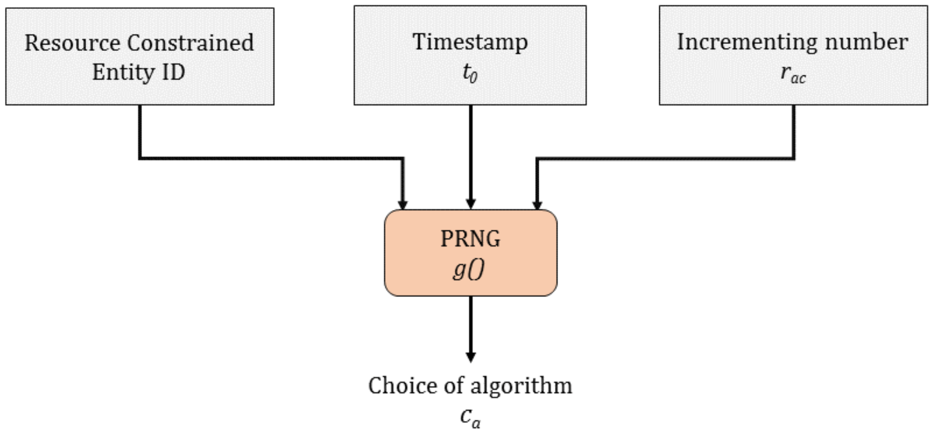

Resource Constrained Entity ID
Timestamp t_0
Incrementing number r_{ac}
PRNG $g()$
Choice of algorithm c_a

Figure 4.2: Illustration of Algorithm Choice Process

Our framework can be tweaked to include more (or reduced) choice in algorithms, depending on the application needs and the extent of constraints on the available resources. Thus, our framework has an implicit support to scalability, with minimal changes necessary to accommodate more algorithms in the framework. The changes would be in updating the algorithm chooser logic, specifically by updating n_{Alg} and a possible change to the circuit to extract C_a using modulo operation. Algorithm 1 summarizes the working of the algorithm choice logic in our framework (for a case when number of algorithms, $n_{Alg} = 3$).

4.3 Summary

In this chapter, we proposed a new framework to employ multiple algorithms to accomplish key generation and authentication on a resource-constrained device, and a

Table 4.1: Example Illustrating Generation of Algorithm Choice (C_a)

ID (bits)	r_{ac} (bits)	t_s0 (bits)	$seed_{ac}$	C_a
36 (00110110)	0 (00000000)	17 (00010111)	21	$g(21)$ mod 3
36 (00110110)	1 (00000001)	17 (00010111)	20	$g(20)$ mod 3
36 (00110110)	2 (00000010)	17 (00010111)	23	$g(23)$ mod 3
36 (00110110)	0 (00000000)	17 (00010111)	21	$g(21)$ mod 3
36 (00110110)	1 (00000001)	29 (00101001)	1E	$g(1E)$ mod 3
36 (00110110)	2 (00000010)	49 (01001001)	7E	$g(7E)$ mod 3

Algorithm 1 Algorithm Choice

```
1:  procedure ChooseAlgorithm
2:      $rac \leftarrow r_{ac}$
3:      $t0 \leftarrow t_0$
4:      $nAlg \leftarrow 3$
5:      //Generate seed:
6:      $rac \leftarrow (rac + 1)\ mod\ nAlg$
7:      $seedAC \leftarrow rac \oplus t0 \oplus ID$
8:      $CA \leftarrow g(\ seedAC\ )\ mod\ nAlg$
9:      //Choose algorithm:
10:     if $CA = 0$ then
11:         //Algorithm chosen = algorithm 1
12:     else if $CA = 1$ then
13:         //Algorithm chosen = algorithm 2
14:     else if $CA = 2$ then
15:         //Algorithm chosen = algorithm 3
```

mechanism to choose one out of the n_{Alg} algorithms that might be deployed. This is an attempt to include uncertainty at an algorithm choice level, which when combined with varying internal states of each algorithm, would make the system more secure owing to the increased unpredictability. Our framework is reconfigurable, in that it has the ability to dynamically choose one of the available algorithms to secure each communicated message, or it has the ability to change its structure for each message. Furthermore, this metamorphic behavior is deterministic for a synchronized and authenticated entity, however remains unpredictable for an unauthorized observer.

As discussed in Chapter 2, the main focus of our framework is key management and mutual authentication, with the generated keys and authentication parameters being compatible to work with any symmetric encryption algorithm. To accomplish this, we propose three algorithms for key management and authentication in Chapter 5, each discussed as independent algorithms designed to work with encryption and hash algorithms as part of complete encryption schemes. The primary goals of these algorithms are modularity and re-usability, which will ensure simplicity, while guaranteeing high unpredictability. In Chapter 6, we summarize as to how these algorithms will work as part of our framework discussed in this chapter.

Chapter 5

Proposed Component Algorithms

5.1 Chapter Overview

In this chapter, we first introduce some of the preliminary concepts of our work, including the concept of *Biomimetics*, which is one of the foundational concepts of our work, before presenting each constituent algorithm of the metamorphic framework proposed in Chapter 4. The first algorithm, GeM1 [11], was proposed as a key management and mutual authentication algorithm based on gene mutation and transfer. Owing to certain limitations of GeM1, we improved the algorithm and present it in this chapter as GeM2 [12]. Following this, we propose a new way to update pseudo-random number generator (PRNG) seeds, naïvely based on the Butterfly effect from Chaos theory, and present an encryption scheme based on the same. We refer to this as Butterfly1 [13]. Finally, we propose a new algorithm for multiple aspects of security called HiveSec1, which is inspired by the functioning of beehives and bee swarms. With our GeM2 and Butterfly1 approaches initially proposed for RFID systems, we discuss extending them for use in WBAN systems in Section 5.6.

5.2 Preliminaries

5.2.1 Biomimetics

Systems we observe in nature are independent in existence and functionally complete. Man made objects are only used to assist one such complete and independent system, a.k.a. humans, in the functions that we perform. In other words, the systems we create need to 'mimic' our own abilities to carry out certain functions. Extending this concept to our cohabitants on this planet, we can derive inspirations from other natural systems/processes to create better systems to assist us. This process of adopting biological processes, such as working principle of a neuron, ascent of sap in plants and so on, in fields such as engineering, robotics, electronics, etc. to create new systems is referred to as *Biomimetics* [114], which literally translates to mimicking biological concepts.

Biomimetics has been used to create many systems that we use in our daily lives, such as aircraft — Leonardo da Vinci based his designs of flying machines on the ability of birds to fly, Velcro — whose design was derived from the hooked seeds of the burdock plant, anti-reflective surfaces — which are created using polythene sheets mimicking insect eyes, wings and leaves of plants. These are only a few examples chosen from a variety of other solutions to which we have grown accustomed.

No formal framework exists to specifically use biological concepts in other disciplines. The basic principle in creating such bio-mimicking systems is to carefully observe and understand the concepts and draw inspirations for designing appropriate systems.

In our work, we draw inspiration from natural processes to accomplish specific functions to generate keys and authentication parameters. In GeM2, we draw inspiration from the transfer of genes and their possible mutation in key generation, while our HiveSec1 proposal is inspired by the symmetrical hexagonal structure of beehives and on bee swarms. Furthermore, the working of our proposal is inspired by the manner in which a chameleon changes its colour depending on the context, i.e. its surroundings.

5.2.2 State Identifiers

A (deterministic) PRNG is analogous to a Finite State Machine; particularly, the Mealy Sequential Machine [113]. In this, the system output depends on the present state and the present inputs. This is appropriate since the output of a PRNG at any instant depends on the seed that is input and the pseudorandom sequence that it has generated until that instant. However, it can be argued that it is similar to a Moore Sequential Machine (in which the output only depends on the current state) if the seed is considered to be a constant.

The unpredictability of a deterministic system can be increased by dynamically demanding the system to generate the output from a specific state as desired. The unpredictability further increases when we change the input to the system, in addition to demanding the output from a specific state. This way, it will make it harder for an observer, with partial knowledge about the system, to be able to determine the exact output of the next system state. This is particularly useful in cryptography, where it is desired that the adversary not be able to decode the parameters associated with a particular state of the cryptosystem.

In our approach, we employ numeric state identifiers. The initial state, say S_0, is "0", the next state, S_1 is "1", and so on. In our approach, we use state identifiers to help in generating encryption keys on demand and message signatures. Use of state identifiers helps entities using our algorithms be able to generate synchronized parameters that can be used for key generation, message integrity verification and mutual authentication.

5.3 GeM2: Improved Key Generation and Mutual Authentication Algorithm based on Gene Transfer and Genetic Mutation

5.3.1 Overview

As with any computer program, characteristics or attributes of living organisms are functions that are decided by instructions. In biology, such instructions translate to the genetic code or genetic instructions that are contained in a basic molecular structure called deoxyribonucleic acid (DNA). DNA contains genetic information that help an organism realize its functions such as development, digestion and so on. When organisms procreate, this genetic information is passed on from the parent to the progeny. Therefore, DNA plays a significant role in continuity of species and in preserving characteristics of each species. This phenomenon by which genetic information is transferred from one generation to the next is referred to as gene transmission [115, 116].

During such transmission, factors in the environment internal or external to the organism could result alterations to these genes, and such alterations could be passed on to subsequent generations. Such alterations, which could be evolutionary or abrupt, are called mutations. Thus, in general, characteristics of a generation of any organism are transferred from one generation to another by means of gene transmission and the transferred characteristics may include mutations as well.

GeM1 [11] is a protocol that mimics the concept of 'generations' and genetic

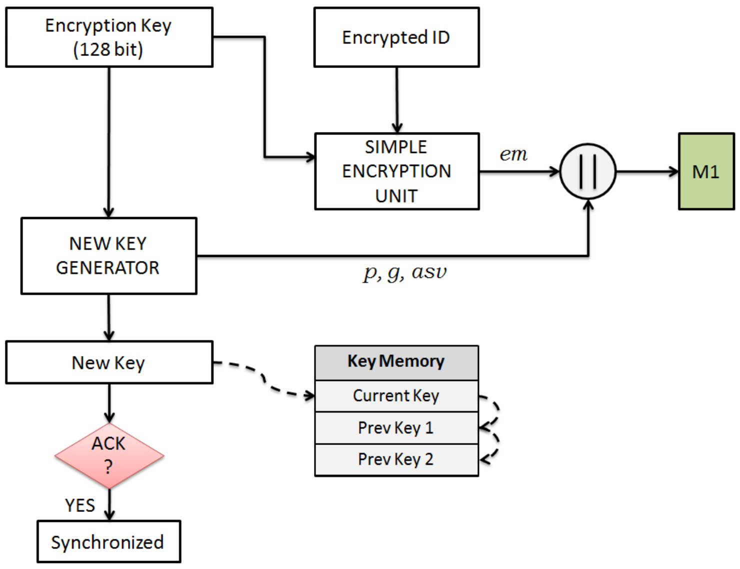


Figure 5.1: Overview of GeM1 [11]

mutation. This proposal was intended for RFID systems, specifically on lightweight or heavyweight that tags have the ability to perform minimally complex computations such as one-way hash function, and have sufficient storage capacity. We discuss GeM1 in the next section.

5.3.2 GeM1: The Predecessor of GeM2

In GeM1, our focus was to create a new algorithm that focused on eliminating key exchange messages, and providing independent linked key updates using current components such as PRNGs in a way that would make the system state unpredictable for an adversary, yet keep it deterministic for the communicating entities to authenticate each other and synchronize their states.

GeM1 requires communicating entities (Alice, the tag and Bob, the server) to pre-share a '*parent*' key, referred to as the initial key, IK. During their communication, this key is used to compute "new keys", NK, or '*generations*'. The *parent* and *generation* numbers specify the state of the system, implying that given an initial key and specific values of *parent* and *generation*, an entity could generate the appropriate parameters. Figure 5.1 illustrates an overview of the operation of GeM1.

Let us begin our discussion of GeM1 with state i considered to be the current synchronized state of Alice and Bob. The communication begins with a reader querying Alice. On reception of this query, Alice performs the following operations:

Step-1: Alice uses the currently synchronized encryption key, k_i, to encrypt its encrypted ID, ID. We choose to further encrypt an already encrypted ID stored on the tag to add another layer of security. Thus, it results in the encrypted message, em, given by Equation (5.1).

$$em = E_{k_i}\ (\ ID\ ) \tag{5.1}$$

Step-2: Alice then updates her key using a concept mimicking gene mutation and transfer [11], where the currently synchronized key is considered as the parent key and changes, or 'mutations', are introduced to the bit patterns in a way to preserve the parent key patterns yet change the pattern strategically. This means that the bit pattern of the parent key is masked and possible changes introduced only to those bits that were '0' in the parent key. The mutation is computed as summarized by Equation (5.2).

$$mutation = \ numX \bullet \ \neg k_i \tag{5.2}$$

Here, $\neg k_i$ is the logical inverse of the current key, k_i, generated in an attempt to preserve the current key patterns. $numX$ is the first pseudorandom sequence, respectively, generated by the PRNG, $g()$, with a specific seed, sd_i, as summarized by Equation (5.3).

$$numX = g\ (\ sd_i\ )_{first\_128bit\_sequence} \tag{5.3}$$

The seeds, sd_i, used are Fibonacci numbers [117]. The initial seeds are loaded (and synchronized) on the entities at deploy time, and their update is by adding the two previous seeds to generate the next, i.e. as defined by the linear recurrence equation with the polynomial coefficients being 1 [118]. The seed update is defined by Equation (5.4). We use this seed update mechanism to introduce an additional layer of uncertainty in our approach.

$$sd_i = sd_{i-1} + sd_{i-2} \tag{5.4}$$

Step-3: A PRNG is used to introduce these changes. If the changes introduced result in a key, k_X, that is the same as a pre-stored pattern called *genLimit*, then, the system forces an 'evolution' of keys by changing the parent key. Thus, it forces a refresh in the key generation cycle. This process is summarized by Equation (5.5).

$$k_{i+1} = \begin{cases} k_X & \text{if } k_X \neq genLimit \\ k_Y & \text{if } k_X = genLimit \end{cases} \tag{5.5}$$

Here, k_X and k_Y are intermediate keys that are generated based on Equations (5.6) and (5.7).

$$k_X = k_i \ \oplus \ mutation \tag{5.6}$$

$$k_Y = numY \bullet \ k_X \tag{5.7}$$

$numY$ is the second pseudorandom sequence generated by the PRNG, $g()$, with a specific seed, sd_i, as summarized by Equation (5.8). Note that the first PRNG sequence is $numX$ (Equation (5.3)).

$$numY = g \ (\ sd_i \ )_{second\_128bit\_sequence} \tag{5.8}$$

Step-4: Next, the algorithm generates the authentication-synchronization vector, asv_i, by using the $numX$ pattern and a pre-shared pattern called the $pattern_{asv}$, as explained in Equation (5.9). Following this, it updates the seed, sd_i.

$$asv_i = h \ (\ numX \ \oplus \ pattern_{asv} \ ) \tag{5.9}$$

Step-5: To maintain a local record of the state of the system, Alice and Bob maintain two variables, p and g, to represent *parent* and *generation* states, respectively. In case the key is updated, g is incremented by '1', while in case of an 'evolution' i.e. change in parent key, p is incremented and g is reset to '0'.

After the above processes, Alice responds to Bob with the message, $M1$, summarized in Equation (5.10).

$$M1 = \ p \parallel g \parallel em \parallel tagSignature \tag{5.10}$$

Where, $\parallel$ represents concatenation operation and asv_i is sent as the $tagSignature$.

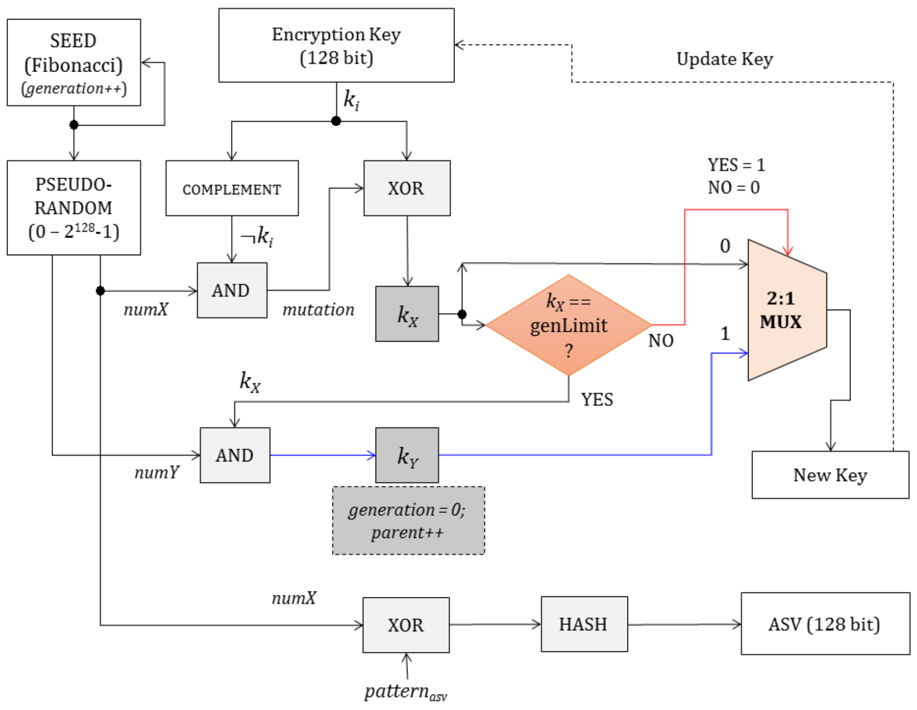


Figure 5.2: Key Generation in GeM1 [11]

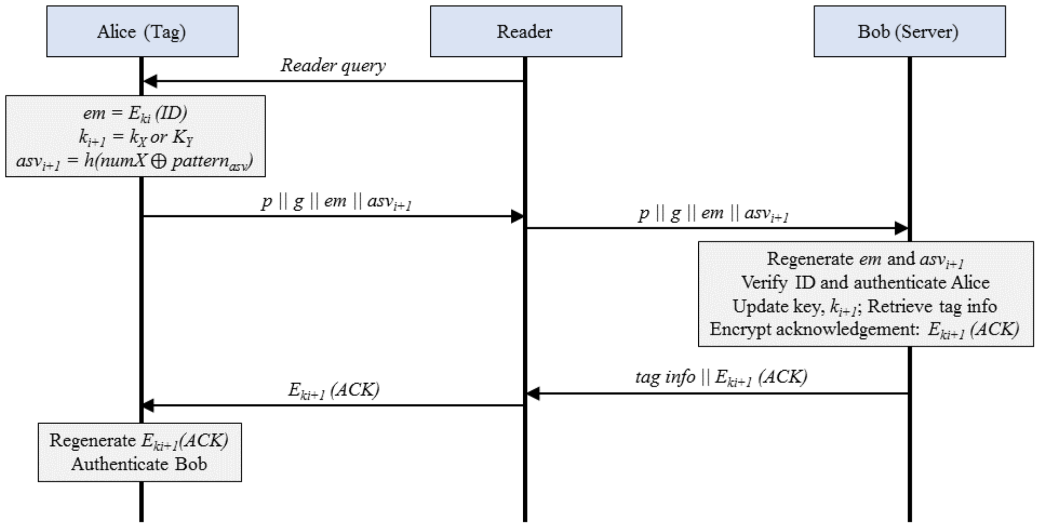

Alice (Tag)
Reader
Bob (Server)
Reader query
$em = E_{ki}(ID)$
$k_{i+1} = k_X$ or K_Y
$asv_{i+1} = h(numX \oplus pattern_{asv})$
$p \,||\, g \,||\, em \,||\, asv_{i+1}$
$p \,||\, g \,||\, em \,||\, asv_{i+1}$
Regenerate em and asv_{i+1}
Verify ID and authenticate Alice
Update key, k_{i+1}; Retrieve tag info
Encrypt acknowledgement: $E_{ki+1}(ACK)$
tag info $||\, E_{ki+1}(ACK)$
$E_{ki+1}(ACK)$
Regenerate $E_{ki+1}(ACK)$
Authenticate Bob

Figure 5.3: Communication Protocol in GeM1 [12]

On receiving this message, Bob first short-lists candidate tags based on the p and g values. Next, Bob generates the key using the p and g values, and the current states of the chosen candidate tags. It then uses the keys to decrypt the encrypted message, em. At this point, Bob is able to verify the tag using the encrypted tag ID. Bob then verifies the new state after updating the key using *tagSignature*. Following this, Bob generates an acknowledgement message, M_A, comprising of an acknowledgement pattern encrypted using the new key, k_{i+1}, and sends it along with the information about the object represented by Alice to the reader. The reader retains the information about the object and forwards the acknowledgement message, M_A, to Alice, who uses this to validate Bob.

Thus, GeM1 facilitates mutual authentication and dynamic-independent key generation at the communicating entities. The key generation process in GeM1 is illustrated in Figure 5.2 and the communication protocol is summarized in Figure 5.3.

5.3.3 GeM2: Improved Key Generation and Mutual Authentication Algorithm

Our evaluation of GeM1 proved that the algorithm was secure and was able to offer better security through increased unpredictability. However, since keys were linked and *genLimit* was set to a pattern of all '1' bits, it proved to have a concealed vulnerability. It meant that we were allowing the keys to grow until (or converge at) a point when all bits of the keys (or most bits of a key) were '1' to force an 'evolution' of the parent key, introducing a predictable pattern in an otherwise unpredictable environment.

As an attempt to overcome this flaw, we worked on modifying the linking between keys and on changing the way parent keys evolve. Essentially, we accomplished this by setting generation limit as the integer count of the number of 'child' keys a 'parent' key can have, i.e. limiting that a parent key can have up to a maximum of *genLimit* child keys. The value of *genLimit* can be set based on the needs of the application. It is the maximum number of child keys that can be generated using a parent key, keeping in mind that the more the number of child keys for a parent, the likelihood of key repetition increases (in our implementation and assessment, we set $genLimit = 5$, given that there are 128 bits in the keys and with the expectation that the likelihood of key repetition within a *genLimit* of 5 would be minimum). This was instead of the mechanism in GeM1 to let the keys evolve/converge towards a specific pattern. Figure 5.4 illustrates the operations involved in the key generation process in GeM2.

GeM2 works mostly the same as GeM1, however, the first important change is

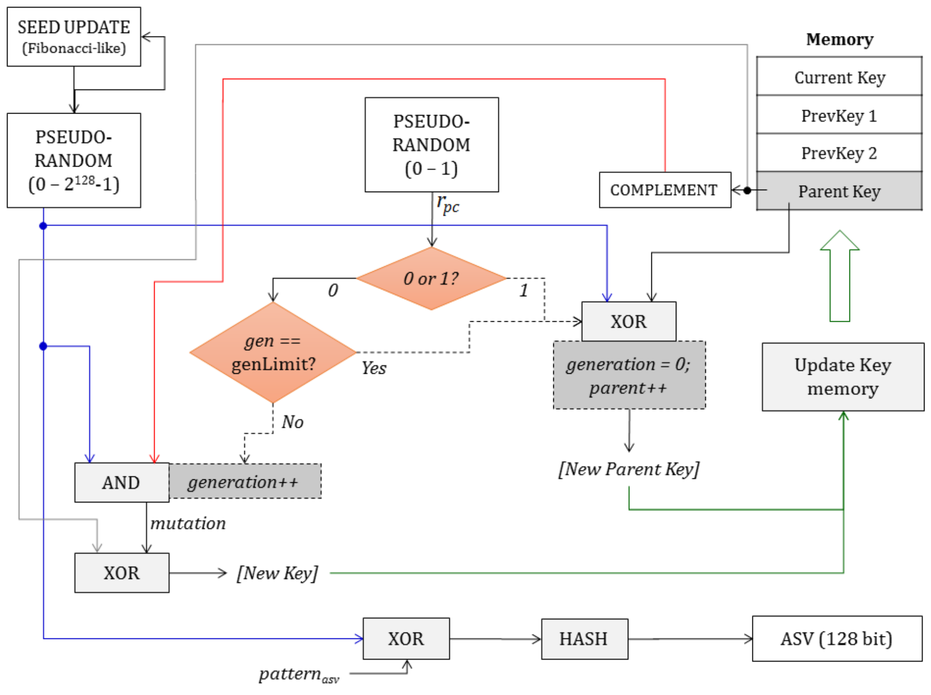


Figure 5.4: Key Generation in GeM2 [12]

in the addition of a new memory segment called $parentKey$ to store the parent key. This is necessary because in GeM2, a parent can have a limited number of child keys or no child key at all, but if it does have child keys, it can have up to $genLimit$ child keys. This defines its improvement over GeM1 — we do not have linked "generations" of keys. In the context of GeM1, linked generations of keys was the manner in which keys were updated, i.e. each new key became the parent key of the subsequent key, which led to their evolutions converging at a pattern of all bits being '1'.

In a state i of communication between entities Alice and Bob, GeM2 works as follows. Note that Alice uses the currently synchronized encryption key, k_i, to encrypt its encrypted ID, ID, to generate the encrypted message, em (similar to GeM1, Equation (5.1)).

Key update

The key generation module first generates a random number (r_{pc}) between '0' and '1' to determine if current parent key will have any further child keys (Equation (5.11)).

$$r_{pc} = \begin{cases} 0 & \text{parent can have up to } genLimit \text{ child keys} \\ 1 & \text{parent key update, i.e. evolution} \end{cases} \tag{5.11}$$

If $r_{pc} = 0$:

- If the current generation count, g, is not equal to *genLimit*, it generates a new key by first preserving the parent key pattern by generating its inverse ($\neg k_i$) and creating a mutation pattern from it. The mutation pattern is created by using the inverse of the key and logically AND-ing it with the random number (as explained in Equation (5.12)).

$$mutation = numX \bullet \neg parent\_key \tag{5.12}$$

- This mutation pattern is applied to the current parent key by XOR operation to generate the new key, $K = k_{i+1}$ (as summarized by Equation (5.13)), and generation count, g, is incremented by 1.

$$K = k_{i+1} = parent\_key \oplus mutation \tag{5.13}$$

  Here, $\oplus$ represents XOR operation between the parent key and the mutation pattern.

- However, the current generation count, g, being equal to *genLimit* forces a parent key change or refresh or what we refer to in the current context as an 'evolution' of the parent key, as discussed next.

If $r_{pc} = 1$ or if the generation count is equal to *genLimit*, the algorithm goes through a parent key refresh. To generate a new parent key, GeM2 generates a new random number, $numX$, and combines it with the current parent key using the XOR operation. This is summarized by Equation (5.14). In this case, the parent key also becomes the new key, K.

$$K = parentKey_{new} = parentKey_{current} \oplus numX \tag{5.14}$$

$numX = g(sd_i)$, is the random number generated by PRNG, $g()$, with seed sd_i, as explained by Equation (5.3).

The authentication vector is then generated in a manner similar to GeM1 (Equation (5.9)). Following this, the algorithm updates the seed, sd_i.

Seed Updates

One of the important aspects in this algorithm is the updates to PRNG seeds with each key generation. We accomplish this using a 'Fibonacci-like' seed update mechanism, as defined by the linear recurrence equation with the polynomial coefficients being 1 [118]. This is governed by Equation (5.4). Initially, Fibonacci numbers were chosen for use as the seed generation algorithm for pseudorandom number generation in GeM1, owing to their ability to grow at an exponential rate. In GeM2, we modified this to be a simple application of the linear recurrence equation, instead of Fibonacci sequence as in GeM1, which meant that when the either of the previous seeds reach the maximum possible value upon update, they can simply wrap around and start afresh. This means that this wrap around effect will be at random, adding another layer of unpredictability, thus increasing the overall security of the system.

GeM1 and GeM2 include seed updates with each message transmission to ensure that the full PRNG state keeps changing continuously. This is a way to make it harder for an unauthorized entity to 'guess' or 'crack' the system state.

Data Encryption

As with GeM1, encryption of the (already encrypted) ID to generate em is governed by Equation (5.1). Data encryption can be accomplished using any symmetric encryption algorithm. However, in our approach, we have used XOR for encryption due to its simplicity and involutory property, i.e. if XOR is used for encryption, its decryption algorithm is also XOR. This helps in reducing the overhead on resource-constrained devices.

Mutual Authentication, Synchronization and Message Integrity

Similar to GeM1, mutual authentication of authentication is guaranteed by asv. The asv is a parameter that is computed using $numX$, which is generated afresh for each communication. With the key being generated using $numX$, it indirectly represents the current state of the system and hence, the key, which together represents the communicating entity. This is the main reason why *asv* is able to assist in mutual authentication, since only verified entities with synchronized states will be able to generate the correct value of *asv*.

Synchronization is an important activity in our approach given that there are no key exchanges, and that the entities need to be in the same state to successfully authenticate each other. If the entity states are not synchronized, it is highly likely that the protocol might fail due to de-synchronization. To avoid this, our approach provides an implicit synchronization feature. Both entities store *three* key states in memory, i.e. two immediate previous keys (prevKey1 and prevKey2) and their associated states (seeds, parent and generation values), and the corresponding states for the current key. This is done taking into account noise in practical deployment environments, which could result in some frames being lost in transmission. However, even in such scenarios, we assume that even if frames are lost, they may not be lost more than three times consecutively, since the system would sense that there is either an attempt to de-synchronize or an error in the channel.

The reason behind the number "3" is to give enough room for the tag-server pair to accommodate any lost frames, while not so as to facilitate repeat or replay attacks. On successful authentication, the entities are synchronized with the values of the current parent, generation and the encryption key and discard the previous values. This is because once synchronized, the previous keys are not required by either the tag or the server, and any future query by a reader with one of the previous keys would imply that it is an attempt at a replay attack.

When verified and synchronized entities communicate, the message is encrypted with a new key for each transmission. This ensures that it is in a way context specific, thereby making the combination of *asv* and *em* a unique way to represent a message for that transaction. This, in a naïve manner, helps entities employing GeM2 accomplish message integrity verification, in addition to mutual authentication.

Algorithm 2 Key Generation and Encryption Using GeM2

```
1: procedure GEM2KEYGENENCRYPT
2:     //Encrypt:
3:     $em \leftarrow ID \oplus k_i$
4:     //Update Keys:
5:     $sd_i \leftarrow sd_{i-1} + sd_{i-2}$
6:     $numX \leftarrow g(\ sd_i\ )$
7:     $r_{pc} \leftarrow g(\ pc\_choice\ )$
8:     //$pc\_choice$ is the seed for PRNG generating $r_{pc}$
9:
10:    if $r_{pc} = 0$ then
11:        if $g \neq genLimit$ then
12:            //Child Key Generation
13:            $mutation \leftarrow numX \bullet \neg parent\_key$
14:            $k_{i+1} \leftarrow parent\_key \oplus mutation$
15:    else
16:        //Parent Key Evolution
17:        $parent\_key\_new = parent\_key \oplus numX$
18:
19:    //Generate ASV
20:    $asv_i \leftarrow h(\ numX \oplus pattern_{asv}\ )$
```

5.3.4 Summary

GeM2 (summarized in Algorithm 2) is thus, an enhancement to GeM1, with an objective of reducing the linking between keys, introducing more random choices for key updates to increase unpredictability, which in turn facilitates entities to accomplish mutual authentication without exchange of keys and other internal state parameters. GeM2 also ensures that the keys do not converge at any specific pattern and that a unique encryption key will be generated for every frame. This further ensures that consecutive keys will not be similar to each other. The GeM2 algorithm, thus, can be applied to a variety of applications, including other resource-constrained wireless networks such as WBANs.

5.4 Butterfly1: Encryption Scheme Featuring Pseudorandom Numbers and Butterfly Seed Generation

5.4.1 Overview

In this section, we discuss a new encryption scheme that uses pseudorandom number generators, a strategic way of updating their seeds and system state identifiers to accomplish several security goals. We derive our inspiration from the "Butterfly effect". *Butterfly effect* is a concept in chaos theory, defined by Poulin [119] as "hypersensitivity to perturbation". This means that in a non-linear deterministic system, if the initial conditions are changed ever so slightly, there will be drastic changes in the output of a later state. The algorithm we describe in this section features a PRNG seed update mechanism that is a naïve adaptation of the concept of Butterfly effect. This was also proposed as an independent security proposal for RFID systems, but can be extended to other resource-constrained wireless networks such as WBANs.

In our approach, we adapt the concept of Butterfly effect to update the seed of the PRNG, use state identifiers to record the current state of the system, and (re-)use PRNG to generate keys. This is an attempt to demonstrate a simple encryption scheme for resource-constrained wireless networks.

5.4.2 Butterfly Seed Generation Algorithm for PRNG

PRNGs are employed by cryptographic algorithms typically for generating encryption keys or nonces for other purposes. PRNGs are inherently deterministic [120, 121]. Their application is common in lightweight systems that have severe resource constraints that limit their ability to perform sophisticated computations for achieving security (e.g. lightweight passive RFID systems) [59, 60, 93, 94]. Their deterministic nature, combined with a changing seed can make the PRNG into a powerful authentication and a simple key generation system, useful for RFID applications.

A PRNG, $g()$, with periodicity, P, that generates an n-bit number given an m-bit seed can be used as an authentication and key management system if Alice and Bob (synchronized with an initial seed) are able to demand specific random numbers from each other. For example, consider that *Alice* demands that *Bob* respond with the 23rd random number. *Bob* responds with the 23rd random number and includes a demand that *Alice* respond with the 49th random number, and so on. This is a naïve approach to mutual authentication. However, if the seed were to change in a particular way after the period, P, we can increase system unpredictability, thereby improving the security. However, an obvious question that arises is the following — would the change in a seed require either regular communication between the hosts or some other costly mechanism to change or generate a new seed for the PRNG? This could lead to another uncertainty — how can we ensure that the entities are synchronized?

We propose the $Butterfly$ seed generation algorithm as an attempt to answer both questions. We interpret and apply the concept of Butterfly effect in our approach as follows — *if even one bit of the seed of the PRNG is changed, the sequences generated by the PRNG would be significantly different.* The proposed seed generation algorithm employs state identifiers to update the PRNG seed, paving the way for a simple scheme for key generation and message signature generation. Use of state identifiers ensures that the entities are always synchronized and that they can demand the authentication parameters and/or encryption keys that need to be used by the other. This algorithm, thus, provides a simple solution to both the questions that were presented above.

In the $Butterfly$ algorithm, PRNG seeds are updated just by changing one bit in the previously agreed seed. If m is the number of bits in the seed and the PRNG generates an n-bit number as its output, then, the periodicity of the PRNG increases from P to $(m+1) \times P$. Although this is not a direct replication of the Butterfly effect or the implementation of the mathematical theory governing it, we can see that the change in one bit of the seed can enable us to generate a completely different set of random numbers. Furthermore, there would be no predictable pattern in the seed to an adversary (if he does not know the initial seed) — it would depend on the initial seed, and a bit of the seed would be changed at the end of each period of the PRNG (or on demand). We illustrate the concept with an example. Consider a hypothetical PRNG, $g()$ that takes a 6-bit ($m = 6$) seed, S, and generates a sequence of 16-bit ($n = 16$) random numbers, governed by Equation (5.15):

$$r = \{\ g(S)\ \} \tag{5.15}$$

Table 5.1: Example to illustrate the proposed Butterfly Seed Generation algorithm

SEED (S)	Random number sequence
$S_0 = 1\ 0\ 0\ 1\ 0\ 1$	$r_0 = \{\ g(S_0)\ \}$
$S_1 = 1\ 0\ 0\ 1\ 0\ \mathbf{0}$	$r_1 = \{\ g(S_1)\ \}$
$S_2 = 1\ 0\ 0\ 1\ \mathbf{1}\ 0$	$r_2 = \{\ g(S_2)\ \}$
$S_3 = 1\ 0\ 0\ \mathbf{0}\ 1\ 0$	$r_3 = \{\ g(S_3)\ \}$
$S_4 = 1\ 0\ \mathbf{1}\ 0\ 1\ 0$	$r_4 = \{\ g(S_4)\ \}$
$S_5 = 1\ \mathbf{1}\ 1\ 0\ 1\ 0$	$r_5 = \{\ g(S_5)\ \}$
$S_6 = \mathbf{0}\ 1\ 1\ 0\ 1\ 0$	$r_6 = \{\ g(S_6)\ \}$

Here, "$\{\cdots\}$" is used to represent a sequence of pseudorandom number numbers. Let the initial seed be $S = 100101$. Table 5.1 illustrates the changing seed, with the bit highlighted by boldface indicating the changing bit. Note that in our implementation, the seed is updated by changing bits one at a time, beginning with the least-significant bit (LSB) until it reaches the most-significant bit (MSB). To summarize, the Butterfly seed generation can be thought of as a function $\phi()$ that transforms a PRNG seed into a variant of the same, as indicated by equation 5.16.

$$S_j = \phi(\ S\ ) \tag{5.16}$$

Although the different seeds $(S_0, S_1, \ \cdots \ S_6)$ are in the range $0 \leq S_i \leq 2^m - 1$, where m is the number of bits of the seed, we have to note that subsequent seeds are not very similar to the previous seeds. Furthermore, j indicates the number of bits that are transformed, i.e. if $j = x$, the x least significant bits are inverted, bit-wise beginning with the LSB.

Such design of a seed update mechanism allows flexibility in implementation, facilitating varying levels of unpredictability. This is because the implementation could take one of the following forms — (a) with each seed update, only one bit, identified by j can be inverted, which would mean that the implementation would only need to offset to the particular bit and invert it, something that can be realized using shift registers (on hardware), applicable in severely resource-constrained applications; or, (b) with each seed update, all bits until and including the bit j can be inverted, implying that it would change the seed significantly, leading to higher unpredictability. An additional unpredictable element can be introduced in the choice of j. In simple implementations, j could be merely an incrementing number that identifies the position of the bit to be inverted. In an alternative implementation, it could be a number chosen at random that indicates the specific bit (or set of bits) to be changed.

In this section, we proposed a means of updating seeds using state identifiers to increase the unpredictability of a pseudorandom number generator, which ultimately leads to increased security.

5.4.3 Encryption Scheme Featuring PRNG and Butterfly Seed Generation

In this section, we propose a simple encryption scheme and protocol for communication that employ the $Butterfly$ seed generation algorithm discussed previously. This scheme has multiple levels of operation, and each level performs a function similar to enclosing the data (input to that level) in an envelope. This mechanism is used for key generation and to accomplish security goals such as authentication, message confidentiality and data integrity using just one pseudorandom number generator circuit. Figure 5.5 illustrates an overview of the working of the proposed encryption scheme. Of the various modules in this scheme, i.e. key management, data encryption, message signature generator and state information modules, all modules, except the encryption module, use PRNG for their respective functions. This is our attempt to re-use an existing function for multiple purposes.

Consider a PRNG, $g()$, initialised with a seed, s_{init}. The current seed is s_j, s_{init}, using the $Butterfly$ algorithm for seed updates. Each seed, s_j, can generate P random numbers. Each message has a message sequence number, i. Therefore, the i^{th} message to be transmitted would be m_i, which is computed as follows:

$$m_i = M \oplus s_j \tag{5.17}$$

Here, M is the actual message and s_j is the updated seed. This is analogous to enclosing the message in an envelope. Let t_i be the timestamp and θ_i be the message signature of m_i. For each tag, T, the server stores the associated key states and tag IDs in a database. The database row mapping the ID of a specific tag to its attributes is identified by a tag number, η_t, which is different from the tag ID. Note that our encryption scheme does not exchange the tag's ID during communication.

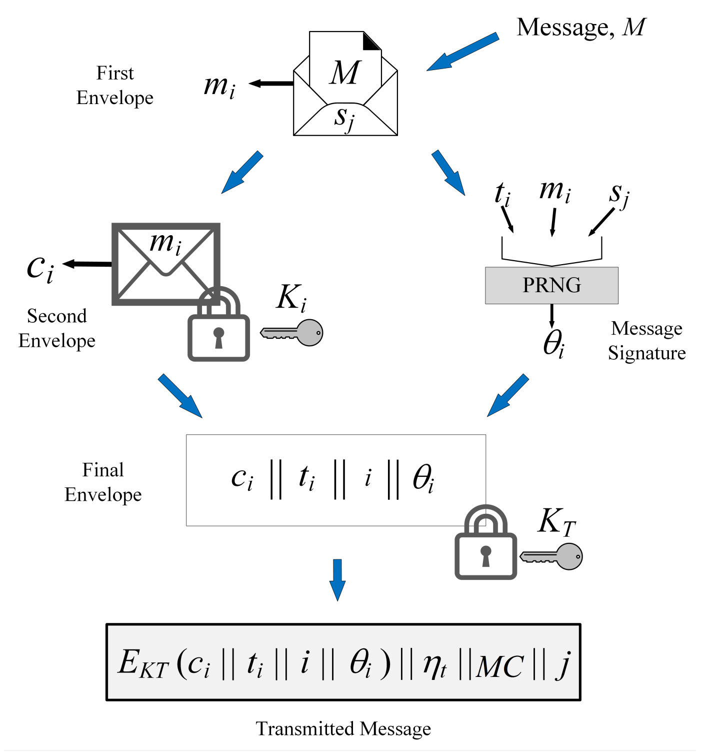


Figure 5.5: Overview of working of the proposed encryption scheme [13]

Key management

Our scheme uses two keys — K_T, the transfer key and K_i, the data encryption key. The data encryption key (K_i) is generated using equation 5.18. This is the key used to encrypt the message, m_i (or enclose the data within another envelope).

$$K_i = g(\ f(\ \phi(\ s_j\ ),\ t_i\ )\ ) \tag{5.18}$$

$\phi()$ is the Butterfly seed transformation function (Equation (5.16)). To reiterate, if $j = 1$, it means that the second bit from the LSB has been transformed, and so on. $f()$ is a function to combine the transformed seed and the timestamp. This is achieved by an XOR operation. Since our algorithm is modular in nature, $f()$ can be changed as required by any implementation without affecting other modules in the scheme.

The transfer key, K_T, is used to encrypt the final message that the tag responds with (or add a final envelope over the data), and is generated also by a PRNG using s_j as the seed. Generation of K_T is summarized by Equation (5.19).

$$K_T = g(\ s_j\ ) \tag{5.19}$$

Data encryption

We use XOR for encryption (Equation (5.20)), because of its involutory property (it is its own inverse, and hence can be used as the decryption function as well) and to keep computations low.

$$cipher_i = E_{Key_x}(\ message_i\ ) = message_i \oplus Key_x \tag{5.20}$$

Here, $cipher_i$ is the cipher text generated when a message, $message_i$, is encrypted using key, Key_x. The encryption function (E_{Key_x}) used is XOR (represented by $\oplus$). Key_x could be K_i when encrypting m_i or K_T when encrypting the combined message prior to transmission.

Message signature generation

Typically, message digests are generated using hash functions [122]. However, PRNGs have the ability to generate unique sequences given a specific seed. We exploit this property of a PRNG and use it to generate a message signature, θ_i, as specified by equation 5.21.

$$\theta_i = g\ (\ f\ (\ s_j,\ m_i,\ t_i\ )\ ) \tag{5.21}$$

Here, $f()$ combines the seed (s_j), message (m_i) and the timestamp (t_i). Note that this is the same combination function used in key generation.

Essentially, we are re-using the modules and hardware to accomplish multiple functions. Furthermore, with this message signature being dynamically generated and dependent on timestamp, it helps our approach provide context (time, message, entity)-specific signatures. In this case, entity-specific property is achieved by deploying different PRNG seeds on different devices and each having the possibility of being in a different state, j.

State information management

The state information module (S_M) updates and maintains sequence number, i, and PRNG seed state, j. Our scheme also uses a 2-bit pattern called message code, MC, which indicates the type of message being communicated. The format for MC is summarized by Table 5.2. MC, along with the timestamp, t_i and the message sequence number, i, provides additional means to prevent replay attacks.

Table 5.2: Message Code (MC) format and description

MC	Bit Pattern	Description
$MC = 0$	00	First message sent by server to tag
$MC = 1$	01	Response sent by tag (equation 5.23)
$MC = 2$	10	Acknowledgement sent by server
$MC = 3$	11	Special message: Instructs tag to update its PRNG seed with the new seed, provided it is able to validate the server (equation 5.26)

Protocol of Operation

The proposed encryption scheme works as follows:

Step-1: The reader, R, requests the server for a connection request. The server then encrypts the current timestamp, t_i using the previously synchronized transfer key, K_T and forwards $E_{K_T}(\ t_i\ ) \parallel MC$ (with $MC = 0$) to the reader.

Step-2: Reader queries the tag (T) with the encrypted timestamp, $E_{K_T}(\ t_i\ ) \parallel MC$. The tag retrieves the t_i using the previously synchronized transfer key. Following this, the tag computes a random value for j, transforms the s_{init} to state s_j (Equation (5.16)). The tag then performs the XOR of its message M with s_j to generate the message to be transmitted, m_i (Equation (5.22)).

$$m_i = M \oplus s_j \tag{5.22}$$

Step-3: Simultaneously, the tag generates the data encryption key, K_i (Equation (5.18)), and the transfer key, K_T (Equation (5.19)).

Step-4: Following this, the tag generates the encrypted message, c_i (Equation (5.20)) and uses it to generate the message signature, θ_i.

Step-5: The transmitted message, M_T is generated as defined by Equation (5.23):

$$M_T = E_{K_T}\ (c_i \parallel t_i \parallel i \parallel \theta_i) \parallel j \parallel MC \parallel \eta_t \tag{5.23}$$

Where, $E_{K_T}()$ represents an additional round of encryption performed on a concatenated message comprising of the cipher text (c_i), message signature

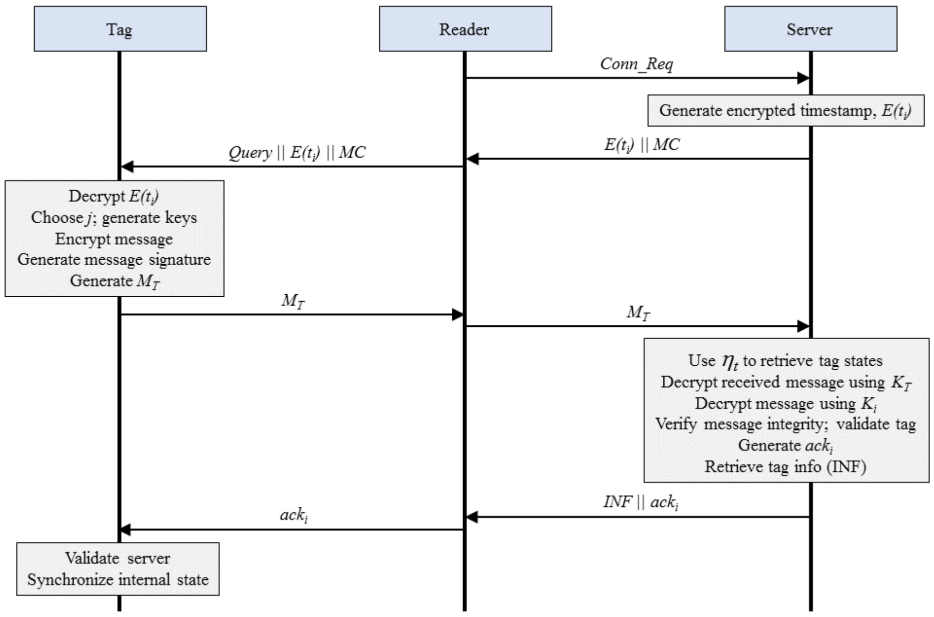

Tag
Reader
Server
Conn_Req
Generate encrypted timestamp, $E(t_i)$
$E(t_i)$ || MC
Query || $E(t_i)$ || MC
Decrypt $E(t_i)$
Choose j; generate keys
Encrypt message
Generate message signature
Generate M_T
M_T
M_T
Use η_t to retrieve tag states
Decrypt received message using K_T
Decrypt message using K_i
Verify message integrity; validate tag
Generate ack_i
Retrieve tag info (INF)
INF || ack_i
ack_i
Validate server
Synchronize internal state

Figure 5.6: Working of the proposed encryption scheme [13]

(θ_i), timestamp (t_i), message sequence number (i), and $MC = 1$ to indicate tag response. $\|$ represents concatenation. The above steps and the working of the proposed encryption scheme are summarized in Figure 5.6.

During decryption, the server uses the tag number, η_t to retrieve the attributes of the specific tag, uses seed state identifier (j) to generate the transfer key (K_T) to decrypt the received message. It then (verifies and) uses timestamp (t_i) to generate the decryption key (K_i) and decrypts the message, m_i. Following this, it computes the message signature and verifies the integrity of the message. This ensures that the tag is authenticated, as only a genuine tag could have generated a valid key with state, s_j, to have encrypted the said message. The server then recovers the message, M by performing XOR (decryption) operation on m_i and s_j. When the tag is found to be valid, the server retrieves information (INF) about the object, updates its database entry corresponding to the tag with updated information (i.e. i). Server then sends INF along with an encrypted acknowledgement (ack_i), containing the current timestamp (t_{i+1}) (Equations (5.24) and (5.25)):

$$c_{ack} = E_{K_i}\ (ACK \parallel t_{i+1}) \tag{5.24}$$

$$ack_i = E_{K_T}\ (c_{ack} \parallel i \parallel \theta_i) \parallel MC \tag{5.25}$$

Here, c_{ack} is the acknowledgement cipher; K_T and K_i are the same keys used by the tag for synchronization; ACK is the pre-agreed acknowledgement pattern; t_{i+1} is the latest timestamp at the server; i is the previous sequence number at the tag (also to ensure synchronization); and $MC = 2$ to indicate that the message is an acknowledgement. On receiving the acknowledgement, the tag validates the server and the states are synchronized.

Seed refresh

In case the server needs the tag to update its seed, it sends a seed refresh message, m_{su}, to the tag:

$$m_{su} = E_{K_T}\ (c_u \parallel t_u \parallel i \parallel \theta_u) \parallel j \parallel MC \parallel \eta_t \tag{5.26}$$

Where, $c_u = E_{K_i}(\ s' \oplus\ s_j)$ is the seed update cipher, which is the result of encrypting the XOR-ed combination of the new seed s' and the old seed s_j, using the key K_i generated as specified by Equation (5.18); t_u is the current (seed update) timestamp; θ_u is the corresponding message signature; i is the updated sequence number, which is one more than the previously synchronized sequence number; and, j is the $Butterfly$ state randomly chosen by the server for the current operation, with $MC = 3$.

5.4.4 Summary

The operation of the Butterfly1 algorithm is summarized in Algorithm 3. In this section, we proposed a new encryption scheme employing PRNGs for key generation and message signature generation, foundation for which is the proposed *Butterfly* algorithm, based on a naïve adaptation of the Butterfly effect. This is an attempt to re-use available functions to accomplish multiple functions such as key generation, encryption, message signature generation, which results in achieving security objectives such as confidentiality, integrity, (mutual) authentication, and non-repudiation (by association). Simplicity and re-use of the Butterfly1 approach encourage its application in other resource-constrained wireless applications.

Algorithm 3 Key Generation and Encryption Using Butterfly1

```
1: procedure BUTTERFLY1KeyGenEncrypt
2:     //Butterfly Seed Update:
3:     $Choose\ j$
4:     $s_j \leftarrow \phi(\ s\ )$
5:
6:     //Generate Keys:
7:     $K_i \leftarrow g(\ f(\ s_j, t_i\ )\ )$
8:     $K_T \leftarrow g(s_j)$
9:
10:    //Choose Appropriate Message Code (2 bits):
11:    if $Message\ is\ sent\ by\ server\ to\ tag$ then $MC \leftarrow 00$
12:    else if $Response\ sent\ by\ tag$ then $MC \leftarrow 01$
13:    else if $Acknowledgement\ sent\ by\ server$ then $MC \leftarrow 10$
14:    else if $Special\ message$ then $MC \leftarrow 11$
15:
16:    if $MC = 00$ or $MC = 01$ or $MC = 10$ then
17:        //First Encrypt:
18:        $m_i \leftarrow M \oplus s_j$
19:        //Second Encrypt:
20:        $c_i \leftarrow m_i \oplus K_i$
21:        //Generate Message Signature:
22:        $\theta_i \leftarrow f(\ t_i \parallel m_i \parallel s_j\ )$
23:        //Final Encrypt:
24:        $EKtCi \leftarrow (\ c_i \parallel t_i \parallel i \parallel \theta_i\ ) \oplus K_T$
25:
26:        //Generate Message to be Transmitted:
27:        $M_T \leftarrow EKtCi \parallel j \parallel MC \parallel \eta_t$
28:    else if $MC = 11$ then
29:        //Generate Seed Update Message:
30:        $m_{su} = E_{K_T}\ (c_u \parallel t_u \parallel i \parallel \theta_u) \parallel j \parallel MC \parallel \eta_t$
```

5.5 HiveSec1: Algorithm for Security Inspired by Beehives and Bee Swarms

5.5.1 Overview

Presented in this section is another new way of using pseudorandom number generators (PRNG) for security in a resource-constrained environment. Specifically, as with the algorithms previously discussed, we propose this as an independent security proposal for RFID systems. HiveSec1 includes with it a new mechanism to generate keys and a mechanism to protect against attacks in progress (such as denial of service, DoS, and replay attacks), and makes use of a message signature scheme called HiveSign for entity authentication [14]. This approach is inspired by the symmetric (hexagonal) structure of beehives and the concept of bee swarms. As with other algorithms proposed in our thesis, HiveSec1 facilitates key generation at the entities instead of key exchange. Although this proposal is a security solution including all aspects of information security such as key management, encryption and authentication, the focus is mainly key management and authentication — key management, since a publicly available encryption algorithm is only as good as the key, and authentication, which is primarily to be established (using generated keys and message signatures) among communicating entities to ensure secure (trust-based) communication. HiveSec1 is designed to work with any symmetric encryption algorithm.

5.5.2 HiveSec1: The Concept

A beehive is made up of hexagonal 3-dimensional structures that we refer to as honey pods, which are actually constructed by bees using beeswax[1]. Each honey pod is used by the bees to store honey, food and eggs. When forming a new colony, the queen bee and the worker bees assemble at a specific location, in a process called swarming. The worker bees then create the beehive using beeswax. Furthermore, if a colony of bees is disturbed, a swarm of bees attack and sting any organisms in their path, as a protective mechanism [124].

In HiveSec1, we propose modules for security that are based on these concepts. In

[1] *Beeswax* is produced by worker bees, and are a "complex mixture of saturated and unsaturated linear and complex monoesters, hydrocarbons, free fatty acids, free fatty alcohols, and other minor substances" [123].

particular, we draw inspiration from the hexagonal structure of the honey pods and the mechanism in which bees attack when their hive is disturbed. In our approach, we construct a *hive* of PRNG seeds (that we refer to as *seedhives*), where each honey pod is a PRNG seed. We consider six such seedhives in our approach. Each time an encryption key needs to be generated, we use the concept of *disturbing* one of the seedhives at random, and using the chosen seed to generate a specific key (*HiveKey* module). We employ the HiveSign message signature algorithm [14] to facilitate verification of the messages and keys. This design also facilitates a mechanism to protect the system against attacks such as replay attacks or denial-of-service (DoS). This is a naïve attempt to *attack the attacker*. This concept has enabled us to propose secure modules for networks with/without resource constraints. The novelty of this approach is in the mechanism used to generate the encryption keys, message signatures, state identification (and session/session key updates), and protection against attacks.

5.5.3 HiveSec1: Security Inspired by Bees

HiveKey: Key Management

In HiveKey, we use a PRNG to generate a unique key for each communication based on the following concept — consider six seedhives in a 'neighbourhood' (A, B, C, D, E and F in Figure 5.7, which can be programmatically conceptualized as an array of seeds). To illustrate the concept, we consider this structure to be similar to a hexagon, with the seedhives at its vertices. Each seedhive in HiveKey is perceived to consist of a finite number of seeds, each of them assumed to occupy one pod (as illustrated in Figure 5.8). To generate a key, we 'disturb' (or, choose) one seedhive at random and choose one seed from the hive (S_C), and assume that this disturbance also 'disturbs' part of the seedhives to the immediate left (S_L) and right (S_R) of S_C. The actual key generated would then be a combination of the three seed components, S_C, S_L and S_R. In Figure 5.7, we illustrate a case where seedhive A is disturbed, and B and F are minimally disturbed (represented by red coloured segments in hives).

Seedhives

HiveKey requires storing six PRNG seeds, referred to as *parent* seeds, $\{p_1, p_2, p_3, ..., p_6\}$. These are deployed in the communicating entities at deploy time. Each *parent* seed

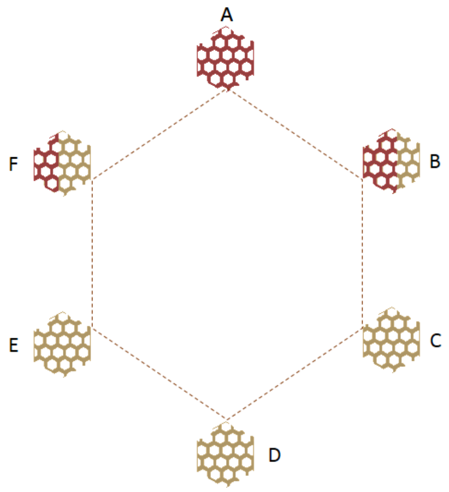


Figure 5.7: Conceptual Illustration of the Seed Generation in HiveKey

is perceived to have either six (6) or eighteen (18) *child* seeds (or, children), based on a 1-bit version code (VER) (Figure 5.8). We say "perceived" to indicate that they are not stored, but generated when required. They are generated as summarized in equation 5.27, with c_j being the x^{th} pseudorandom sequence generated by the PRNG, $g()$, using parent, p_i as the seed.

$$c_j = g_x\ (\ p_i\ ) \tag{5.27}$$

Therefore, we can perceive that a specific seedhive structure having either 6 or 18 *child* seeds exists, even though the only seed we have stored is the *parent* seed, p_i. We can perceive entities to have stored either $6 \times (1\ parent + 6\ children) = 42\ seeds$ or $6 \times (1\ parent + 18\ children) = 114\ seeds$ in total, depending on the version (Figure 5.8).

Seed chooser, a.k.a. the "hive disturber"

HiveKey uses timestamp to choose a seed from the seedhive (or, disturb the hive). We use timestamp as the "hive disturber" or the parameter that chooses one seed from the seedhives. Timestamps are sent in the messages sent by the communication initiator and this helps in determining the version code (VER), the seed and hence, the key. We consider a 32-bit timestamp illustrated this in Figure 5.9. Here, t_{b31} is the most-significant bit (MSB) and t_{b0} is the least-significant bit (LSB). We use *nine* LSBs of the timestamp to derive the version code, and to identify the parent and

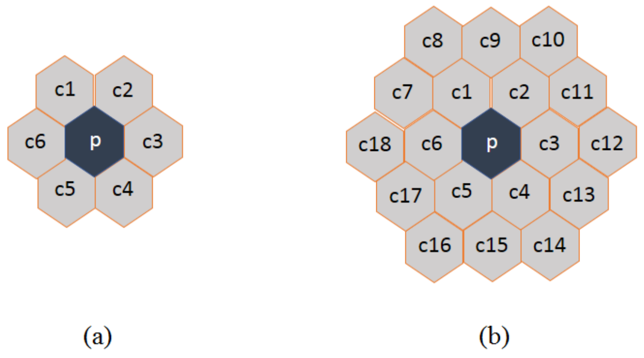


Figure 5.8: Conceptual Illustration of Seedhives

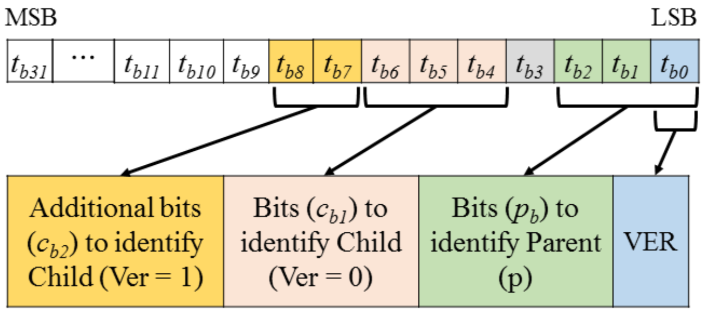

MSB
LSB
t_{b31}
...
t_{b11}
t_{b10}
t_{b9}
t_{b8}
t_{b7}
t_{b6}
t_{b5}
t_{b4}
t_{b3}
t_{b2}
t_{b1}
t_{b0}
Additional bits (c_{b2}) to identify Child (Ver = 1)
Bits (c_{b1}) to identify Child (Ver = 0)
Bits (p_b) to identify Parent (p)
VER

Figure 5.9: Use of Timestamp to Determine Choice of Parent/Child Seeds

child seeds.

On receiving a message from the communication initiator, the responder decodes the timestamp and proceeds as follows:

- The version code (VER) is the LSB of the timestamp.
  - If $VER = 0$, the parent seed, p_i, will have six child seeds, c_j (Figure 5.8 (a)).
  - If $VER = 1$, the parent seed, p_i, will have eighteen child seeds, c_j (Figure 5.8 (b)).
- The three LSBs of the timestamp (t_{b2}, t_{b1}, t_{b0}) are used to determine the parent seed. First, the integer equivalent of the 3-bit parent choice is determined. Since there are six parent seeds stored in memory, the parent chosen, p is determined as follows:

$$p = p_{b(decimal)} \ mod \ 6 \tag{5.28}$$

$p_{b(decimal)}$ indicates decimal representation of the 3-bit number, p_b, and mod represents modulus operation. The modulus operation is used in this case considering the fact that the range of integer equivalents of 3-bit binary numbers is [0,7], while we need only numbers in the range [0,5] to represent the parent seeds stored in HiveSec1.

- To determine the child number, x (i.e. child seed for the chosen parent, p), HiveKey uses the three bit combination (c_{b1}) in case of $VER = 0$ (t_{b6}, t_{b5}, t_{b4}) or the five bit combination ($c_{b2} \parallel c_{b1}$) in case of $VER = 1$ ($t_{b8}, t_{b7}, t_{b6}, t_{b5}, t_{b4}$) (Equations (5.29a) and (5.29b), respectively). Note that $\parallel$ represents concatenation operation.

$$x = c_{b1(decimal)}\ mod\ 6 \tag{5.29a}$$

$$x = (c_{b2} \parallel c_{b1})_{(decimal)}\ mod\ 18 \tag{5.29b}$$

- The algorithm then generates the child seed (as the chosen seed, S_C) using Equation (5.27).

Computation of Actual Key Generation Seed, S_A

To generate the seed used for key generation, S_A, we consider the two neighbours of the chosen parent seed, the left parent (S_L) and the right parent (S_R). We assume the following analogy from bees and beehives — when one seedhive is disturbed, 60% of the left seedhive and 40% of the right seedhive are disturbed. Thus, we extract 60% of the MSBs (or, left-most bits) from the left parent, S_L, and 40% of the LSBs (or, right-most bits) from the right parent, S_R, using bit-masks and the logical AND operation (Equations (5.30a) and (5.30b)). Here, $\bullet$ represents the AND operation, $Left\_Mask$ is a bit pattern with 60% of the left-most bits set to 1, i.e. FFF...0, and $Right\_Mask$ is a bit pattern with 40% of the right-most bits set to 1, i.e. 0...FFF.

$$S'_L = S_L \ \bullet \ Left\_Mask \tag{5.30a}$$

$$S'_R = S_R \ \bullet \ Right\_Mask \tag{5.30b}$$

The actual seed is computed as summarized by Equation (5.31) ($\oplus$ represents the XOR operation and $\vee$ indicates the OR operation). S_A is the actual seed that will be used to generate the encryption key, K_S.

$$S_A = S_C \ \oplus \ (S'_L \ \vee \ S'_R) \ \oplus \ t_s \tag{5.31}$$

PRNG State, s, and Encryption Key, K_S

HiveKey retrieves four LSBs $(t_{b3}, t_{b2}, t_{b1}, t_{b0})$ from the timestamp and assigns them as the 4-bit PRNG state, s. The encryption key is then computed to be the s^{th} (pseudo)random sequence generated by the PRNG, $g()$, using S_A as the seed (Equation (5.32)).

$$K_S = g_s \; (\; S_A \;) \tag{5.32}$$

Outer-envelope key, K_O

In addition to the encryption key, K_S, HiveSec1 makes use of an outer envelope key, K_O (determined similar to K_S). In this case, however, we consider an initial timestamp, t_{s0} (stored at deploy time), to determine the VER code and child seed number (using Equations (5.28), (5.29a) and (5.29b)). Next, the child seed, S_{CO}, is generated (Equation (5.27)), followed by choosing of left and right neighbours, S_{LO} and S_{RO} (Equations (5.30a) and (). The actual key generation seed, S_{AO}, is generated using S_{CO}, S_{LO}, S_{RO} and initial timestamp, t_{s0}, in place of S_C, S_L, S_R and timestamp, t_s, in Equation (5.31). Finally, the VER and p_b bits in t_{s0} are used to determine the state, s_{O1}.

$$s_O = S_{O1} \;\oplus\; s_e \tag{5.33a}$$

$$K_O = g_{s_O} \; (\; S_{AO} \;) \tag{5.33b}$$

The main difference in this key generation is use of the parameter, s_e (chosen at random and sent by the initiator) in combination with s_{O1}, to determine the actual state, s_O (Equation (5.33a)).

Seed Updates

One of the notable concerns with such a key generation mechanism is the probability of chosen seeds, and hence the keys, being repeated. This occurs if the parent seeds stored in the entities remain the same. HiveSec1 also includes a mechanism to update the seeds, which are also determined at random, but specified by the initiator of the communication (such as the server in an RFID system, or a body hub node in WBANs). A 2-bit code called the *Message Code* is used to facilitate such seed updates, while maintaining high unpredictability in the keys used for encryption. This is discussed in detail next.

Algorithm 4 Key Generation Using HiveSec1

```
1: procedure HIVESEC1KeyGen
2:     //Retrieve Timestamp ($t_s$ for computing $K_S$ and $t_{s0}$ for computing $K_O$)
3:     $t_x \leftarrow appropriate\ timestamp$
4:     $VER \leftarrow LSB\ of\ t_x$
5:     $parent\_index \leftarrow INTEGER(t_x(2) \parallel t_x(1) \parallel t_x(0)) mod 6$
6:     $parent\_seed \leftarrow parent\_seed\_array[parent\_index]$
7:
8:     if $VER = 0$ then
9:         $child\_index \leftarrow INTEGER(t_x(6) \parallel t_x(5) \parallel t_x(4)) mod 6$
10:    else if $VER = 1$ then
11:        $child\_index \leftarrow INTEGER(t_x(8) \parallel t_x(7) \parallel t_x(6) \parallel t_x(5) \parallel t_x(4)) mod 18$
12:    $child\_seed \leftarrow g(\ parent\_seed\ )$ //Generate child seed
13:
14:    $left\_parent = parent\_seed\_array[(parent\_index - 1) mod 6]$
15:    $S\_L' \leftarrow left\_parent \bullet Left\_Mask$
16:    $right\_parent = parent\_seed\_array[(parent\_index + 1) mod 6]$
17:    $S\_R' \leftarrow right\_parent \bullet Left\_Mask$
18:
19:    $S_A \leftarrow child\_seed \oplus (S\_L' + S\_R') \oplus t_x$ //Actual key generation seed
20:
21:    if $Key\ is\ \ K_S$ then
22:        $s \leftarrow INTEGER(t_x(3) \parallel t_x(2) \parallel t_x(1) \parallel t_x(0))$
23:        $s\_index \leftarrow s$
24:    else if $Key\ is\ \ K_O$ then
25:        $s_O = S_{O1}\ \oplus\ s_e$
26:        $s\_index \leftarrow s_O$
27:
28:    //Generate $s\_index^{th}$ pseudorandom sequence as key
29:    $Key_x \leftarrow g(\ S\_A\ )$
```

HiveKey (summarized in Algorithm 4) is thus a new and simple mechanism to generate keys uniquely for each message exchanged. The two keys (K_S and K_O) are used to ensure that the message remains confidential and the attributes to generate the key remain a secret. It must be noted that HiveKey does not involve key exchanges, but independent generation of keys at both the initiator and responder. Since key generation depends on the time and parameters specific to each entity (such as the seeds stored and the associated initial timestamps), unpredictability associated with the keys is high.

State Information and Message Types

The state information module in HiveSec1 helps maintain the internal system state, thereby facilitating synchronized key generation at the entities. This module is responsible for computing and maintaining the parameters representing the PRNG state, i.e. s, s_O and s_e sent by the initiator. These determine the exact state of the PRNG to generate the respective keys. Additionally, this module also helps maintain session parameters that will be described in the paragraphs that follow.

HiveSec1 facilitates the notion of sessions in each communication. Two terms are important in this context — (a) Association, and (b) Session. A session is a short duration of time (δ), where entities communicate with each other. The default session duration is δ_0 milliseconds and each session is identified by a session identifier, n_δ. Every session update (after $\delta = \delta_0$ milliseconds, by default) increments n_δ. An association, on the other hand, is a session of longer duration, which is updated every time there is an update to the initial timestamp, t_{s0}. Thus, an association may have multiple sessions.

A parameter called session sequence number, SSN, helps in identifying the sequence number of messages in a session. This is an integer that increments with each new message. SSN is set to 1 when the initiator sends the first message, $M1$, updated to 2 with the response $M2$ sent by the responder, and incremented by 1 with each subsequent message. This is a counter to identify the message sequence number, in a particular session, which also forms an important element in protection against replay attacks.

Table 5.3: Message Code (MC) format and description

MC (Bit pattern)	Message Type	Message Contents & Their Use
$MC = 0$ (00)	Regular Message	t_s: Current timestamp; F_O: Optional; will contain message, encrypted with K_S as explained by Equation (5.45);
$MC = 1$ (01)	Special Message, with δ_O (No change in δ)	t_s: Timestamp to replace t_{s0}; F_O: Contains the value of δ_O; δ_O is used as the duration of the session, overriding default δ; n_δ and SSN are reset to 0;
$MC = 2$ (10)	Special Message, with δ_O (Change in δ)	t_s: Timestamp to replace t_{s0}; F_O: Contains the value of δ_O; δ_O is used as the duration of the session, overriding default δ; n_δ and SSN are reset to 0; In this case, δ_O will replace δ;
$MC = 3$ (11)	Special Message (Seed Update)	t_s: Timestamp to replace t_{s0}; F_O: Contains new *seed*, encrypted as explained by Equation (5.45); p_b bits from the new timestamp (t_s) indicates seed to be replaced; Outer envelope key is derived using previous value of t_{s0}; n_δ and SSN are reset to 0;

Communication between entities may include several types of messages summarized in Table 5.3. Each message type is identified by a 2-bit code called the *Message Code*. The session identifier, n_δ, is computed using the timestamp, t_s, and the initial timestamp, t_{s0}, based on the message type:

- In case of a regular message, $MC = 00$:
  - The entity computes the difference between the two timestamps, t_s and

t_{s0}, and determines the integer division of this difference with the default value of δ, i.e.

$$n_\delta = \frac{(\ t_s\ -\ t_{s0}\ )}{\delta} \tag{5.34}$$

– This value of n_δ is compared with the ideal session update value, $n_{\delta,ideal}$, which is the previously used value of n_δ incremented by 1.

– If $n_\delta\ \geq\ n_{\delta,ideal}$, the session is considered to be updated.

- However, special messages ($MC\ =\ 01$ or $MC\ =\ 10$ or $MC\ =\ 11$) force an association renewal for the system, which means that the default value of n_δ and SSN are reset to 0. Special messages may or may not change the value of session duration, δ.

With this mechanism, HiveSec1 facilitates a system to include the notion of sessions and associations implicitly. Automatic session updates also change authentication parameters as explained next. Furthermore, each communicated message in a session generates a new K_S. Key updates with each message ensures high unpredictability and thus, increased security. The outer envelope key, K_O, in contrast is generated every time there is session update or an increment in n_δ.

Message Signature Module: HiveSign

HiveSec1 also features a mechanism to generate message signatures, as a fingerprint for a specific transaction between a sender, the time of communication and encryption parameters. For this, we employ the HiveSign algorithm [14], which helps in verifying the integrity of the message and the encryption key, in addition to verifying the authenticity of the sender. HiveSign generates a message signature, $mSign_{Entity}$ (with $Entity\ =\ I$, for initiator and $Entity\ =\ R$, for responder), by generating a unique component to be signed, extracted from the intended message, the key and other parameters. The component to be signed is based on the value of VER, which is the LSB of the timestamp, t_s.

Initiator Message Signature, $mSign_I$

The initiator uses t_s and K_S, in addition to SSN and its ID, ID_I, to compute the message signature as follows:

- When $VER = 0$: Most-significant half of t_s and least-significant half of K_S are extracted using masks, as summarized by Equations (5.35) and (5.36), respectively. Here, $\bullet$ represents logical AND operation, $LEFT\_MASK\_HS$ is a pattern with the most-significant half set to 1 (i.e. $FFF...0$), and $RIGHT\_MASK\_HS$ is a pattern with the least-significant half set to 1 (i.e. $0...FFF$). Note that this is different from the left/right masks used for seed extraction since these extract 50% of the bits from left/right halves of the considered components, while during the generation of the actual seed, S_A, 60% of the left neighbour and 40% of the right neighbour seed are extracted using the masks.

$$C1 = t_s \;\bullet\; LEFT\_MASK\_HS \tag{5.35}$$

$$C2 = K_S \;\bullet\; RIGHT\_MASK\_HS \tag{5.36}$$

- When $VER = 1$: Least-significant half of t_s and most-significant half of K_S are extracted using the $RIGHT\_MASK$ and $LEFT\_MASK$, respectively:

$$C1 = K_S \;\bullet\; LEFT\_MASK\_HS \tag{5.37}$$

$$C2 = t_s \;\bullet\; RIGHT\_MASK\_HS \tag{5.38}$$

- Using $C1$ and $C2$, the component to be signed (CS_I) is determined as summarized in Equation (5.39). Here, $\vee$ represents the logical OR operation, which is used to combine the extracted components with the entity ID.

$$CS_I = (\ C1 \ \vee\ C2 \ \vee\ ID_I\ ) \parallel SSN \tag{5.39}$$

- Then, using VER (t_{s0}) (LSB of t_{s0}), the message signature is computed as follows (note: $CS_{Entity} = CS_I$, and $mSign_{Entity} = mSign_I$ in this case):
  - If VER $(t_{s0}) = 0$, the message signature is the output of a hash operation, $h()$, performed on CS_{Entity}.

$$msign_{Entity} = h\ (\ CS_{Entity}\ ) \tag{5.40}$$

  - If VER $(t_{s0}) = 1$, the message signature is the output of PRNG function, $g()$, performed using CS_{Entity} as the seed.

$$msign_{Entity} = g\ (\ CS_{Entity}\ ) \tag{5.41}$$

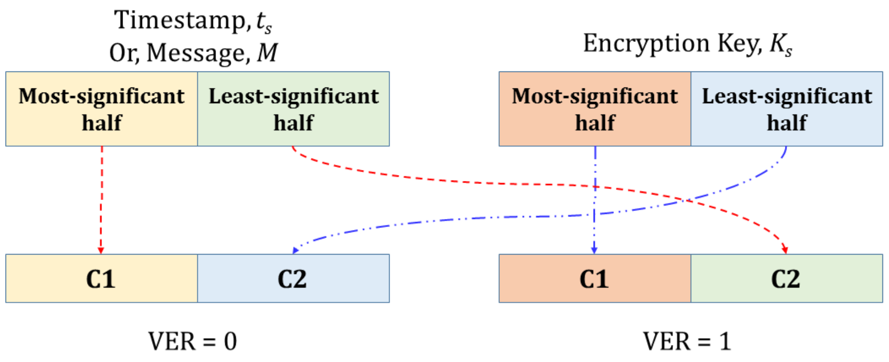


Figure 5.10: Extraction of Components $C1$ and $C2$ [14]

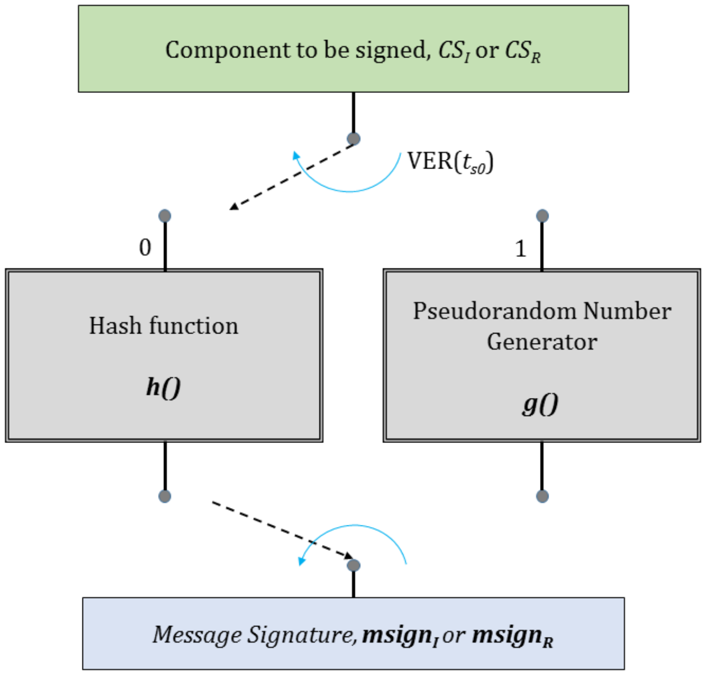

Component to be signed, CS_I or CS_R
VER(t_{s0})
0
1
Hash function
h()
Pseudorandom Number Generator
g()
Message Signature, msign$_I$ or msign$_R$

Figure 5.11: Generation of Message Signatures [14]

Responder Message Signature, $mSign_R$

$mSign_R$ includes the response (M) instead of t_s, along with a half of K_S chosen at random. As with computing $mSign_I$, $C1$ and $C2$ are extracted as summarized in Equations (5.35) and (5.36) (or, Equations (5.37) and (), respectively. CS_R is the component to be signed, which is determined as a combination of $C1$, $C2$, ID_R, SSN and n_δ, as summarized in Equation (5.42).

$$CS_R = (\ C1\ \vee\ C2\ \vee\ ID_R\ ) \parallel n_\delta \parallel SSN \tag{5.42}$$

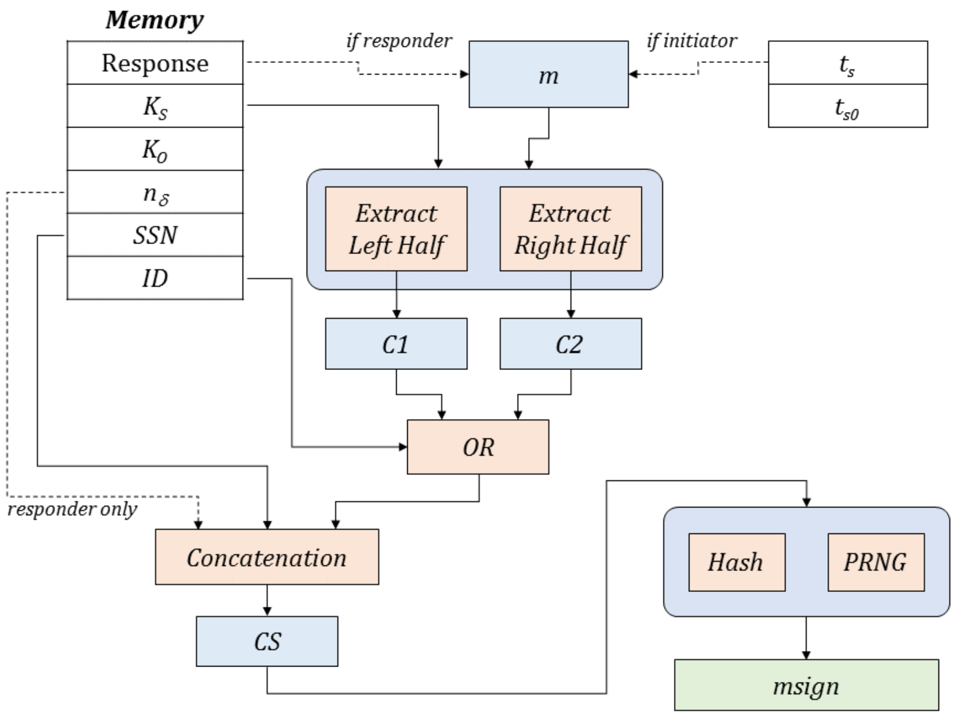


Figure 5.12: HiveSign: Operational Modules/Units [14]

Then, based on $VER\ (t_{s0})$, either Equation (5.40) or Equation (5.41) is used to compute the responder message signature, $mSign_R$.

The message signatures are, therefore, a unique combination of parameters that represent each communicating entity, the session, the message and the key, which facilitates mutual authentication of entities, and the verification of the message and the key. Its ability to represent each transaction uniquely is a measure to protect the system against replay attacks.

The process of extracting the components $C1$ and $C2$ (in generation of both $msign_I$ and $msign_R$) is summarized by Figure 5.10, while the generation of the message signatures themselves is illustrated by Figure 5.11. Figure 5.12 illustrates the various operational modules/units in HiveSign.

Data Encryption

For our encryption scheme featuring HiveSec1 for key generation and authentication, we use XOR as the encryption function, mainly as with other schemes proposed in this thesis, for its involutory property and reduced resource usage. Encryption and

Algorithm 5 Encryption Using HiveSec1

```
1: procedure HIVESEC1Encrypt
2:     //Determine level of encryption
3:     //First level: $K_S$ is used for encryption the message
4:     //Second level: $K_O$ is used to encrypt the already
5:     // encrypted message and other parameters
6:
7:     if First level of encryption then
8:         $Key_x \leftarrow HIVESEC1KeyGen\ (\ generate\_K_S\ )$
9:     else if Second level of encryption then
10:        $Key_x \leftarrow HIVESEC1KeyGen\ (\ generate\_K_O\ )$
11:
12:    $cipher_i \leftarrow E_{Key_x}(m_i) = m_i\ \oplus\ Key_x$
```

decryption in HiveSec1 are summarized by Equations (5.43a) and (5.43a).

$$cipher_i = E_{Key_x}(m_i) = m_i\ \oplus\ Key_x \tag{5.43a}$$

$$decrypted\ message, m'_i = E_{Key_x}(m_i)\ \oplus\ Key_x \tag{5.43b}$$

Here, m_i is the message to be encrypted; Key_x is the encryption key, which could be either K_S or K_O, and m'_i is the decrypted message. Algorithm 5 summarizes data encryption using HiveSec1.

Protocol of Operation

The protocol of operation in HiveSec1 is illustrated in Figure 5.13. When two entities, initiator (ID_I) and responder (ID_R), want to communicate using HiveSec1, the operation proceeds as follows:

Step-1: $M1$ is the first message in a communication and is sent by the initiator. It contains three components, namely — (a) a block encrypted using the outer envelope key, K_O, containing the current timestamp, t_s, the initiator message signature, $msign_I$, and an optional field, F_O; (b) followed by an unencrypted and random 6-bit state variable, s_e; and, (c) an unencrypted 2-bit message code,

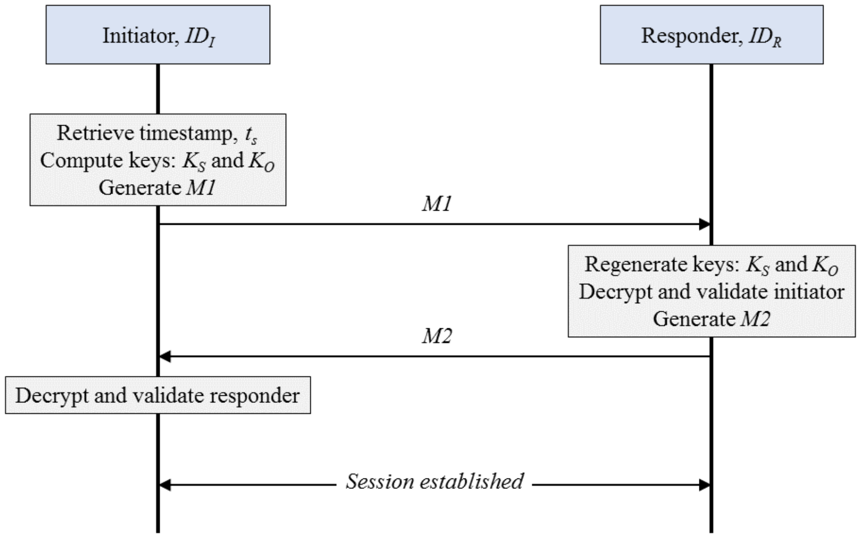

Initiator, ID_I
Responder, ID_R
Retrieve timestamp, t_s
Compute keys: K_S and K_O
Generate *M1*
M1
Regenerate keys: K_S and K_O
Decrypt and validate initiator
Generate *M2*
M2
Decrypt and validate responder
Session established

Figure 5.13: HiveSec1: Protocol of Operation

MC. The structure of $M1$ can be summarized as follows:

$$M1 = E_{K_O}\ (\ t_s \parallel msign_I \parallel F_O\ ) \parallel s_e \parallel MC \tag{5.44}$$

Step-2: To construct $M1$, the initiator uses the following steps:

- The initiator first decides the type of message to be sent. Next, it retrieves timestamp, t_s, and generates the encryption key, K_S, as described in Equations (5.28), (5.29a) (or 5.29b), (5.30a), (5.30b), (5.31) and (5.32).
- Simultaneously, it uses bits from the initial timestamp, t_s0, and generates the outer-envelope key, K_O, employing the techniques described in Equations (5.28), (5.29a) (or 5.29b), (5.30a), (5.30b), (5.31), (5.33a) and (5.33b).
- Following this, it determines the data portion to be encrypted, i.e. the intended message. In HiveSec1, the initiator need not always send a data portion in the regular message ($MC = 00$). If data is present however, it is encrypted using key, K_S, and included in the optional field (F_O) of $M1$.

$$F_O = E_{K_S}(message) \tag{5.45}$$

- In case of special messages, i.e. regular messages with δ_O ($MC = 01$ or $MC = 10$) or seed update messages ($MC = 11$), the optional field is always present and contains either the unencrypted value of δ_O or the encrypted value of the new seed (the new seed is encrypted using K_S, as summarized in Equation (5.45)).

- Next, the initiator generates the message signature, $msign_I$, using HiveSign.
- At this point, the initiator has in possession the timestamp, any encrypted data and the message signature. This block of data is now encrypted using the outer envelope key, K_O, to generate the first component of the message, $M1$.
- In case of seed update message ($MC = 11$), the initiator stores the generated value of K_O for verification of the acknowledgement sent by the responder. In this case, the outer envelope key is generated using the previous values of t_{s0} so that the responder can decrypt the sent message. The seed is updated following receipt of the acknowledgement message.
- The initiator then attaches the remaining components, 6-bit state variable, s_e and the message code, MC to this encrypted message and transmits it as the message, $M1$.

Step-3: On receiving $M1$, the responder deciphers the message, as follows:

- It first retrieves the message code, MC, and the 6-bit state variable, s_e. Following this, it generates the outer envelope key, K_O, to decrypt the first component of $M1$.
- It then retrieves the three sub-components, i.e. timestamp (t_s), message signature ($msign_I$) and the optional field (F_O - if present). The next step is generation of the encryption key, K_S, to decrypt the intended message (if one is present).
- Based on the type of message, it proceeds with its tasks as follows:
  - In case of a regular message ($MC = 00$), it computes its response, encrypts it using K_S, and generates the responder message signature, $msign_R$.
  - In case of regular messages with δ_O ($MC = 01$ or $MC = 10$), it updates delta, resets the session identifying parameters. Following

this, it computes its response, and encrypts it using K_S, and updates the initial timestamp, t_{s0}, thereby forcing an association renewal.

  – In case of a seed update message ($MC = 11$), it updates the seed as indicated by Table 5.3 and updates the initial timestamp, t_{s0}, forcing an association renewal. Following this, it uses the mechanism summarized by Equations (5.28)—(5.32) to generate a new key, K_S. In this case, since the current timestamp determines the parent seed to be updated, the same parent seed is used to generate the key, K_S, using a random child seed there of, that is also determined by the timestamp. This is only used for acknowledging the receipt and update of the seed. This key, K_S, is then used to encrypt the received timestamp, t_s, which also serves as an acknowledgement pattern. In this special case, the same outer envelope key used to decrypt $M1$ is also used as the outer envelope key while generating $M2$.

- At this time, the responder would have generated its encrypted response, E_{K_S} ($M_{Response}$), which could be the encrypted response or acknowledgement. Following this, it generates the responder message signature, $msign_R$, using HiveSign as described earlier in this section.

Step-4: The responder then assembles the two components and computes message, $M2$, and responds to the initiator.

$$M2 = E_{K_O}\ (\ E_{K_S}(\ M_{Response}\ ) \parallel msign_R\ ) \tag{5.46}$$

Step-5: *Mutual Authentication*: The entities use message signatures ($msign_I$ and $msign_R$) for authenticating each other. On receiving $M1$, the responder computes message signature, $msign'_I$, and attempts to validate the initiator by checking whether $msign'_I = msign_I$. Subsequently, the initiator computes $msign'_R$ and validates the received message ($M2$) and the responder by checking whether $msign'_R = msign_R$.

5.5.4 BeeSwarm: Mechanism for Protection Against Attacks

Protecting an entity against attacks would involve two steps — (a) *identification* of the attack, and (b) *protection* against the identified attack. The conventional and effective form of protection is to preserve confidentiality and integrity of the message using cryptographic techniques. However, cryptographic algorithms might not serve to be effective for protection against attacks such as denial of service (DoS) or replay attacks. HiveSec1 modules serve as means to accomplish security through increased unpredictability in the computed encryption keys. We also go one step ahead and propose a way to protect the entity against an attack in progress. The approach we adopt is *attacking the attacker*, and the technique we propose is called *BeeSwarm*. The "attacking the attacker" philosophy is derived from the concept of bees swarming and attacking any creature in their way, when their hive is attacked or disturbed.

To attack an attacker, however, we would first need to identify that an attack is in progress. The message signature module in HiveSec1 is designed to act as a shield against replay attacks, and DoS attacks. HiveSec1 uses message signatures ($msign_I$ or $msign_R$) to identify any replay attacks, since their computation also includes n_δ and SSN as parameters.

BeeSwarm works based on one key assumption — *if either the initiator or the responder find ε_E consecutive messages to be erroneous, they flag the instance as a possible attack*. The rationale for this is as follows. In an ideal system, communication would be error-free. However, in a real-life application, which need not be as ideal, one cannot ignore external (noise) sources that have the potential to corrupt messages. In HiveSec1, messages are encrypted using keys generated based on the timestamp. This ensures that the same message encrypted on two separate occasions will yield different ciphertexts. Thus, even in an imperfect (non-ideal) scenario, a system implementing HiveSec1 will definitely be able to identify errors, i.e. the very first instance when a sent message is erroneous. Implementations and standards in organizations would dictate the tolerance of such errors, and hence, the value of ε_E. In our implementation, we assume that a value of $\varepsilon_E = 3$ offers a balance (or a threshold) between classifying messages as being communicated on an erroneous channel and classifying the system as being under attack. Thus, if $number\_of\_erroneous\_messages > \varepsilon_E$, HiveSec1 classifies the system as being under attack, and initiates the BeeSwarm protocol.

The BeeSwarm protocol works on the following principle — the entity will flood the attacker with a permitted number (n_B) of non-meaningful responses. Each message will be in the same structure as a normal message on the system, but the data will be pseudorandom numbers, unrelated to either the actual message or keys. We consider the permitted number of messages, n_B, as part of the implementation to accommodate for restrictions that might exist in resource-constrained systems. For example, an RFID tag may be energized only for a specified period of time, or a wireless sensor (or a wireless body sensor) might only be able to stay "alive" and respond only for a specific period of time without exhausting its battery power and/or increasing its surface temperature. Thus, after identifying an attack, the BeeSwarm protocol works as follows:

- The entity uses SSN as a PRNG seed.
- If the entity is an initiator, it generates three random numbers to conform to the syntax of message $M1$, as shown in Equation (5.44). The initiator error response, ϵ_{I_E}, is structured as shown in Equation (5.47).

  $$\epsilon_{I_E} = \epsilon_{I_{E0}} \mid\mid \epsilon_{I_{E1}} \mid\mid \epsilon_{I_{E2}} \tag{5.47}$$

  Here, $\epsilon_{I_{Ei}}$ are the random numbers generated.
- In case the entity is a responder, it generates one random number to conform to the syntax of the response, $M2$, as shown in Equation (5.46). The responder error response, ϵ_{R_E}, is structured as shown in Equation (5.48), where $\epsilon_{R_{E0}}$ are the random numbers generated.

  $$\epsilon_{R_E} = \epsilon_{R_{E0}} \tag{5.48}$$
- The entity then transmits this error response over the channel. It will continue to regenerate the random numbers and re-transmit the appropriate error response up to a maximum of n_B messages.

To summarize the BeeSwarm protocol, a communicating entity first identifies the attack (and hence, the attacker) and bombards the attacker with a reasonably large number of messages (similar to a swarm of bees), thereby turning an attacking attempt into an attack on the attacker itself. The messages generated are simple random numbers, with a new sequence being transmitted as each error response. This, in addition to the fact that PRNG sequence generation is not very resource

Algorithm 6 BeeSwarm Protocol in HiveSec1

```
procedure HIVESEC1BEESWARM
    //Set PRNG seed
    $swarm\_seed \leftarrow SSN$

    if Entity is initiator then
        $swarm\_response \leftarrow g(swarm\_seed) \parallel g(swarm\_seed) \parallel g(swarm\_seed)$
    else if Entity is responder then
        $swarm\_response \leftarrow g(swarm\_seed)$

    $num\_responses \leftarrow$ set based on duration or number of responses

    $response\_count \leftarrow 0$
    while $response\_count != num\_responses$ do
        $SEND\_TO\_SENDER$ ( $swarm\_response$ )
```

intensive, ensures that even a simple device, such as a passive RFID tag, can attack the attacker, thereby attempting to protect the system. Algorithm 6 summarizes the operation of BeeSwarm.

5.5.5 Summary

In this section, we described the HiveSec1 security modules that are used to generate keys, message signatures and to protect a system under attack. To accomplish the desired functionality and achieve unpredictability, we use multiple seeds, PRNG and a mechanism derived from the concept of beehives to choose a combination of seeds to generate keys for encryption. We perceive seeds to exist in the structure of a "seedhive", the size of which is dynamically determined by the timestamp. This key generation process, though simple to illustrate and imagine, introduces complexity for an adversary to guess or crack keys to attack the system. The overall algorithm design is in a way that introduces a perception of complexity, when the actual system concept is simple. This is a deliberate attempt to protect systems implementing HiveSec1 from attacks such as replay and de-synchronization attacks.

HiveSec1 is aimed at improving security through combinations of operations with random choices, be it for choosing seeds to generate keys or to choose different components of messages and keys to generate the signature. Our approach is intended to be low on resource consumption, facilitating its use in resource-constrained devices without significant increase in available resources. Furthermore, our approach modular, in that it is can be deployed with any specific encryption algorithm and hash algorithm, depending on the needs of the application.

5.6 Extending the Proposals to WBAN

RFID tags, especially passive tags, are even more resource-constrained than WBAN on-body sensors. This is primarily because of the absence of an on-chip power source, which limits the operating time of the tag. Our proposals, mainly GeM2 and Butterfly1, were proposed for RFID systems as lightweight (independent) solutions for security. HiveSec1, on the other hand, can be used in either RFID systems or WBAN systems, or in any other network applications, owing to its session update mechanism. In this section, we discuss as to how GeM2, Butterfly1 and HiveSec1 can be extended/adapted to work in WBAN systems.

As discussed in Chapter 1, a WBAN consists of a set of on-body sensors, optional actuators, personal server (intermediary) and the monitoring station. WBAN sensors are connected to a power source (batteries), which enables them to perform the required 'sensing' operation, among other tasks. However, the presence of a power source does not imply that the sensors can be always on, since that would increase the battery consumption and require frequent maintenance (battery replacement). For optimal performance, sensors may be programmed to enter 'sleep' state following message transmission, for a predetermined duration of time [66]. This, in a way, makes them similar to active RFID tags.

Our approaches, GeM2, Butterfly1 and HiveSec1, can be used to protect the communication between the sensor nodes and the WBAN body central node or the hub (*phase 1*), and can also be employed for the communication between the hub and the personal server (*phase 2*). With the personal server and the monitoring station known to have higher computational abilities, we do not consider their security in the purview of our work, although the key generation mechanisms proposed are generic and can be used in that phase of communication as well.

5.6.1 Using GeM2 in WBANs

To use GeM2 in WBAN systems, we envision a scenario in which the hub and the sensor nodes are initialized with the same initial parent key. This reduces the load on the hub of having to store too many parent keys, depending on the number of on-body sensors. One more assumption that we make is that the hub will store the initial parent in a separate memory segment, in addition to the current (synchronized) state parent keys of all on-body sensor nodes. This is to ensure that the hub is able to — (a) regenerate associated parameters to identify any attempts of de-synchronization, (b) demand any future or past state parameters such as *asv* to require a sensor to prove itself in case it detects a potential attack, and (c) generate any parameters demanded by the personal server/monitoring station, during the phase 2 of the communication. Note that the reader role is eliminated in this extension of GeM2 to WBANs.

Any node when transmitting data to the hub in phase 1 will update its keys, encrypt the current data using the generated key, generate the *asv*, and its communication with the hub is governed by the same protocol of operation described in Section 5.3. In case the hub wishes to update the seed of any node, it updates the seed internally first and transfers it to the node, using the last synchronized key and *asv* for encryption and authentication, respectively. The acknowledgement then, will include a key and *asv* generated using the newly updated seed.

Phase 2 communication between the hub and the personal server (or the monitoring station through the personal server) uses the initial parent key that is stored in the hub. While initiating communication, the personal server or the hub will pick a value of p and g at random and generate the corresponding key and *asv*, beginning with the initial parent key as the first parent. Let us consider these values to be $p_{initiator}$, $g_{initiator}$, $k_{initiator}$ and $asv_{initiator}$. The receiver authenticates the sender and validates the session using its stored value of the initial parent key and the received values. Following this, it picks another set of values of p and g to compute the corresponding values. Let us consider these values to be $p_{responder}$, $g_{responder}$, $k_{responder}$ and $asv_{responder}$. On receiving these values, the initiator can authenticate the responder and validate the session. They can communicate using any key that they generate in this manner, throughout the session. If required and if the technology permits, the same scheme of on-demand authentication and key generation can be used in phase 1 as well.

Thus, we can see that GeM2 is not only valuable as a lightweight mechanism in RFID systems, but it can be employed in other resource-constrained environments such as WBANs as well. The ability of GeM2 to be able to generate parameters dynamically and facilitate on demand generation of parameters increases the overall uncertainty of the system, and hence the unpredictability, since keys are never exchanged and seed updates are internal with each message transmission.

5.6.2 Using Butterfly1 in WBANs

As with GeM2, Butterfly1 too can be employed in both phase 1 and phase 2 communication in WBANs. When we consider phase 1, we assume that each sensor node has a specific identifier (ID) that the protocol identifies as η_t and that each node has a specific seed, s_n that is stored at deploy time. The hub stores all these seeds and IDs in memory, and maintains a separate seed for its own communication with the personal server. Note that the reader role is eliminated in this extension of Butterfly1 to WBANs.

Butterfly1 also facilitates either entity to be able to demand the other generate specific parameters for encryption and authentication. This begins with the random choice of j at the initiator and retrieval of the timestamp (t_i). If the initiator is the sensor node, it computes the keys and the message signature based on the value of j and t_i, and transmits the encrypted recorded data to the hub as specified in Section 5.4. Following this, the hub authenticates the node using η_t and j, and sends an acknowledgement on successful authentication, also as specified in Section 5.4. The same mechanism can be applied when used in phase 2 of the WBAN communication, with the personal server and the hub generating specific parameters on demand, as decided by j and t_i.

We can observe that the simplicity of the Butterfly1 algorithm enables us to extend it to use with other resource-constrained application domains such as WBANs with negligible changes/adaptations. If and when required, the size of the seeds (and hence, the keys generated) can be increased to increase the security of the system.

5.6.3 Using HiveSec1 in WBANs

There are three ways in which HiveSec1 can be used in WBANs —

(a) having each sensor node store six parent seeds (as with RFID tags) and the hub store a set of six seeds for each on-body sensor node, and potentially one set for itself (if HiveSec1 is also used for phase 2),

(b) setting up the WBAN nodes to be a cluster in case there are six on-body sensors. In this case, each node would have to store six parent seeds, the hub would also store the same six seeds, in addition to one set of parent seeds for itself, if HiveSec1 is also used for phase 2, or,

(c) a combination of (a) and (b).

In case (a), the communication is governed by the same protocol discussed in Section 5.5, with the feature of dynamic sessions now representing the security association and the message transmission between each node and the hub. The seed updates, if and when required, will also be as discussed in Section 5.5, with a specific seed of a particular node being updated each time. This is a straightforward application of HiveSec1 to WBANs, without the need for any modifications.

Case (b), however, facilitates using only one set of parent seeds for all sensor nodes. This will save memory utilization on the hub, and facilitate creation of a group-wise key for broadcast communication. This would require that an additional key, called the HiveSec1 Group Key, K_G, be generated, as follows — the entities in the group, i.e. the nodes and the hub, combine all the parent seeds, p_i , to generate the group seed, $Seed_G$ (the combination is done using XOR operation). Following this, they use the deploy-time timestamp (t_{s0}), to determine s_g and generate the s_g^{th} pseudorandom number for use as K_G. This is summarized by Equations (5.49), (5.50) and (5.51), respectively.

$$Seed_G = p_1 \oplus p_2 \oplus p_3 \oplus p_4 \oplus p_5 \oplus p_6 \tag{5.49}$$

$$s_g = Integer(\ t_{s0}(3) \parallel t_{s0}(2) \parallel t_{s0}(1) \parallel t_{s0}(0)\ ) \tag{5.50}$$

$$K_G = g_{s_g}\ (\ Seed_G\ ) \tag{5.51}$$

The limitation with this, though will be that if any of the sensors are physically compromised, the whole network of nodes and the hub can be compromised. Physical attacks can not be prevented by cryptographic solutions and separate tamper-proofing mechanisms need to be employed to secure the sensors from such attacks.

A combination of both cases (a) and (b) can also be considered as case (c). This provides a mechanism to not only increase the security of each node, due to the presence of a unique set of parent seeds for each node, but can also facilitate the creation of a group-wise key as discussed earlier. This provides each node the ability to generate the group-wise key by itself, not requiring them to depend on the transmission of such keys by the hub, although a simple extension of case (a) could be to include the group-wise key transmission as a separate message type. In this case though, the message code, $MC = 3(11)$, would need to be modified to set F_O to include the encrypted group-wise key generated by the hub and timestamp, t_s, set to the previous acknowledged t_s value to indicate group-key transmission.

We see that HiveSec1 not only facilitates direct application in WBANs, but also has the potential for modification to support group-wise key generation with added uncertainty, which can be separately generated on demand when both GeM2 and Butteflly1 are used in WBANs.

5.7 Summary

The algorithms discussed in this chapter are independent, in their abilities to perform as standalone algorithms for security in resource-constrained wireless networks, and modular, having the ability for changing one or more of the internal modules (such as encryption or hash algorithms) without affecting the overall design. These algorithms are applicable to RFID systems as discussed in Sections 5.3, 5.4 and 5.5, and can be extended for use in WBAN systems (and other resource-constrained networks) as discussed in Section 5.6. Simplicity in their operation facilitates such easy adaptations to different domains with very minimal modifications, while their mechanisms of key/authentication parameter generation makes the system states unpredictable. This adds to the overall security of the system, in addition to the type of PRNG, encryption and message digest generation algorithms used.

Our thesis aims to include these standalone algorithms as constituents of the framework proposed in Chapter 4, with the timestamps in Butterfly1 and HiveSec1 being used as part of the algorithm choice logic described by Equation (4.2). We discuss their use with our framework and present an overall algorithm of operation of the framework in the next chapter. We also discuss two use cases, discussing how the framework and the constituent algorithms can be applied in those specific applications.

Chapter 6

Synthesis: Integrating the Proposed Concepts

6.1 Overview

In the previous chapters, we have presented a discussion on the new metamorphic framework for security, and its constituent algorithms. In this chapter, we discuss the generic mechanism of using the constituent algorithms discussed in Chapter 5 with the framework presented in Chapter 4. Following this, we will explore two use cases, where the proposed framework or the constituent algorithms can be applied. Our framework is designed so that it can be deployed with its constituent algorithms in any hardware implementation, and the simple mechanism described in Chapter 4 facilitates switching between the constituent algorithms at run time. Furthermore, each algorithm proposed in this thesis is designed to independently secure resource-constrained wireless networks, and therefore can be deployed as standalone algorithms in various systems or collectively as part of the framework.

6.2 GeM2, Butterfly1 and HiveSec1 in the Metamorphic Framework for Security

When GeM2, Butterfly1 and HiveSec1 are used in the proposed framework for security, the algorithm choice logic (discussed in Section 4.2 on Page 45) can be visualized as illustrated in Figure 6.1. The value of n_{Alg} will be set to 3 in the computation of r_{ac} (as defined in Equation (4.1)). The PRNG seed, $seed_{ac}$, would then depend on the ID of the entity (RFID tag, WBAN sensor, RFID server or WBAN hub), and t_0, in this case will be updated as explained in Equation (6.1).

$$t_0 = \begin{cases} t_i & \text{if algorithm used for current transmission is Butterfly1} \\ t_{s0} & \text{if algorithm used for current transmission is HiveSec1} \end{cases} \tag{6.1}$$

Note that the value of t_0 will be updated to the new value as specified in Equation (6.1) following successful acknowledgement during the current transmission. This means

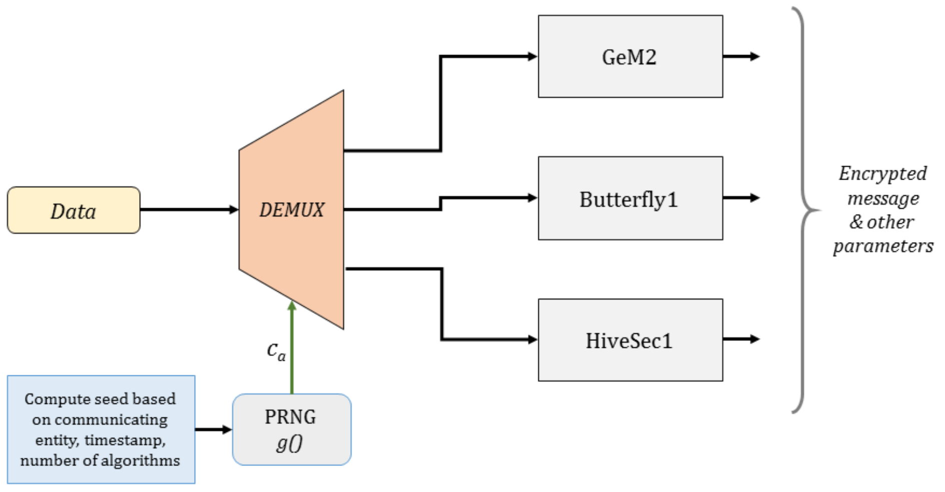

GeM2
Butterfly1
HiveSec1
Encrypted message & other parameters
Data
DEMUX
C_a
Compute seed based on communicating entity, timestamp, number of algorithms
PRNG g()

Figure 6.1: Algorithm Choice Logic with GeM2, Butterfly1 and HiveSec1

that for the current transmission, the key and other parameter generation processes would use the previously stored (and acknowledged) value of t_0, and update this value following acknowledgement of the current message.

When using GeM2, Butterfly1 and HiveSec1 as part of our framework, the protocol of operation is as follows for a communication of the i^{th} message between Alice and Bob (Figure 6.2):

Step-1: Alice retrieves the previously acknowledged and synchronized timestamp, t_0, and increments the stored value of r_{ac}, and computes the algorithm choice, C_a using Equations (4.1), (4.2) and (4.3).

Step-2: The algorithms are chosen as per Equation (6.2).

$$C_a = \begin{cases} 0 & \text{algorithm chosen} = \text{GeM2} \\ 1 & \text{algorithm chosen} = \text{Butterfly1} \\ 2 & \text{algorithm chosen} = \text{HiveSec1} \end{cases} \tag{6.2}$$

Step-3: On choosing one of the available algorithms, Alice computes the encryption keys and associated authentication parameters as explained in Chapter 5.

Step-4: Once messages to be transmitted are assembled as per the protocol of the chosen algorithm, the framework verifies the length of the message to be transmitted (M_{TI}). For uniformity and to ensure uncertainty of the chosen algorithm,

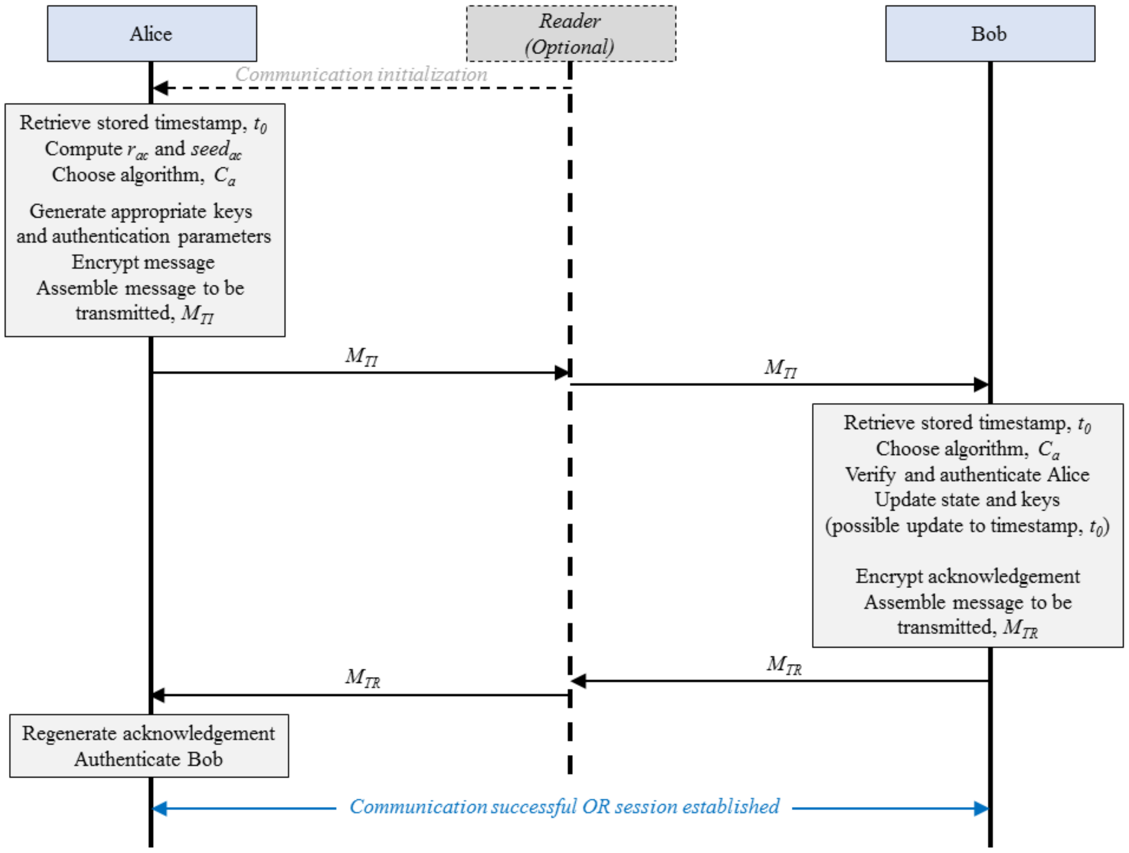

Alice
Reader (Optional)
Bob
Communication initialization
Retrieve stored timestamp, t_0
Compute r_{ac} and $seed_{ac}$
Choose algorithm, C_a
Generate appropriate keys and authentication parameters
Encrypt message
Assemble message to be transmitted, M_{TI}
M_{TI}
M_{TI}
Retrieve stored timestamp, t_0
Choose algorithm, C_a
Verify and authenticate Alice
Update state and keys
(possible update to timestamp, t_0)
Encrypt acknowledgement
Assemble message to be transmitted, M_{TR}
M_{TR}
M_{TR}
Regenerate acknowledgement
Authenticate Bob
Communication successful OR session established

Figure 6.2: Framework: Protocol of Operation

the length of the transmitted message is always considered to be the length (in bits) of the longest message among all the algorithms. In our case, the longest message transmitted is by Butterfly1. We consider this as λ_n. The framework makes a decision on the transmitted message based on Equation (6.3).

$$Transmitted\ Message = \begin{cases} M_{TI} & \text{if } length(M_{TI}) = \lambda_n \\ M_{TI} \parallel g(SSN) & \text{if } length(M_{TI}) < \lambda_n \end{cases} \tag{6.3}$$

Here, $g(SSN)$ is a pseudorandom number generated using SSN as the seed, only to generate pseudorandom numbers (that have no meaning for the algorithms) for padding the message to be transmitted so as to make it be of the same length as the longest message. In case messages in Butterfly1 (or any other algorithm) become longer than λ_n, they are broken into separate messages and transmitted using the same mechanism.

Step-5: On receiving the i^{th} message transmitted by Alice, Bob retrieves the timestamp, t_0 and performs the same computations as Alice to choose the algorithm, generate the keys and authentication parameters. The message integrity verification and entity authentication then proceeds as specified by the chosen algorithm (discussed in Chapter 5). Note that on choosing the algorithm, Bob will be able to discard the additional $g(SSN)$ bits, if the algorithm used is either GeM2 or HiveSec1.

Step-6: If Bob can authenticate Alice, Bob may respond with an encrypted acknowledgement/response as specified by the chosen algorithm, which could lead to session establishment or conclusion of a message transfer. On successful authentication, Alice and Bob have synchronized states, and will be able to continue with their communication.

Step-7: On the other hand, if Bob cannot authenticate Alice, action will proceed as follows. Bob starts an internal counter, to keep track of the number of erroneous messages or failed authentication attempts. If the next attempt results in a successful authentication, the internal counter is reset to 0, and communication proceeds as directed by the chosen algorithm. In case the counter reaches 3, Bob invokes the BeeSwarm algorithm (Section 5.5.4) and responds with the allowed number of meaningless responses, as an attempt to mitigate a potential attack.

Our framework is thus, able to utilize the best possible options from the available algorithms to ensure security. By retaining the length of the transmitted message to be a constant, our framework introduces an additional element of unpredictability to an observer, while by using the BeeSwarm algorithm to attack the attacker (on detection of a possible attack), it is able to use a mechanism not available in GeM2 or Butterfly1 to mitigate attacks. This is also made possible by the re-use and reconfigurability properties of our framework. The overall operation of the framework with GeM2, Butterfly1 and HiveSec1 considered is summarized by Algorithm 7.

With the constituent algorithms being independent each having its own protocol of operation, the framework modules can be conceptualized to be as illustrated in Figure 6.3. Each algorithm will maintain its internal state in a specific segment of the memory, which can be conceptualized to be a separate module. The seed

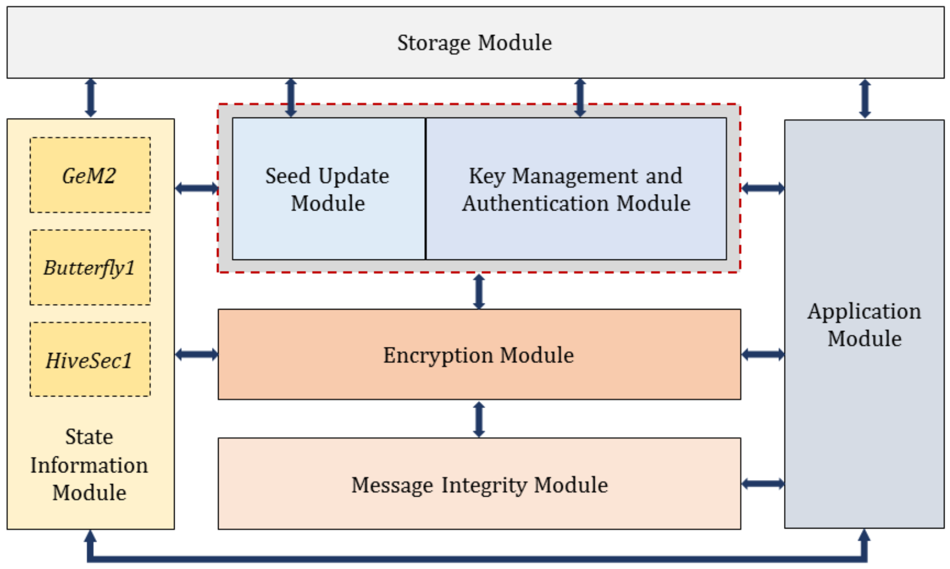


Figure 6.3: Conceptual Illustration of the Framework Modules with GeM2, Butterfly1 and HiveSec1

update algorithms, linear recursive mechanism used in GeM2 and bit-wise seed update mechanism based on the Butterfly effect used in Butterfly1, can be conceptualized to be located with the key management (and authentication) module. Both the seed update module and key management (and authentication) modules would require separate accesses to the memory to store current seed and key states, respectively. The application module, which contains application specific parameters and functionality that generates the data to be encrypted using the keys generated by our approach, can be conceptualized to be able to access the storage, key management, data encryption and message integrity modules for invoking specific functions, as and when required. Such a highly modular approach to security will facilitate reconfigurability and re-use of commonly used functions, to reduce the overall computational overhead, increasing the system efficiency.

Having discussed the operation of the protocol when used with GeM2, Butterfly1 and HiveSec1 in this section, we consider two application use cases in the next section. Under each application, we discuss as to how the framework and/or the independent algorithms can be employed to ensure high security, due to unpredictability.

Algorithm 7 Operation of the Framework (with GeM2, Butterfly1 and HiveSec1)

```
procedure FrameworkOperation
    //Invoke ChooseAlgorithm algorithm in Section 4.2
    $CA \leftarrow ChooseAlgorithm()$

    if $CA = 0$ then
        //Algorithm chosen = GeM2
        $GEM2KeyGenEncrypt()$ //Section 5.3
        //assemble message to be transmitted, $MTI$
    else if $CA = 1$ then
        //Algorithm chosen = Butterfly1
        $BUTTERFLY1KeyGenEncrypt()$ //Section 5.4
        //assemble message to be transmitted, $MTI$
    else if $CA = 2$ then
        //Algorithm chosen = HiveSec1
        $HIVESEC1KeyGen()$ //Section 5.5
        $HIVESEC1Encrypt()$ //Section 5.5
        //assemble message to be transmitted, $MTI$

    if $length(MTI) < \lambda_n$ then $MTX \leftarrow MTI \parallel g(SSN)$
    else if $\lambda_{MTI} = \lambda_n$ then $MTX \leftarrow MTI$
    else if $\lambda_{MTI} > \lambda_n$ then
        //MTX consists of chunks of length, $\lambda_{MTI}$

    if $authentication = failed$ then
        $AuthFailCounter++$
        if $AuthFailCounter = 3$ then
            //Invoke BeeSwarm algorithm in Section 5.5.4
            $HIVESEC1BeeSwarm()$
    else if $authentication = success$ then
        $TRANSMIT\_MESSAGE\ (\ MTX\ )$
```

6.3 Use Case 1: RFID Application for Location Identification and Guidance

6.3.1 Overview

Consider a scenario in which users need to navigate through a manufacturing plant. The manufacturing process, and the intended application of the product X, imposes severe restrictions on the placement of RFID tags on the manufactured products themselves, owing to the use of steel and carbon fibre components. However, to keep a track of the tools used in the manufacturing process, the manufacturing organization places RFID tags on tool kits, with each kit known to have specific tools. Each mechanic is authenticated prior to being given access to the tools, which also depends on the access level of the mechanic.

6.3.2 Application for Location Identification and Guidance

We present an application in which the tools used and the location of the mechanic are used to identify the task, and hence suggest an appropriate path for the mechanic to determine the approximate location of the particular task. The working of this application is as follows:

- The mechanic swipes his/her access card (containing an RFID tag) at location, L_X to access the tools. Here, L_X is obtained by computing a concatenation of latitude (L_{lt}) and longitude (L_{ln}), determined by the mobile device using WiFi access point as reference and the received signal strength indicator (RSSI) to determine the approximate location.

$$L_X = L_{lt} \parallel L_{ln} \tag{6.4}$$

- The mechanic's card is authenticated and his/her access level is determined by the server. On gaining access, the mechanic retrieves the tool kit (containing tools used for the required task).
- The RFID tag on the access card and the reader at L_X communicate with each other using the proposed framework (or an appropriate algorithm) for validation and data encryption.

- We assume that the mechanic has access to either a company approved mobile device, smartphone or a tablet computer, running the application to identify the location and suggest an approximate route for the location of the task. The application is assumed to have a button to generate the route. Attached to the mobile device is a short-range RFID reader that identifies the tools available in the tool kit based on the RFID tag associated with the tool kit, when a route is requested.

- When the mechanic presses the button in the application to generate the route, the server uses the approximate current location of the user, uses the tag on the tool kit to retrieve the location of the task, and dynamically constructs and displays the route from the current location of the mechanic to the location of the task.

In this application, security is an essential component since access to tools must be granted only to authorized mechanics, with organization approved mobile devices. Although the access control protocol itself is outside the purview of our work, we assume that our framework is deployed for securing the communication between the tags on the tool kits and the reader on the mobile device, and between the tag on the access card and the server granting access. We discuss the feasibility of using our framework and/or the constituent algorithms in the paragraphs that follow.

- *Using the proposed framework*: Our framework can be deployed in its entirety, i.e. with all three constituent algorithms, or with algorithm pairs (GeM2-Butterfly1, GeM2-HiveSec1, or Butterfly1-HiveSec1). We expect the communication between the mechanic's access card and the reader at stations granting the mechanic access to specific tools or locations would employ our framework configurations (either with all three algorithms, C3 or with algorithm pairs, C2). We introduced the possibility of using the framework with two algorithms to provide flexibility for use in multiple application environments. We consider that granting access is a more important task in this application domain, and our framework provides the unpredictability required to ensure security of this communication. The uncertainty associated with the choice of the algorithms and the varying internal states of each algorithm in the framework makes this a prime choice for securing this aspect of communication in the location identification and guidance application.

- *Using the constituent algorithms*: The algorithms proposed in this thesis can be deployed as standalone security solutions to secure the communication between the mobile short-range RFID reader and the tags on the tool kits. We suggest the use of one of the possible algorithms and not the framework for this aspect of communication because the tags on the tool kits could be just identity tags [16], but still possess the ability to secure the communication using our lightweight proposals. GeM2 can be deployed if a scheme such as a "rolling code" [89, 90] is required since the key generation begins with one initial key and 'evolves' until a particular state. If on demand parameter generation is desired, any of the proposed algorithms can be deployed on the entities. Note that in this aspect of communication in the application, emphasis is on authentication rather than encryption, since the tags used are identity tags only to identify the tool kits and might not be required to store sensitive information. Furthermore, this is an additional security feature provided following authentication of the mechanic (and hence, the associated mobile device/reader).

6.4 Use Case 2: WBAN Application for Remote Monitoring of High-Risk Pregnancies

6.4.1 Overview

For this use case, we consider the application for remote monitoring of high risk pregnancies that we had previously proposed [125]. We consider a situation in which a pregnant woman, *Ava*, is being remotely monitored by her doctor. She is diabetic, and her doctor, *Ben*, has identified a possible risk of pregnancy induced hypertension and advised her to be on continuous monitoring. However, since she lives in a rural location, she is being monitored by the doctor using a WBAN application, where the doctor can monitor the health of both her unborn baby and her own.

The key aspects of communication in this application are as follows —

- There are sensors to record the Ava's blood pressure, ECG (electrocardiograph), blood glucose and fetal heart rate.
- The WBAN also has a BCU or a hub, which receives the recorded data periodically from the various sensors and transmits it to a personal server (Ava's cell phone). The data is then transmitted to and stored in the monitoring station.
- The WBAN monitoring station is programmed to send periodic updates and emergency alerts (threshold-based), enabling Ben to monitor Ava's health remotely.

Security becomes a critical requirement of this application with the data recorded by the sensors representing Ava's health at a particular instant of time and with them being used to influence medical decisions. As discussed in Section 5.6, any of the algorithms individually can be applied in this scenario. However, we consider the following mechanism to accomplish security in this application — using the proposed framework (C3) in the communication between the hub and the personal server, and using HiveSec1 for securing the communication between the sensor nodes and the hub.

- *Using the proposed framework*: As discussed in the RFID-based location identification and guidance application, the proposed framework (C3, with all algorithms included) is the perfect candidate to secure messages just as they leave the on-body WBAN (from the hub) towards the monitoring station, via the personal server. This is a critical part of the communication because the recorded data is being collected and transmitted to the monitoring station. The framework's ability to dynamically choose an algorithm to generate keys and authentication parameters is suitable for this aspect of communication because that would ensure that only Ava's cellular phone that is authenticated by and synchronized with the WBAN hub will be able to authenticate the WBAN data, and forward it to the monitoring station.
- *Using HiveSec1 for intra-WBAN communication*: We suggest the use of HiveSec1 for securing the intra-WBAN communication, i.e. the communication between sensors and the hub because it has the facility to generate/refresh group-wise

keys, as discussed in Section 5.6. Furthermore, using HiveSec1 allows the hub to either use different parent seed sets for each sensor, or the cluster-based seed storage/key generation mechanism, also facilitating the hub to switch between these arrangements. The facility of automatically updating sessions (and therefore, session keys) enables each sensor to have a dynamically changing session with the hub, increasing the overall security of the WBAN. This is further beneficial since it removes the need for separate session update/change message exchanges between entities, and consequently, reduces the associated overhead.

6.5 Summary

The strength of our proposals, as noted here is in the options available for deployment in different applications. Depending on the needs of various aspects of the application, different elements of security may be deployed. If the communication requires only authentication of entities with the devices not performing any data exchange/encryption, any individual algorithm might be used for generating the authentication parameters. The framework could also ideally be deployed in such a scenario if resources were available. However, the framework or any constituent algorithm may be used in any scenario involving exchange of sensitive information that needs data to be encrypted and entities authenticated; although we envision the framework being applied in such scenarios owing to the increased uncertainty and consequently, security.

6.5.1 Revisiting our Hypotheses

Our framework and the constituent algorithms were designed with the goal of making resource-constrained devices more ‘trustworthy’, by giving them the ability to generate parameters synchronous with a verified server. Our hypotheses, presented in Chapter 3 (page 41), make broad predictions about the behavior of the framework and its constituent algorithms. To test these hypotheses, we will use the following assessments:

- We will evaluate algorithm choice ($H1$) by a proof of concept evaluation of the proposed framework (and constituent algorithms), with random updates in states and determining the algorithm used. We anticipate that the system state updates will justify our claim and each algorithm will be chosen with a probability of $\frac{1}{n_{Alg}}$ in the total number of trials. Here, n_{Alg} is the number of algorithms.

- We will evaluate the dynamic nature in key generation ($H2$) also using a proof of concept evaluation of the framework (and constituent algorithms), and quantify this measure using a uniqueness factor, υ_{ns}, which is a measure of the number of new keys generated in a given number of messages. We quantify it as specified by Equation (6.5). We expect that the uniqueness factor will be 100% for the main encryption keys in the algorithms.

$$\upsilon_{ns} = \frac{number\_of\_new\_keys}{total\_number\_of\_messages} \tag{6.5}$$

- We evaluate key unpredictability ($H3$) using key similarity evaluation (with similarity quantified by a number in the range 0.0 – 1.0) and key sequence randomness assessment (based on the ability of the generated sequence to have near equal presence of '1' bits and '0' bits). We anticipate that the keys will be less similar to subsequent keys and that the key sequences will satisfy the randomness criteria.

- In our proof of concept implementations, we will introduce an adversary entity, which will inject messages previously transmitted (replay), attempt to inject modified messages (data modification) and sending multiple requests (DoS), and we expect each algorithm (hence, the framework) to detect such attacks. This is our attempt to test our security hypothesis ($H4$). We expect our proposals to pass the security assessment, be able to achieve security objectives and be able to detect all considered attacks.

- We will evaluate resource utility by our proposals ($H5$) by deploying them on a reconfigurable FPGA (Field-Programmable Gate Array), which helps in assessing the amount of resources required by the proposals on Application Specific Integrated Circuits (ASICs) in terms of an approximate gate count and logical resources required for their implementation. Since our proposals employ bitwise logical/simple arithmetic operations, we expect that our proposals are efficient in terms of resource utility, and do not impose significant overheads for memory, storage or computation.

Our framework and the constituent algorithms accomplish security through unpredictability introduced by strategically combined logical operations. We discussed this in the individual proposals of our algorithms and the framework, and we also discussed their possible (adaptation and) applicability in real-time applications presented in this chapter. In the next chapter, we present the mechanisms we have considered to evaluate our proposals and discuss the results obtained.

Chapter 7

Evaluation and Results

7.1 Overview

This chapter summarizes the methodologies used to evaluate our biomimetic metamorphic framework proposal for security in resource-constrained wireless networks and the results obtained. We first discuss the proof of concept implementation and the results obtained that help us ascertain our claims of unpredictability, followed by a detailed security analysis, and by summarizing the resources utilized by our framework as a whole and the individual constituent algorithms.

7.2 Proof of Concept Implementation

In order to be certain of the security offered by the keys generated by each algorithm independently and the framework, we needed to establish that the concepts would indeed perform as intended. Specifically, we accomplished this by:

- verifying the concept of each algorithm and the framework as a whole;
- evaluating algorithm choices (for the framework);
- verifying dynamic key generation (for the individual algorithms and the framework);
- evaluating key similarity (using Sörensen's Similarity Index (SSI), for the individual algorithms and the framework); and,
- assessing randomness associated with the keys.

7.2.1 Implementation Details

We implemented the functionality of each constituent algorithm and the framework using Java programming language [126], and used Eclipse integrated development environment (IDE) [127] for development. Table 7.1 summarizes the characteristics and environment of the computer used for the proof of concept implementation.

Our implementation included a separate class file for each algorithm and for the framework, and independent methods for accomplishing each function of the algorithm/framework. Modularity is a central aspect in our proposal and we based this implementation on the same concept. A separate file was used to manage static constants. We used one test wrapper file to test all aspects of our proposal. We executed the program to extract 10240 keys from the configurations summarized in Table 7.2. While configurations $C1 - C3$ and $C4$ were mainly to evaluate each individual algorithm and the overall framework, respectively, configurations $C5 - C7$ were used to explore the performance of the framework when only two of the proposed constituent algorithms were considered (all possible combinations) instead of all three together. We performed this assessment because we wanted to explore the performance of our framework in a scenario where the system deploying our framework required high unpredictability, but did not necessarily possess the resources to deploy all three algorithms. In all configurations where GeM2 is a component, the initial key was set to $92EB8D6ECF7F808A705D1A4566991AF0$, the initial seeds to compute the PRNG seed were set to 14930352 and 24157817. In all configurations where Butterfly1 is a component, the PRNG used to choose the value of the variable j at random, which decides the state of the seed (s_j), was initialized to $192BC333250CCCFF$, while the seed (s) itself was initially set to 12345678. For our HiveSec1 implementation, the six parent seeds of the 'seedhive' were initialized to $21365448FEA32DE0$,

Table 7.1: Characteristics and environment of the computer used for the proof of concept implementation

Processor	Intel(R) Core (TM) i7-3632QM CPU @ 2.20 GHz
Installed memory (RAM)	16.0 GB (15.9 GB usable)
System type	64-bit Operating System, x64-based processor
Operating System	Windows 8.1
Java Development Kit (JDK) version	1.7.0 update 40
Eclipse version	4.3 (Kepler)

Table 7.2: Configurations tested using the proof of concept implementation

Configuration	Description
$C1$	GeM2 algorithm
$C2$	Butterfly1 algorithm
$C3$	HiveSec1 algorithm
$C4$	Framework, containing GeM2, Butterfly1 and HiveSec1 algorithms
$C5$	Framework, containing GeM2 and Butterfly1 algorithms
$C6$	Framework, containing GeM2 and HiveSec1 algorithms
$C7$	Framework, containing Butterfly1 and HiveSec1 algorithms

$F40925AB6B446781$, $E82745AEF95112DDC$, $67A8366EFC8CC294$, $48656CBCEAA36291$ and $998163AFE2A88A0A$, while the seed for the PRNG when used to choose the the value of s_e in HiveSec1 was set to $12345678FEDCBA98$.

While generating the keys, we introduced random delays (0 to 2 seconds) between key generation cycles, in order to emulate real-time data communication, and introduced random key "refresh" cycles, forcing seed updates and system state changes. We then used the generated keys to assess the various aspects of our proposal. Furthermore, to programmatically extract the system timestamp, we used the standard Java method, $System.currentTimeMillis()$. It is also to be noted that we used the standard Java definitions of *Random* functions (defined in the Random class) for extracting pseudorandom sequences as and when required. The Random class in Java also facilitates setting the value for the seed, which is a requirement for all our algorithms. For the proof of concept, we also used SHA-1 (Secure Hash Algorithm) as the hash algorithm, with a slightly modified implementation based on the definition found in the SHA-1 online website [128], since our proposals are independent of the type of PRNG or hash algorithm used.

Preliminary inspection of the program outputs justified that the algorithms (and the framework) were performing as they were expected to, which led us to continue with further assessment, which we describe in the following subsections.

7.2.2 Evaluation of Algorithm Choices

Methodology

In evaluating the proposals, specifically the framework (configurations C4 – C7), we also logged the algorithm choices computed by the algorithm choice logic (Equations (4.1), (4.2) and (4.3) in Chapter 4). We anticipated that the system state updates cause each algorithm to be chosen with a probability of $P(Alg)$ in the total number of trials.

$$P(Alg) = \frac{1}{n(Alg)} \tag{7.1}$$

Here, $n(Alg)$ is the number of algorithms.

Results

Table 7.3 summarizes the count (key count) of each algorithm key generated by the frameworks in a total of 10240 keys and the percentage of the total number of keys the count represents. In our hypothesis (H1), we stated that each algorithm would be equi-probable, with the algorithm choice probability represented by Equation (7.1). For configuration $C4$, the number of algorithms is 3, giving a probability, $P(Alg, C4) = 33.33\%$. For configurations $C5 - C7$, the number of algorithms is 2, giving a probability, $P(Alg, C5) = P(Alg, C6) = P(Alg, C7) = 50\%$. From Table 7.3, we can observe that the actual key count percentages obtained are very close in value to the expected probabilities, with an average error of $\pm 1.00\%$.

Figures 7.1 — 7.4 graphically illustrate the algorithm choices for each key generation cycle, summarized in Table 7.3. In these illustrations, the vertical axis represents algorithms, where Algorithm 1 is GeM2, Algorithm 2 is Butterfly1 and Algorithm 3 is HiveSec1. Figures 7.1(a), 7.2(a), 7.3(a) and 7.4(a) represent the algorithm choices for all 10240 keys in the respective configurations, while Figures 7.1(b), 7.2(b), 7.3(b) and 7.4(b) illustrate a part (108 keys) of the total 10240 keys. We have included the latter for each configuration to highlight the variation in key choices. Although the key generation percentages are close to the expected probabilities, it can be noted that the variation in algorithm choices, and hence, unpredictability is more in configuration $C4$, while the amount of variations observed in configurations $C5 - C7$ are lesser. Nevertheless, since the algorithms are chosen using a pseudorandom number

Table 7.3: Summary of algorithm choices by the proposed framework (configurations C4 – C7)

Configurations	GeM2	Butterfly1	HiveSec1	Totals
C4 (Key count)	3351	3446	3443	10240
C4 (Percentage)	32.72%	33.65%	33.62%	100%
C5 (Key count)	4917	5323	—	10240
C5 (Percentage)	48.02%	51.98%	—	100%
C6 (Key count)	5230	—	5010	10240
C6 (Percentage)	51.07%	—	48.93%	100%
C7 (Key count)	—	5103	5137	10240
C7 (Percentage)	—	49.83%	50.17%	100%

generator, algorithm choices are still unpredictable in configurations $C5$ – $C7$ although the variation is less. Note that the illustrations shown in this section are only to illustrate the variation in algorithm choices, and do not imply any specific ordering of algorithms, since the choice of algorithms is dependent on the entity initiating the communication and the time (indicated by timestamp).

In Figure 7.2, the algorithm choice is only between GeM2 and Butterfly1 (configuration $C5$), in Figure 7.3, the algorithm choice is only between GeM2 and HiveSec1 (configuration $C6$), and in Figure 7.4, the algorithm choice is only between Butterfly1 and HiveSec1 (configuration $C7$).

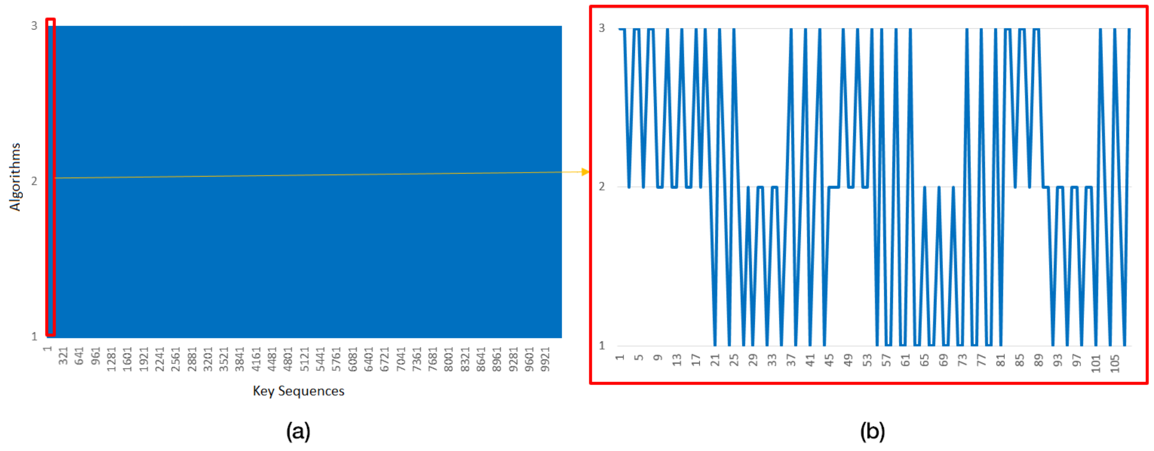


Figure 7.1: Variation in algorithm choices for Configuration $C4$, when considering all 10240 key sequences generated (a), and only 108 key sequences (b)

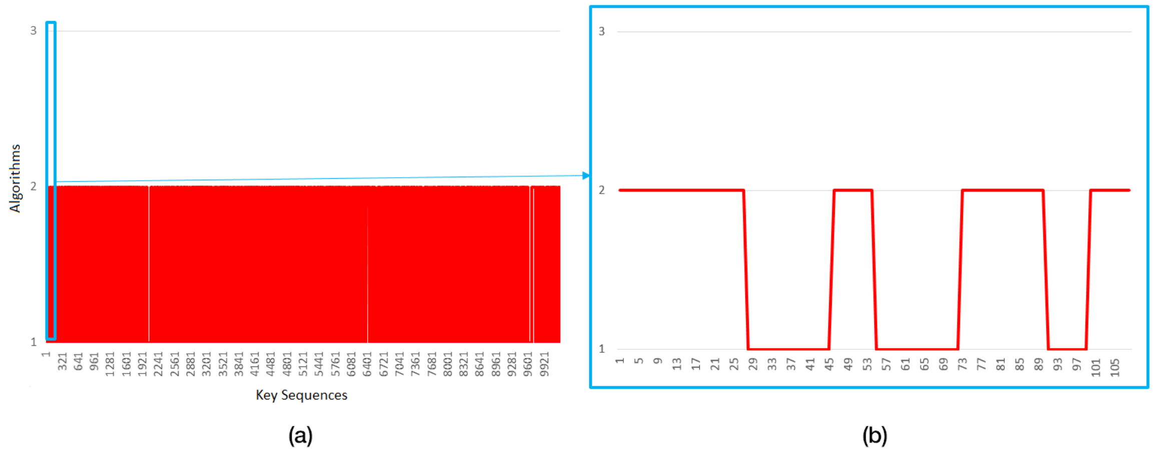


Figure 7.2: Variation in algorithm choices for Configuration $C5$, when considering all 10240 key sequences generated (a), and only 108 key sequences (b)

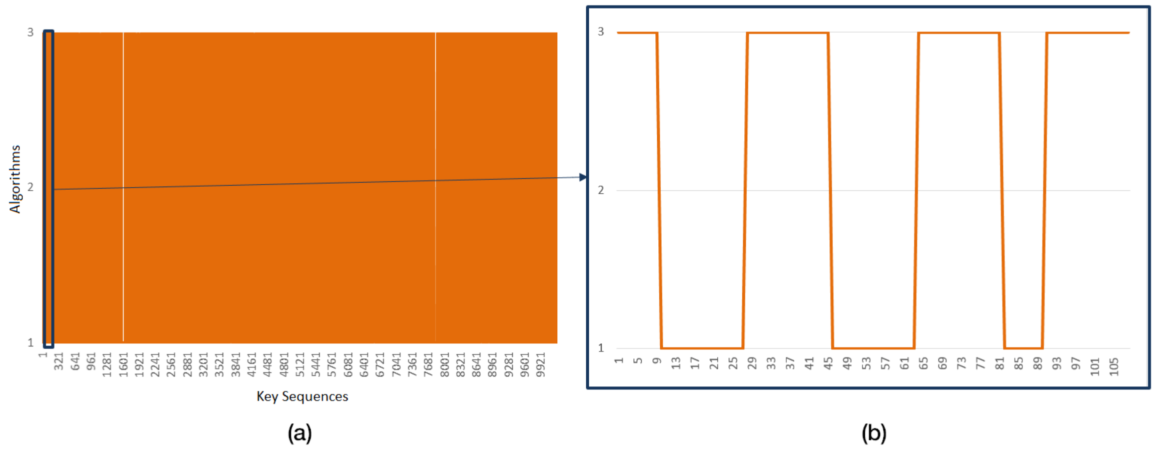


Figure 7.3: Variation in algorithm choices for Configuration $C6$, when considering all 10240 key sequences generated (a), and only 108 key sequences (b)

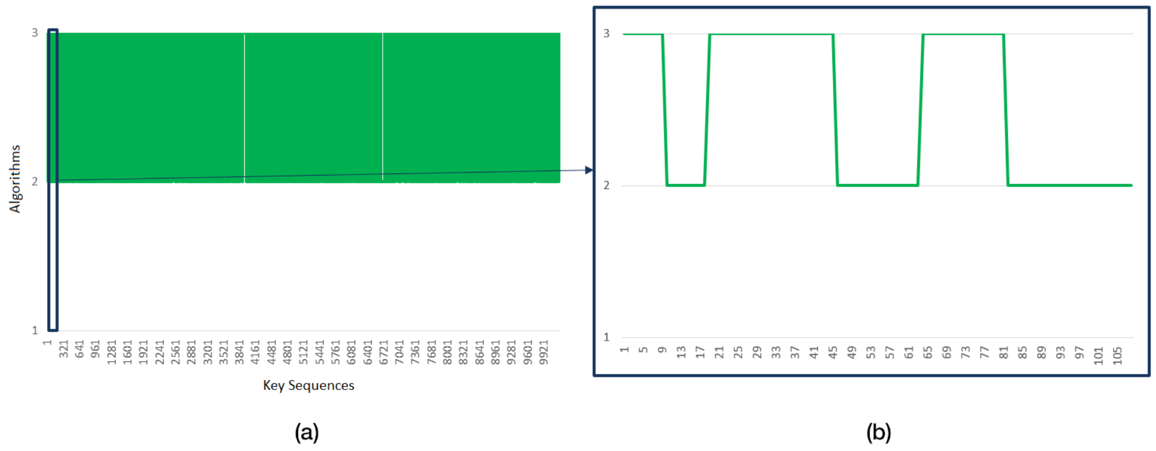


Figure 7.4: Variation in algorithm choices for Configuration $C7$, when considering all 10240 key sequences generated (a), and only 108 key sequences (b)

One thing to note is that even though the algorithm choices are unpredictable to an unauthorized entity observing the communication (owing to changing internal states of the variables used), it still remains deterministic to an authorized entity synchronized with the sender, which facilitates mutual authentication, and key agreement without the need for exchanging keys.

7.2.3 Uniqueness of Keys Generated

Methodology

To determine whether keys are unique, we compared the generated key streams for repeated keys. Our expectation was that the system would generate unique keys for each communicated message. Specifically, we expect this behavior for main encryption keys such as, the new key, K, in GeM2, encryption key, K_i in Butterfly1, and the encryption key, K_S, in HiveSec1, and a combination of these keys in the framework configurations, $C4$ — $C7$. We did not expect the seed s_j and transfer key, K_T in Butterfly1 to exhibit the same behavior since we expect them to change as decided by the server; similarly for the outer envelope key, K_O, in HiveSec1, which we expect to be the same for the duration of an automatically updating session. We quantified this measure using a uniqueness factor, υ_{ns}, which is a percentage measure of the number of new keys generated in a given number of messages (Equation (6.5) on Page 111).

Results

Table 7.4 summarizes the uniqueness assessment for each configuration we tested. The table includes two sections, one summarizing the behavior of each configuration for all 10240 keys and the second summarizing the behavior for the initial 108 keys. We consider the latter as it helps better understand how each configuration generates unique keys and whether there are instances when certain configurations generate the same key, i.e. to examine any unexpected behavior. We illustrate the former case (uniqueness behavior in 10240 keys) in Figure 7.5 and the latter case (uniqueness behavior in 108 keys) in Figure 7.6.

Our expectation for this assessment was that the various configurations generate as many unique keys as possible, ideally, 10240 unique keys. We observe that the main keys, i.e. K in GeM2 (C1), K_i in Butterfly1 (C2) and K_S in HiveSec1 (C3), and their combination configurations (framework), $C4(K, K_i, K_S)$, $C5(K, K_i)$, $C6(K, K_S)$ and $C7(K_i, K_S)$ have an average of 10237.28 $\approx$ 10237 unique keys out of 10240 (or, 99.97%).

When we examine the algorithms individually, we observe that K in GeM2 (C1)

Table 7.4: Summary of key uniqueness

Configuration (Keys considered)	Uniqueness Measure (Summary, 10240 Keys)		Uniqueness Measure (Initial 108 Keys)	
	Key count	υ_{ns}	Key count	υ_{ns}
C1 (K)	10240	100.00%	108	100.00%
C2 (s_j)	7656	74.77%	83	76.85%
C2 (K_i)	10235	99.95%	108	100.00%
C2 (K_T)	6258	61.11%	68	62.96%
C3 (K_S)	10237	99.97%	108	100.00%
C3 (K_O)	9585	93.60%	102	94.44%
C4 (K, K_i, K_S)	10239	99.99%	108	100.00%
C4 (K, K_T, K_O)	9909	96.77%	100	92.59%
C5 (K, K_i)	10238	99.98%	108	100.00%
C5 (K, K_T)	8179	79.87%	81	75.00%
C6 (K, K_S)	10236	99.96%	108	100.00%
C6 (K, K_O)	9920	96.88%	104	96.30%
C7 (K_i, K_S)	10236	99.96%	108	100.00%
C7 (K_T, K_O)	7982	77.95%	77	71.30%

generated the highest number of unique keys, 10240 / 10240 (100.00%), a behavior expected from all main key configurations. We expect this from the main key configurations because the transfer and outer envelope keys in Butterfly1 and HiveSec1, respectively, are dependent on the internal states of the updated seed s_j and the session identifier, n_δ, respectively. However, the unique key shortfall for both K_i (Butterfly1) and K_S (HiveSec1), and the framework configurations, can be attributed to our use of a random delay between successive key generation cycles. This random choice between 0 and 2 seconds between successive key generations means that the timestamp could have been the same in case the delay was 0 seconds, making the keys identical. This is acceptable since real-time delays between key generations (and message communications) can be expected to be more than 0 seconds. Since this was the belief, one could wonder why we chose to include 0 seconds within the range of acceptable random delays in the proof of concept and our assessment. This was to take into account the off-chance of two or more messages encrypted in rapid succession.

For additional key configurations, i.e. $C2(s_j)$, $C2(K_T)$, $C3(K_O)$, $C4(K, K_T, K_O)$, $C5(K, K_T)$, $C6(K, K_O)$ and $C7(K_T, K_O)$, the lack of near 100% uniqueness can be

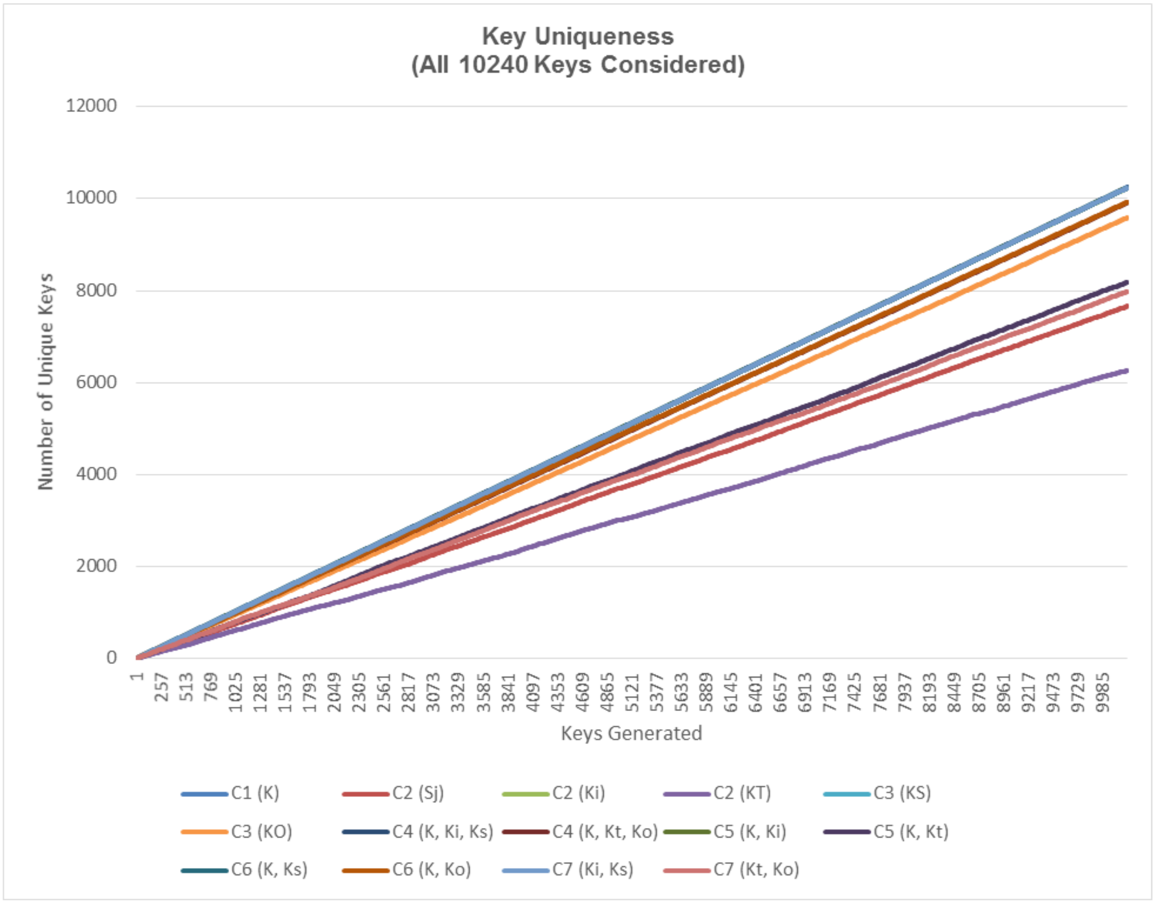


Figure 7.5: Plot illustrating patterns in unique keys generated, per 10240 keys in all configurations

attributed to many factors. First and foremost, random choice of j — s_j is the seed that is modified as per the sender's instruction. Since the choice of the variable j was random, it could be the same for consecutive key generation cycles. This also influences K_T, since it is a pseudorandom number generated using s_j as the seed. Secondly, the choice and generation of K_O is based on automatic session updates, as specified in HiveSec1. This behavior is consistent with the framework configurations $C4(K, K_T, K_O)$, $C5(K, K_T)$, $C6(K, K_O)$ and $C7(K_T, K_O)$, where transfer and internal keys are considered.

Although these transfer and internal keys could remain constant, their state updates remain unpredictable and the continuous updates to the encryption keys ensure that the overall encrypted message remains unpredictable. Another layer of unpredictability is added by the framework itself, which chooses one of these algorithms at

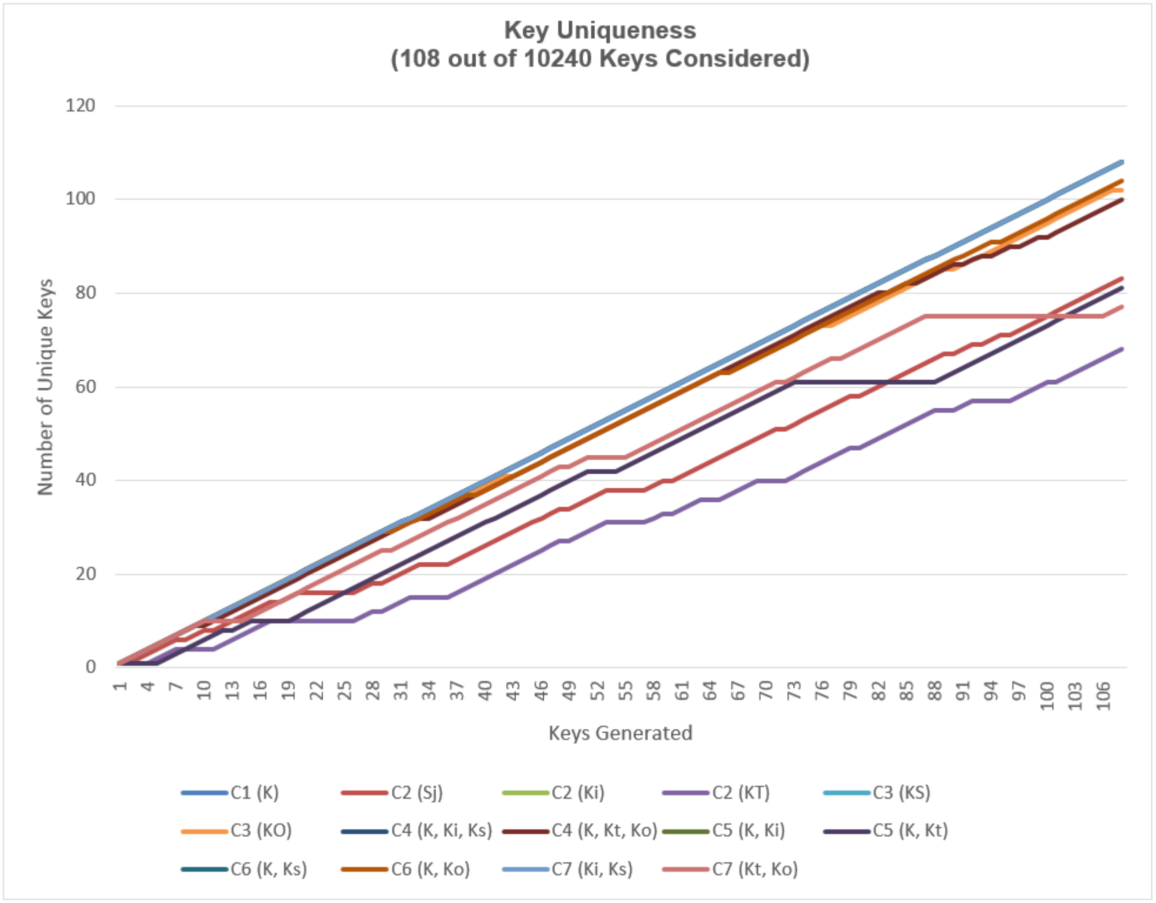


Figure 7.6: Plot illustrating patterns in unique keys generated, per 108 / 10240 keys in all configurations

random for each key generation/message encryption cycle.

7.2.4 Evaluation of Similarity Between Keys

Methodology

According to Kerckhoff's principle [31], the knowledge of the operational specifics of a cryptosystem (except the key) must not reduce its security. Thus, it is essential to ensure that there are no apparent patterns in the encryption keys that could compromise the security of the cryptosystem. This led us to verify how similar consecutive keys generated by each algorithm and the framework are, since our approaches use varying logical operations to accomplish the desired functionality. The other reason why this assessment is necessary is due to the approach of GeM2 to use a 'parent key' and derive future keys from that. To evaluate similarity between keys, we considered keys generated by the system and compared pairs of keys. We used Sörensen's Similarity Index (SSI) [129] to quantify the amount of similarity between consecutive keys. SSI is a measure of how similar the various pairs of keys are, i.e. it is the ratio of twice the total similar characters in the two keys to the total size (in characters) of each key. Equation (7.2) summarizes the computation of SSI.

$$SSI = \frac{2 \times n(A \cap B)}{n(A) \ + \ n(B)} \tag{7.2}$$

Here, $n(A \cap B)$ represents the number of characters (or, numbers) in the key pair that are same, $n(A)$ and $n(B)$ represent the total number of characters (or, numbers) in each of the keys A and B of the key pair, respectively.

We computed SSI for each pair of keys, and compute the average SSI for each configuration highlighted in Table 7.2. Furthermore, we divided each key into 32-bit blocks (*key substrings*) and computed the SSI between each such block for the same pair of keys. When comparing key substrings, we refer to the most significant 32-bit block (bits 96...127) as $substring1$, the next block (bits 64...95) as $substring2$, the block (bits 32...63) as $substring3$, and the least significant 32-bit block (bits 0...31) as $substring4$.

To understand how SSI computation is useful in assessing the security of keys, let us consider three example cases — (a) non-ideal, (b) ideal and (c) average, with 16-bit keys, i.e. $n(A) = n(B) = 16$. In the non-ideal case, let us consider that the two consecutive keys are identical, e.g. $Key1 = Key2 = 0123456789ABCDEF$, implying that $n(A \cap B) = 16$. From Equation (7.2), we get $SSI_{non-ideal} = 1.0000$. In the ideal case, consider that $Key1 = 376533A30ACC7653$ and $Key2 = F124489BDEF14FEB$, where both $Key1$ and $Key2$ are unequal, implying that $n(A \cap B) = 0$, and therefore, from Equation (7.2), we get $SSI_{ideal} = 0.0000$. In the average case, consider that $Key1 = 010124AC67899736$ and $Key2 = 5347BDEEA4330FFF$, as might be the case when keys are generated using (pseudo)random number generators. In this case, the unique characters in $Key1$ are $\{0, 1, 2, 3, 4, 6, 7, 8, 9, A, C\}$ and the same in $Key2$ are $\{0, 3, 4, 5, 7, A, B, D, E, F\}$, which results in $n(A \cap B) = 5$. From Equation (7.2), we compute $SSI_{average} = 0.3125$.

There are 32 characters, i.e. combinations of hexadecimal digits $(0...9, A...F)$, in total in a 128-bit key. Since (pseudo)random number generator output characters are equi-probable, i.e. have equal probability of being one of all possible outputs, it is reasonable to expect that there can be at least 10 characters that can be found in both keys, despite them being different. If we substitute $n(A \cap B) = 10$ and $n(A) = n(B) = 32$ in Equation (7.2), we get an SSI value of 0.3125. Due to this reason, our expectation was that both the framework and the individual algorithms will generate dissimilar keys (when considered wholly or as 32-bit blocks), with an average SSI value in the vicinity of 0.30, as we would need the similarity between keys to be sufficiently low (at least less than 0.50) in order for the keys to be secure, and the encrypted data to be safe from attacks. Low values of SSI also means that the number of repeating characters in the key sequences is lesser, which also adds to the security.

Results

Table 7.5 summarizes the average SSI values for 10240 keys for all configurations, when keys are considered wholly (128 bits) or as four 32-bit blocks. Figure 7.7 illustrates these average SSI values, from which we can observe that the SSI for full keys (128 bits) lie in the range $0.30 \leq SSI_{av,fullkey} \leq 0.40$, except for $C2(s_j)$, which has a full key SSI value of 0.2092. Low average values of SSI indicate that the keys generated by our algorithms and the framework are largely dissimilar. Next, we discuss the similarity of each configuration, considering both full keys and 32-bit blocks in each, and illustrating the variation in similarity between successive keys. It has to be noted that the illustrations for each configuration contains only the initial

Table 7.5: Summary of similarity between keys

Configuration	Average SSI (SSI_{av})				
(Keys considered)	Full Key	Substring1	Substring2	Substring3	Substring4
C1 (K)	0.3040	0.3371	0.3315	0.3307	0.3312
C2 (s_j)	0.2092	0.1250	0.1250	0.0936	0.7178
C2 (K_i)	0.3809	0.3279	0.3237	0.3277	0.3252
C2 (K_T)	0.3833	0.5328	0.6167	0.6951	0.6271
C3 (K_S)	0.3816	0.3271	0.3246	0.3259	0.3254
C3 (K_O)	0.3847	0.3516	0.3587	0.3568	0.3526
C4 (K, K_i, K_S)	0.3514	0.3125	0.3067	0.3109	0.3103
C4 (K, K_T, K_O)	0.3406	0.3069	0.3432	0.3624	0.3434
C5 (K, K_i)	0.3437	0.3285	0.3274	0.3280	0.3288
C5 (K, K_T)	0.3437	0.4314	0.4767	0.5170	0.4806
C6 (K, K_S)	0.3415	0.3287	0.3271	0.3280	0.3263
C6 (K, K_O)	0.3442	0.3453	0.3426	0.3449	0.3420
C7 (K_i, K_S)	0.3812	0.3262	0.3249	0.3255	0.3263
C7 (K_T, K_O)	0.3827	0.4391	0.4801	0.5195	0.4878

108 comparisons, as a way to illustrate the variability clearly within the confines of a letter-sized page.

Figure 7.8 illustrates the SSI variation in the initial 108 comparisons of keys generated by configuration C1 (GeM2). The variability in SSI is high, being centred around 0.3000, which is as expected. In the keys generated by GeM2, the child keys are expected to have the same base pattern bit as the parent key, with changes (mutations) introduced in the other bits. This is one of the reasons why certain successive keys appear to have increasing similarity with the previous keys, or the similarity appears to “grow” to a certain point. However, GeM2 includes a mechanism of random choice between generating a child key or a parent key refresh, which ensures variability in SSI and unpredictability in key generation, thus adding to the security.

This variability can also be observed in the substring comparisons, representing comparisons of the four 32-bit blocks of the keys. However, the observed similarity has a larger range, $0.0000 \leq SSI_{substrings} \leq 0.6000$, with a few outlying values beyond 0.6000. This can be attributed to the reduced size of the block under comparison.

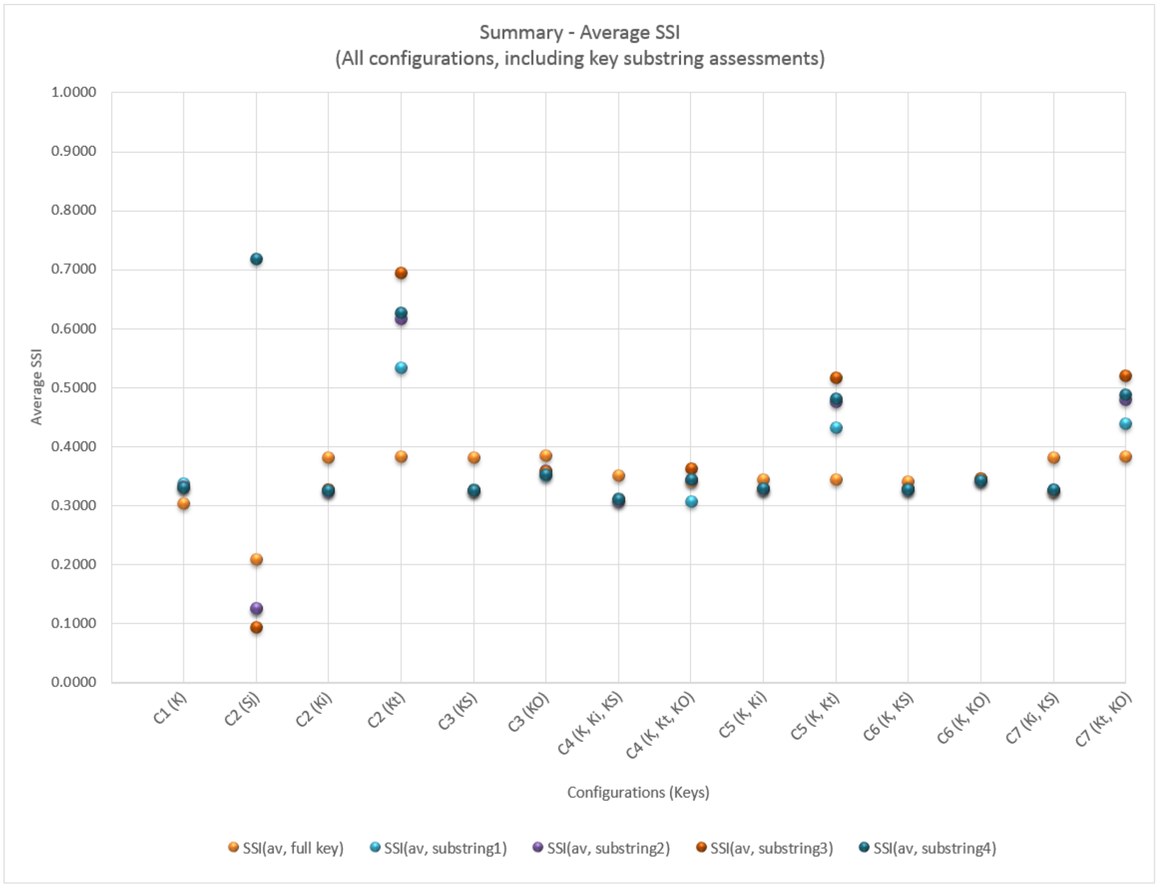


Figure 7.7: Variation in average SSI for all configurations

There are 8 characters in a 32-bit block and the more similar characters there are in the blocks under comparison, the more the similarity between them. However, increased variability in the similarity coefficients between successive keys (as observed in the illustration) implies that even though the SSI range increases, the security is not reduced.

Figures 7.9, 7.10 and 7.11 illustrate the variation of SSI in configuration C2 (Butterfly1). One aspect that stands out when observing Figure 7.9 is the lack of variation in $substring1$ and $substring2$. This is because the seed, s_j, in the our proof of concept implementation is a 64-bit number, which left the upper half of the 128-bit generic SSI comparisons as 0s (note that all our assessments were performed with 128-bit keys, and any lesser sized key was zero padded prior to the assessment). With all characters being 0s, the SSI is constantly 0.1250. $substring3$ and $substring4$ vary, albeit not considerably, as decided by a random choice of the state change variable, j

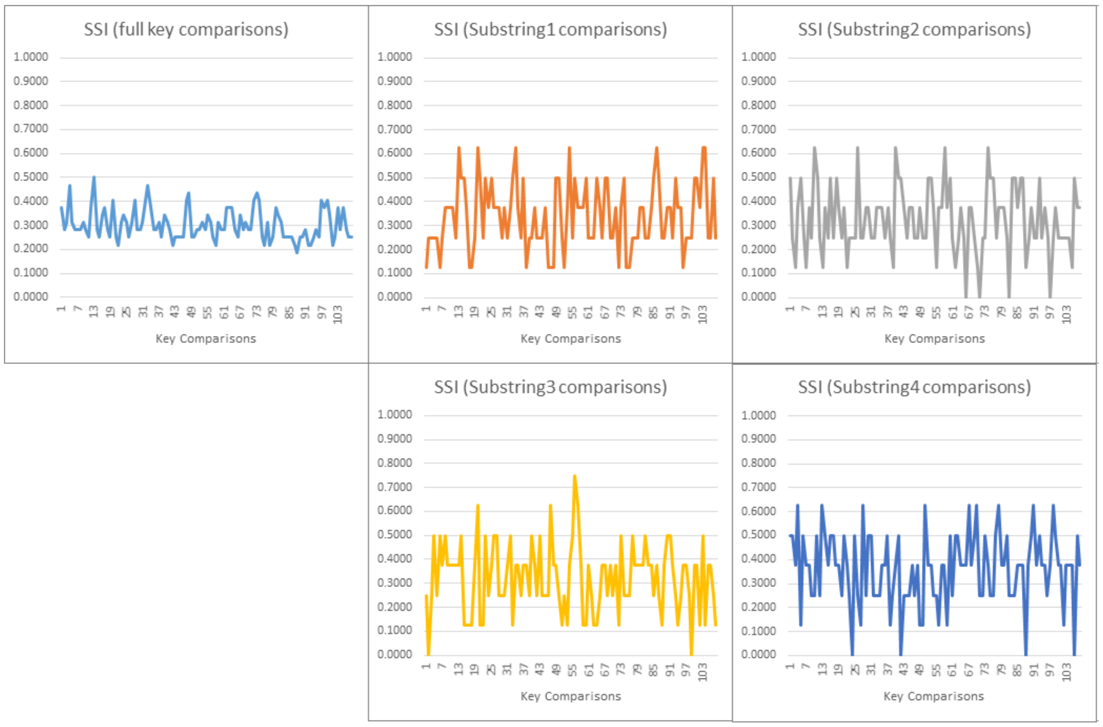


Figure 7.8: SSI Variation (Configuration $C1$)

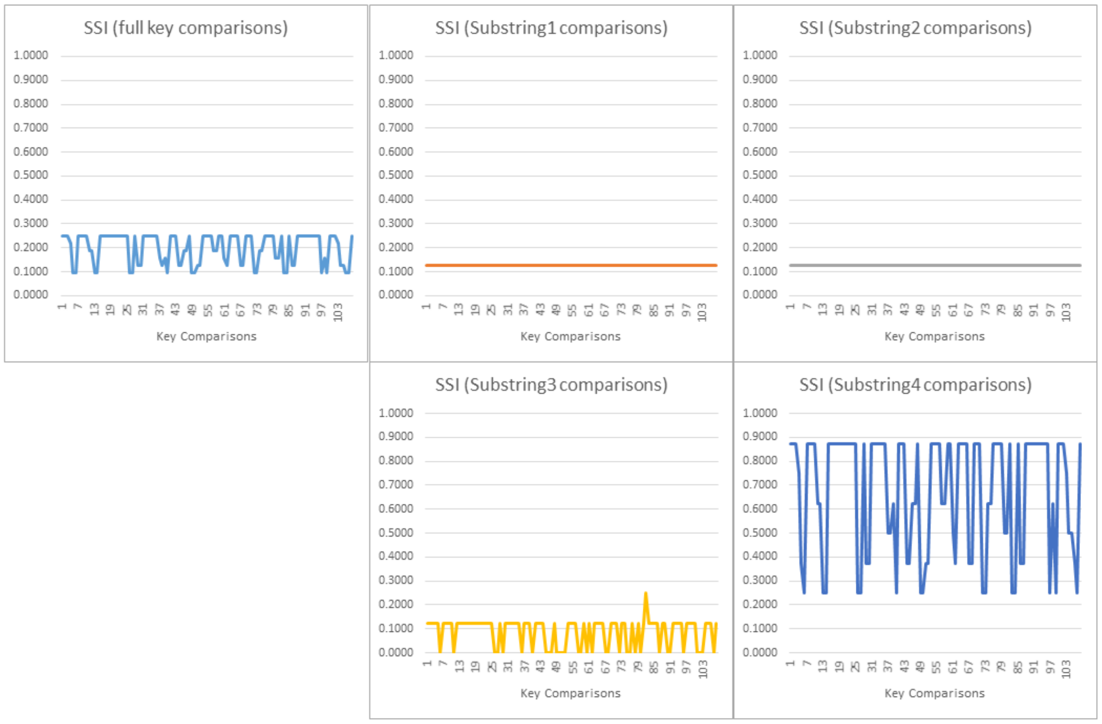


Figure 7.9: SSI Variation (Configuration $C2$, $key = s_j$)

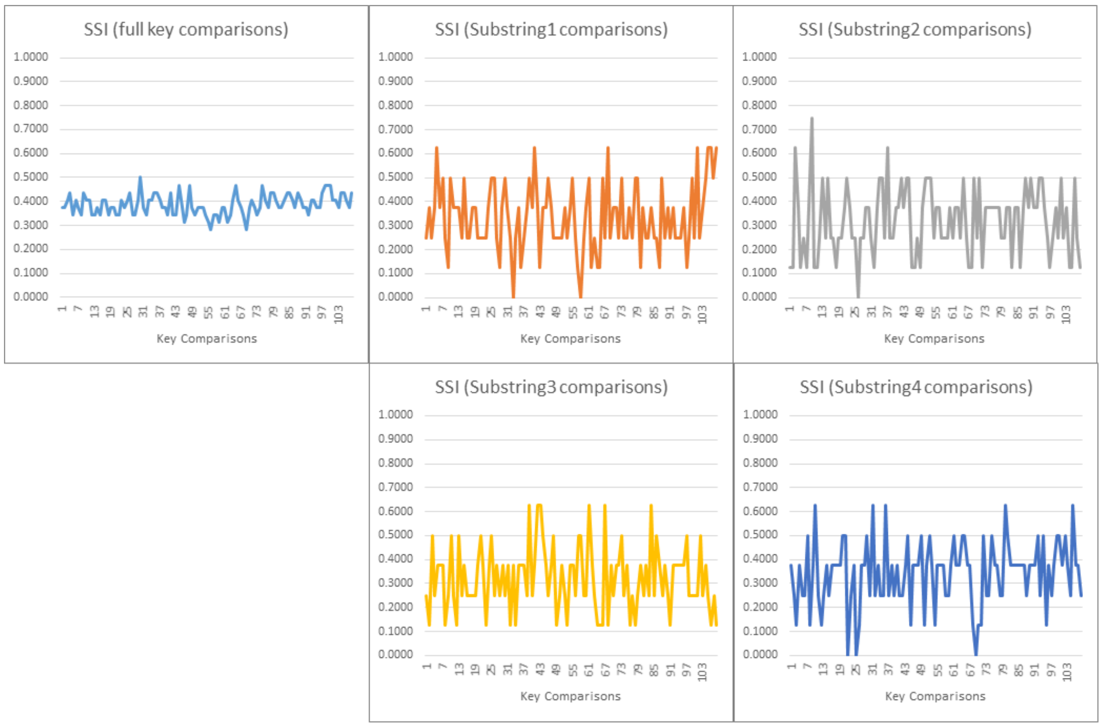


Figure 7.10: SSI Variation (Configuration $C2,\ key = K_i$)

that is decided by the server. The increased similarity in $substring4$ can be attributed to s_j changing by a lower factor than other keys. The low value of SSI for the full key comparisons for s_j, thus, is dependent on the lack of variation in similarity in $substring1$ and $substring2$. However, since s_j is only one of the parameters that are used to generate the encryption key, K_i, and transfer key, K_T, and with its updates controlled by the server, we can deduce that the lack of variability in similarity in this case does not compromise the security of the overall system.

This aspect is justified by the increased variability in the similarity in keys, K_i and K_T. The full keys and each substring in K_i and K_T display considerable variation, which can be attributed to the changing timestamp and s_j, respectively, that generate these keys. We have to note that although there is a lack of variation in s_j, PRNG algorithms are designed in a way so as to change the outputs significantly with small changes in the seed value; an aspect we capitalize on in our Butterfly1 proposal.

The keys generated by HiveSec1 are dependent on several factors, including the timestamp, random choice of one of six parent seeds (and its neighbours), and one of either 6 or 18 possible states of the child seeds, which are then used to generate the

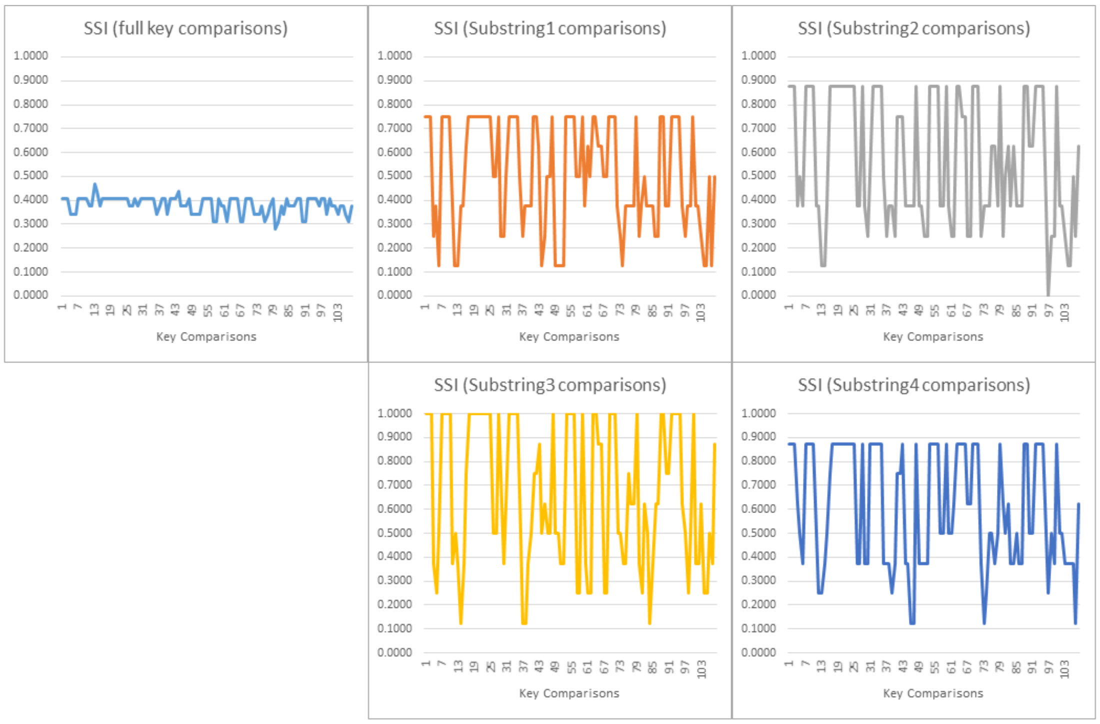


Figure 7.11: SSI Variation (Configuration $C2$, $key = K_T$)

final key. This ensures unpredictability in key generation. When considered with the fact that the sessions update either at random or as decided by the session parameter, n_δ, and the use of initial timestamp (t_{s0}), the unpredictability of the overall algorithm remains high. Our similarity assessment of this algorithm highlights the increased variability in similarity coefficients, with the full key SSI for K_S centred around 0.3700 (Figure 7.12) and that for K_O centred around 0.3800 (Figure 7.13).

The substring similarity remains similar to configuration C1 in K_S, with most substring SSIs in the range,$0.0000 \leq SSI_{substrings} \leq 0.6000$, with a few exceptions. However, when we observe the substrings for K_O, we see some values being exactly similar, i.e. with $SSI = 1.0000$. This is because in a single session, the outer envelope key (K_O) is not expected to change. Nevertheless, with the sessions changing as mandated by the server (through s_e), the variability in the full keys overcomes any lack thereof in the substrings, justifying our claim that the security remains high due to varying similarity between consecutive keys.

The similarity of keys in the framework configurations $C4$, $C5$, $C6$ and $C7$ depend on the following factors — the number of algorithms used in the framework (3 in $C4$

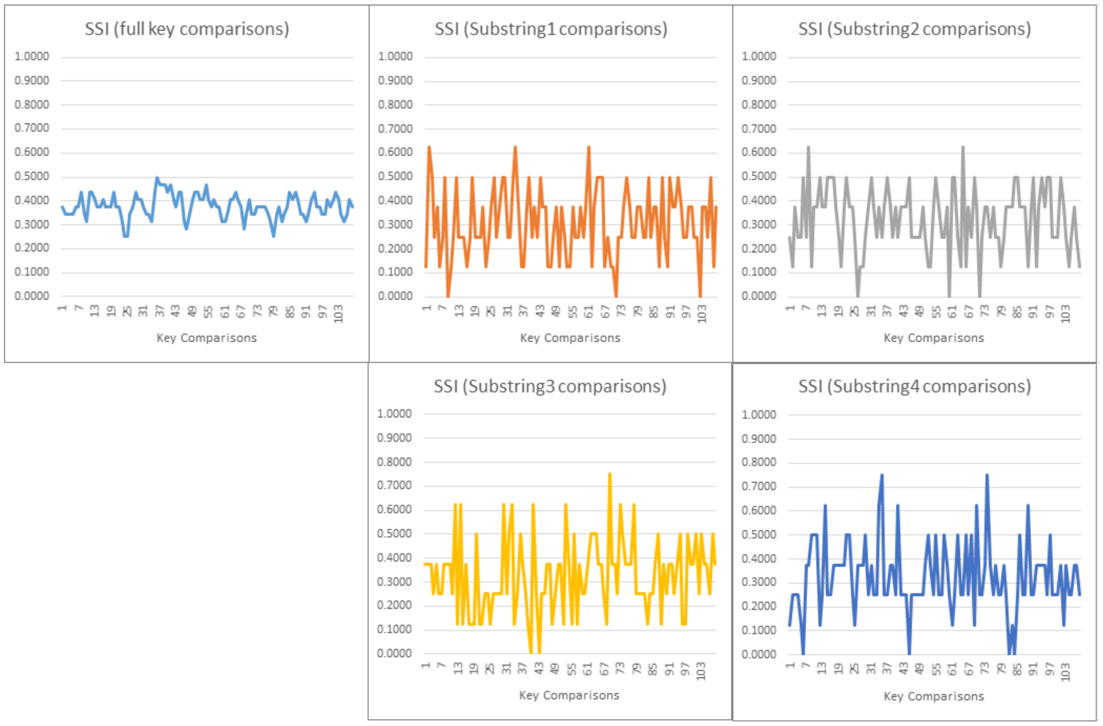


Figure 7.12: SSI Variation (Configuration $C3$, $key = K_S$)

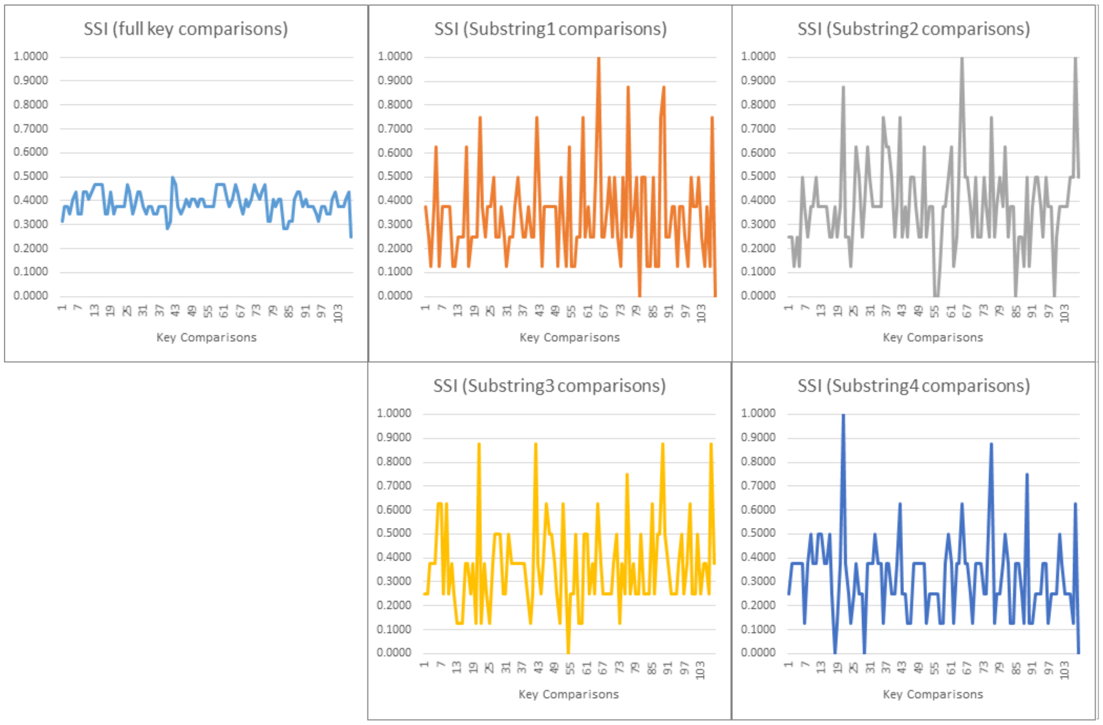


Figure 7.13: SSI Variation (Configuration $C3$, $key = K_O$)

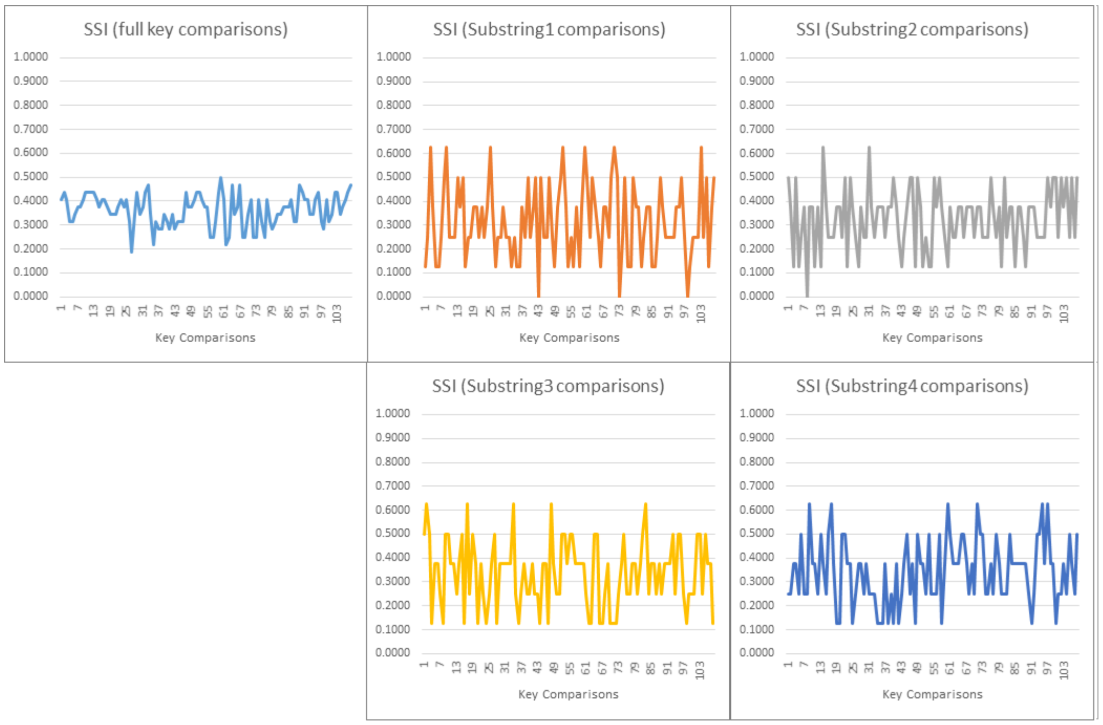


Figure 7.14: SSI Variation (Configuration $C4$, $keys = \{K, K_i, K_S\}$)

and 2 in $C5, C6, C7$), and on the random choice of one of the available algorithms. The keys in these configurations are therefore, implicitly not expected to be similar to each other. This is supported by the varying similarities in the keys generated by the individual algorithms discussed previously, and by the plots of the SSI values illustrated in Figures 7.14 — 7.21. The main key combination configurations in the framework, i.e. $C4(K, K_i, K_S)$, $C5(K, K_i)$, $C6(K, K_S)$ and $C7(K_i, K_S)$ all display a high amount of variability in SSI coefficients, with an overall trend of the SSI values being centred around 0.3700. The additional key configurations, i.e. $C4(K, K_T, K_O)$, $C5(K, K_T)$, $C6(K, K_O)$ and $C7(K_T, K_O)$, display varied similarities, although they are generally centred around 0.3850.

This exercise of determining the similarity between the keys leads us to believe that all configurations have SSI values closer to the average (expected) case discussed in Section 7.2.4. This goes to justify our claim that the "quantified" similarity between keys is lower, in most cases less than 0.4000, adding to the unpredictability of keys (if keys are not similar, it is less likely that an adversary might 'guess' the patterns in the subsequent keys) and thus, adding to the security of our proposal.

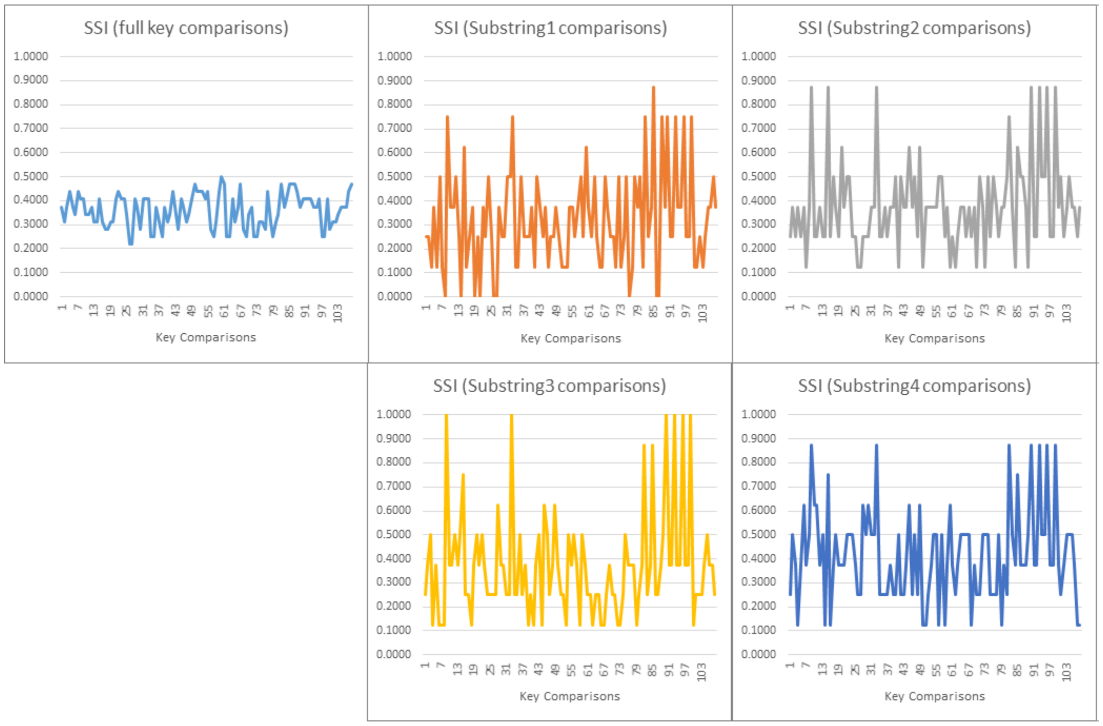


Figure 7.15: SSI Variation (Configuration $C4$, $keys = \{K, K_T, K_O\}$)

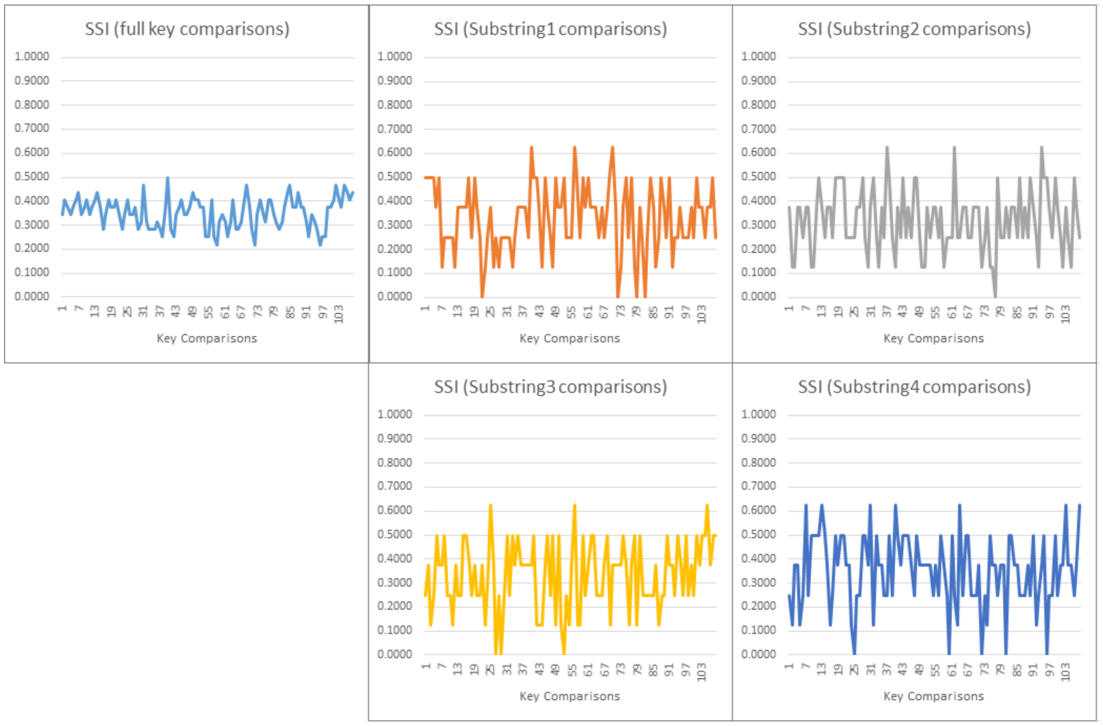


Figure 7.16: SSI Variation (Configuration $C5$, $keys = \{K, K_i\}$)

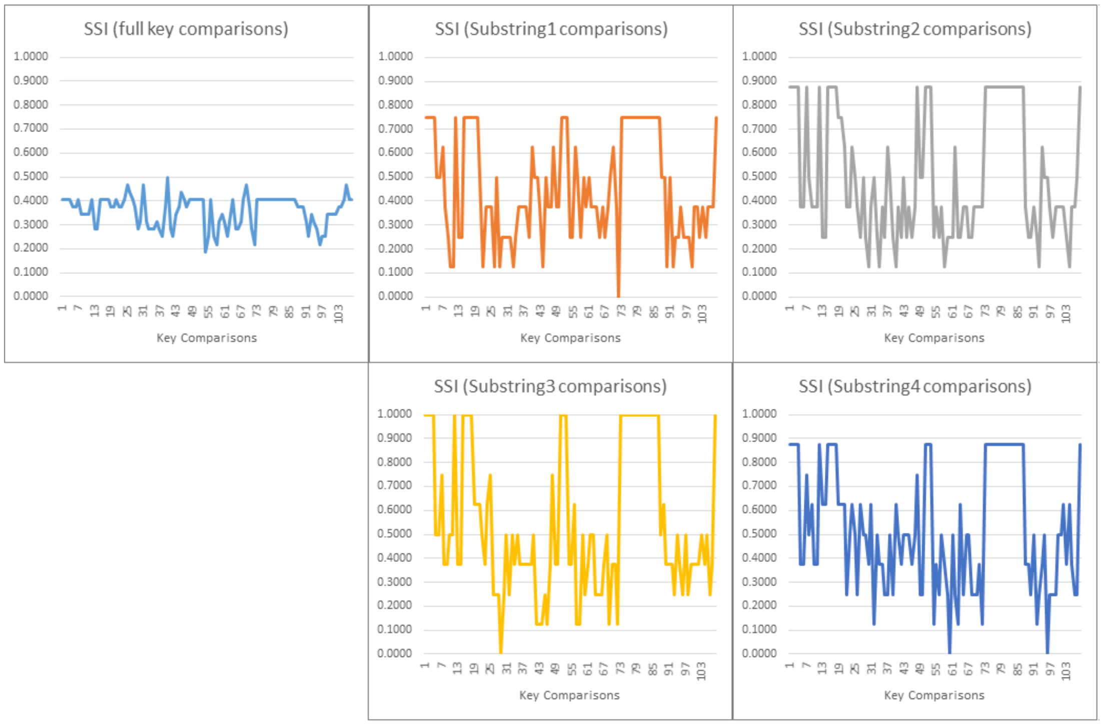


Figure 7.17: SSI Variation (Configuration $C5$, $keys = \{K, K_T\}$)

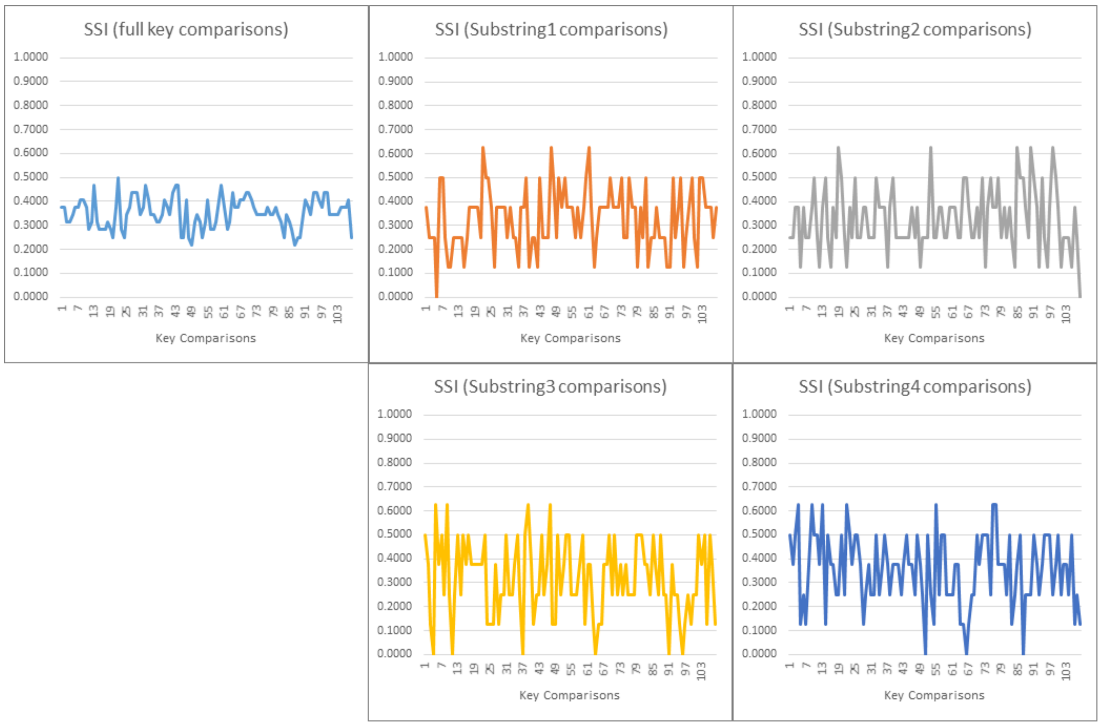


Figure 7.18: SSI Variation (Configuration $C6$, $keys = \{K, K_S\}$)

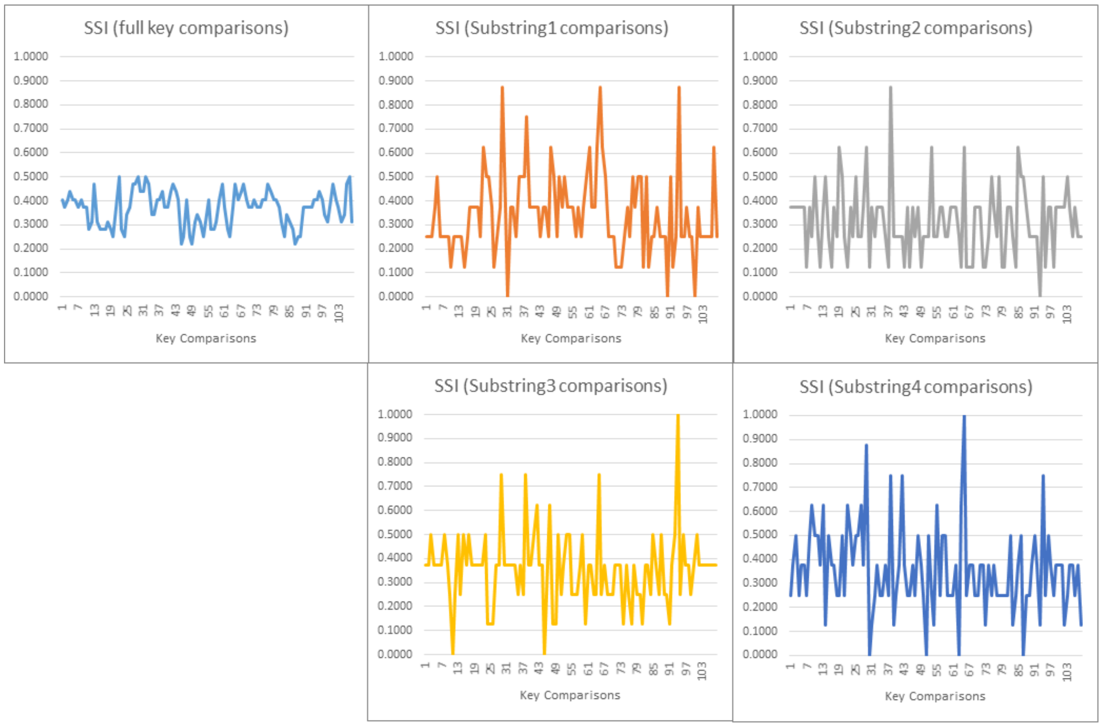


Figure 7.19: SSI Variation (Configuration $C6$, $keys = \{K, K_O\}$)

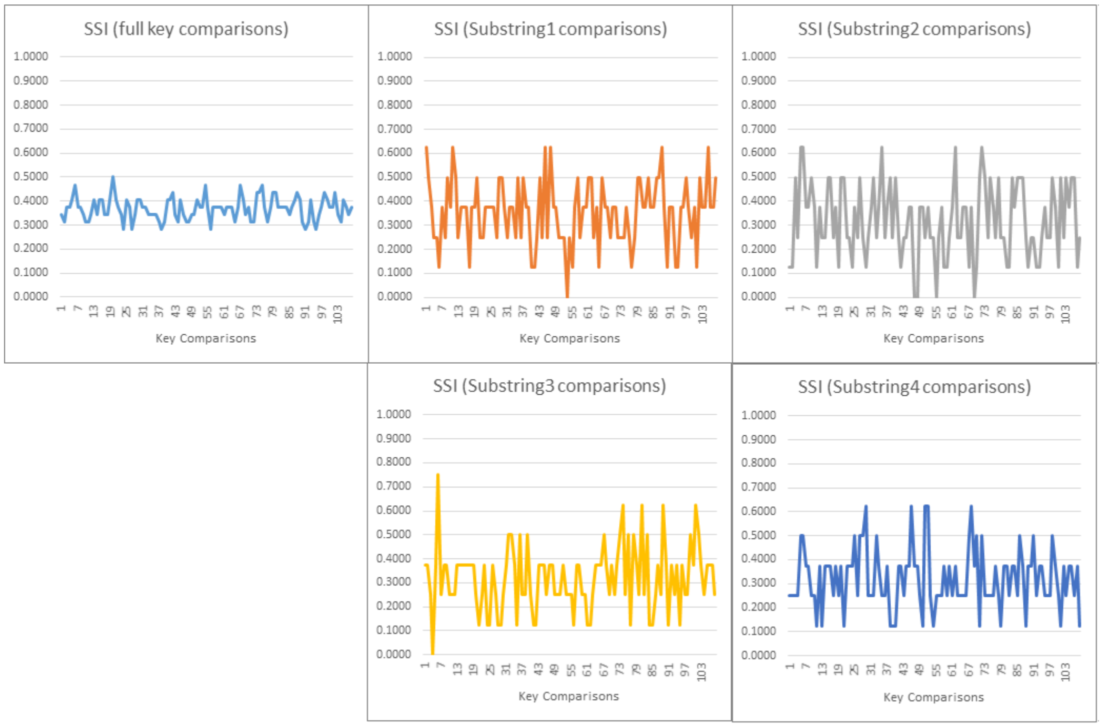


Figure 7.20: SSI Variation (Configuration $C7$, $keys = \{K_i, K_S\}$)

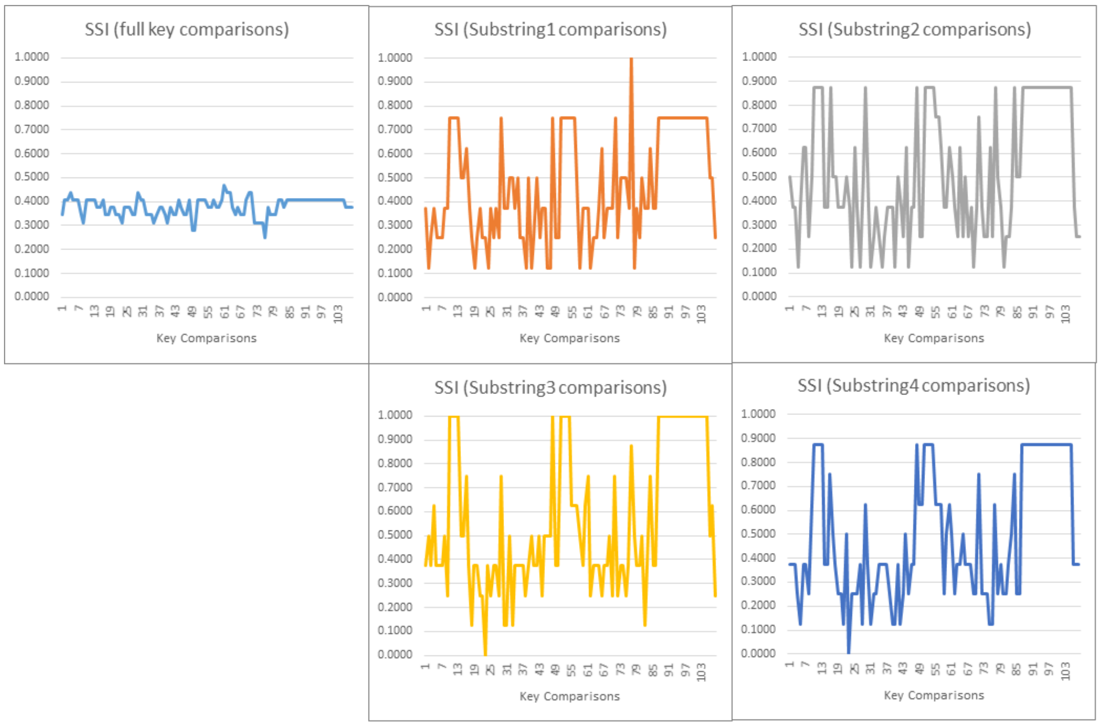


Figure 7.21: SSI Variation (Configuration $C7$, $keys = \{K_T, K_O\}$)

7.2.5 Evaluation of Key Randomness and Unpredictability

Methodology

Further to our assessment of similarity between consecutive keys (Section 7.2.4), we wanted to study the overall randomness associated with the key sequences generated by our proposed framework and its constituent algorithms. To evaluate the randomness (and unpredictability) properties, we used the Statistical Test Suite (STS) for Random and Pseudorandom Number Generators for Cryptographic Applications, specified by the National Institute of Standards and Technology (NIST) [130]. Although STS has been proposed to assess the randomness and unpredictability of random and pseudorandom number generators, and our proposals are *not* intended to be used as pseudorandom number generators, we use this assessment purely as a means to assess the randomness properties associated with the key sequences generated by our proposals.

For the purposes of tests using STS, we computed the binary values of the hexadecimal keys generated by our proof of concept implementation, using the same set up detailed in Table 7.1 and for each configuration listed in Table 7.2.

STS assessment assumes that each sequence of keys, in our case 128-bit keys, is a sequence of 1s and 0s. The test suite assesses randomness, i.e. the property of each bit being chosen at random using a variety of statistical tests. It is expected that for cryptographic applications, random and pseudorandom number generators must have forward and backward unpredictability. Forward unpredictability implies that even if an entity has a knowledge of a set of sequences generated by the PRNG under test, the entity must not be able to predict the sequences that follow. Backward unpredictability is the property of a PRNG to be safe from attacks intended at decoding the seed given a knowledge of a set of output pseudorandom sequences.

When testing any PRNG algorithm, the null hypothesis that is tested by STS is that the given sequence (and hence, the algorithm) is random, i.e. a "tentative assumption of randomness" [130]. If the tests result in overwhelming support of the algorithm being tested, the null hypothesis is not rejected. The randomness test statistic and the corresponding critical value are determined by the STS, to assist in the decision regarding the null hypothesis. For each test, the bit sequence is used to compute the test statistic and $\alpha = 0.01$. Each randomness test in STS computes a P-value, which is a measure of the probability that the sequence being tested is more random than that generated by a perfect random number generator. This means that the sequence would "appear to have perfect randomness" for a value of $P = 1.0$, and "appear to be completely non-random" for a value of $P = 0.0$.

For the STS tests, the bit sequence (of length n) is represented by:

$$\varepsilon = \varepsilon_1, \varepsilon_2, \varepsilon_3, ..., \varepsilon_n \tag{7.3}$$

It must be noted that the STS has been designed to evaluate long sequences of bits, i.e. greater than 20000 bits, as would be expected from a (pseudo)random number generator. Even though STS is expected to work with sequences with length less than 20000 bits, we limit our assessment to evaluate the number of sequences that pass each test, as an attempt to determine the unpredictability associated with that configuration. For 10000 sequences (sample size, m), STS determines the range of acceptable proportions, i.e. the confidence interval, as follows [130]:

$$Confidence\ interval, CI = \hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}} \tag{7.4}$$

Here, $\hat{p}$ is $(1-\alpha)$, with α being the significance level. In this assessment, $\alpha = 0.01$ and sample size, $m = 10000$. The confidence interval is $CI = 0.99 \pm 3\sqrt{\frac{0.99(1-0.99)}{10000}} = 0.99 \pm 0.00298496$. Therefore, for a set of sequences (and hence, configuration) to be considered random, the minimum (acceptable) number of sequences that have to pass the test are $(0.99 - 0.00298496) \times 10000 = 0.98701504 \times 10000 \approx 9870$ sequences.

We use the following statistical tests, to assess the (10000×128-bit) sequences generated using the various configurations listed in Table 7.2.

- Frequency (Monobit) Test:
  This test is used to compute the proportion of 1s and 0s for the entire sequence. Specifically, the test assesses whether the sequence being tested has an equal number of 1s and 0s. The absolute value of the sum of the bit distribution is first computed, i.e. $S_n = \sum X_i$ with $X_i = +1$ if a bit is 1 and $X_i = -1$ if a bit is 0. The test statistic is the ratio of the absolute value to the length of the bit string (n), i.e. $s_{obs} = \frac{S_n}{n}$. If the number of 1s and 0s in each sequence are

approximately equal in number, the value of the test statistic is nearer to 0.0, resulting in high P-values, implying that the sequence appears to be random.

- Frequency Test within a Block:
  This test is similar to the Frequency (Monobit) test. It is used to determine the number of 1s in a set of M-bit blocks in the sequence. The proportion of 1s, π_i, is computed as $\pi_i = \frac{\sum_{j=1}^{M} \varepsilon_{(i-1)M+j}}{M}$, for $1 \leq i \leq N$ and $N = \lfloor \frac{n}{M} \rfloor =$ number of non-overlapping blocks. Using π_i, the test statistic is determined as $\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - \frac{1}{2})^2$. As with the Frequency (Monobit) test, if the number of 1s in each block is approximately equal in half of the length of the block, the value of the test statistic is nearer to 0.0, resulting in high P-values, implying that the block (and hence, the sequence) appears to be random.

  For our assessment, we set the block size to be 32 bits, to be consistent with the tests for similarity for the configurations.

- Runs Test:
  A contiguous set of identical bits is referred to as a *run*, and the Runs test is used to determine the number of such runs that exist in the sequence under test. This test is used to determine if such runs are spread out in the sequence, i.e. "whether the oscillation between 0s and 1s is too fast or too slow" [130]. First, the proportion of 1s, π is computed as $\pi = \frac{\sum \varepsilon_j}{n}$. Then, it proceeds to determine if the sequence has passed the Frequency (Monobit) test. Runs test is not performed if $|\pi - \frac{1}{2}| \geq \tau$, with $\tau = \frac{2}{\sqrt{n}}$. Following this, if the sequence passes the Frequency (Monobit) test, the test statistic is computed as $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$, with $r(k) = 0$ when $\varepsilon_k = \varepsilon_{k+1}$ and $r(k) = 1$ otherwise. The P-value is computed using the test statistic and when $P < 0.01$, the sequence is considered to be non-random.

- Longest Run of Ones in a Block:
  This test checks for the longest run of 1s in blocks of size, M bits, as a means to verify if the length of such a run is the same as what would be generated by a random sequence. For our assessment, the block size, M was set to 8 bits (predefined by STS, for a sequence length of 128 bits). The test statistic is obtained as $\chi^2(obs) = \sum_{i=0}^{K} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}$. For a block size of $M = 8$ bits, STS

defines $K = 3$, $N = 16$ and π_i are theoretical probabilities of $K + 1$ classes. ν_i represents the frequencies of the longest runs of 1s in the categories ≤ 1, 2, 3 or ≥ 4, for $M = 8$ bits. Higher P-values (nearer to 1.0) indicate that the sequence is considered random.

- Discrete Fourier Transform (Spectral Test):
  A random sequence requires that periodic features (such as repetitive patterns) not be present. This test is used to determine whether any such features exist, by computing the "normalized difference (d) between the observed and expected number of frequency components that are beyond the 95% threshold". The value of d is determined as $d = \frac{(N_1 - N_0)}{\sqrt{n(0.95)(0.05)/4}}$, with N_0 being the expected (95%) number of peaks when randomness is assumed that are less than the threshold, $T = \sqrt{(\log(\frac{1}{0.05}))n}$, and N_1 being the actual number of peaks that are less than T. A low P-value indicates the presence of periodic features, and hence means that the sequence under test must be considered non-random.

STS determines the uniform distribution of P-values as a summary $P - value_T$ computed using the complementary incomplete gamma function, *igamc*, using the following expression:

$$P - value_T = igamc(\frac{9}{2}, \frac{\chi^2}{2}) \tag{7.5}$$

Here, $\chi^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10}$, with F_i being the total number of P-values in the interval, i, and i is the one of ten equal intervals between 0.0 and 1.0. If the resultant $P - value_T \geq 0.0001$, then, the P-values are considered to be uniformly distributed, a characteristic desired in (pseudo)random number generators. However, to reiterate, since our proposals are not intended to function as (pseudo)random number generators, we explore the acceptance with regard to the number of proportions that pass each test. In the next section, we discuss the results obtained when we tested the sequences generated by our algorithms using STS.

Results

We limited the analysis to the total number of sequences (out of 10000) that passed each statistical test discussed in the previous section. To be consistent in our analysis, we considered the following configurations (and keys) for assessment, and considered a block size of 32 bits for the frequency test within a block.

- K generated by GeM2 (configuration $C1$);
- K_i and K_T generated by the Butterfly1 algorithm (configuration $C2$);
- K_S and K_O generated by the HiveSec1 algorithm (configuration $C3$);
- Key combinations (K, K_i, K_S) and (K, K_T, K_O) generated by the framework with all three algorithms (configuration $C4$);
- Key combinations (K, K_i) and (K, K_T) generated by the framework with GeM2 and Butterfly1 algorithms (configuration $C5$);
- Key combinations (K, K_S) and (K, K_O) generated by the framework with GeM2 and HiveSec1 algorithms (configuration $C6$);
- Key combinations (K_i, K_S) and (K_T, K_O) generated by the framework with Butterfly1 and HiveSec1 algorithms (configuration $C7$);

Table 7.6 summarizes the number of sequences from each configuration that passed each statistical test, noting that according to Equation (7.4), the total number of sequences that need to pass a test to be considered as unpredictable or random are 9870 out of 10000 (or, 98.70%).

We observe from Table 7.6 that although many configurations pass the 98.70% criterion, some significantly, several configurations fail the test. We discuss how the configurations performed in each test in the paragraphs that follow.

We observe that configurations C2, C3 and C7, with their different key combinations outperform the others in the Frequency (Monobit) test. This goes to show that the Butterfly1 and HiveSec1 key generation, and their combinations, proposals perform better. We can also observe that GeM2 (configuration C1) performs poorly. This was expected in the STS analysis because the concept of gene mutation and transfer, as applied in this algorithm, makes the keys evolve keeping one key as the "parent". This implies that such a restricted evolution of keys limits the randomness in the keys, although the keys output by GeM2 are low in terms of similarity between consecutive keys and in terms of the unique keys generated. This behavior is carried

Table 7.6: NIST STS Assessment: Summary of results (Number of sequences out of 10000 that passed each test)

Configuration (Keys considered)	FM Test[1]	FB Test[2]	Runs Test	LR Test[3]	DFT Test[4]
C1 (K)	4998	5027	5140	5360	9933
C2 (K_i)	9895	9908	9906	9896	9862
C2 (K_T)	9906	9906	10000	9917	10000
C3 (K_S)	9908	9921	9907	9918	9857
C3 (K_O)	9885	9930	9909	9904	9804
C4 (K, K_i, K_S)	8288	8320	8342	8400	9856
C4 (K, K_T, K_O)	8299	8318	8365	8416	9917
C5 (K, K_i)	7555	7564	7623	7716	9885
C5 (K, K_T)	7564	7572	7666	7724	9964
C6 (K, K_S)	7405	7423	7482	7585	9876
C6 (K, K_O)	7406	7427	7467	7573	9889
C7 (K_i, K_S)	9885	9907	9880	9905	9837
C7 (K_T, K_O)	9901	9897	9951	9918	9909

1 : Frequency (Monobit) Test

2 : Frequency Test within a Block

3 : Longest Runs of Ones in a Block

4 : Discrete Fourier Transform (Spectral Test)

into configurations C4, C5 and C6, where GeM2 is a component. The improvement due to Butterfly1 and HiveSec1 can be clearly observed when we compare the number of sequences that passed the frequency test for configurations C4, C5, C6 and C7, i.e. the framework configurations. We observe that both key combinations of C7 pass the test, while the success rate of C5 and C6 are lower. When we observe C4, the performance in the test seems to be better compared to C5 and C6, due to the presence of both Butterfly1 and HiveSec1.

The performance of each configuration in all other tests, except the Discrete Fourier Transform test, is similar to the frequency test in that GeM2 performs poorly, Butterfly1 and HiveSec1 outperform it considerably, and contribute to the improvement in the performance of the framework even when GeM2 is included. In this context, we refer to performance as being indicative of the number of sequences that passed the test.

When we consider Discrete Fourier Transform test, however, we observe that almost all the configurations, including C1 (GeM2), pass this test. The configurations that come close to the passing criterion are C2 (K_i), C3 (K_S), C3 (K_O) and C7 (K_i, K_S). This indicates that the distributed periodic features come close to performing similar to a random sequence.

From our observations and from the data in Table 7.6, we can conclude that the configurations C2, C3 and C7 prove to be acceptable, when considering the proportions of test that pass the criterion (> 9870). However, detailed results (in Appendix A) indicate that the summary P-values, i.e. $P - value_T$, representing the uniform distribution of P-values is 0.0000 in all cases. The detailed result tables in Appendix A also indicate the distribution of P-values in each of the ten equal intervals, Ci, between 0.0 and 1.0. We observe that although a set of sequences pass each test based on the proportion criterion (> 9870), the distribution of P-values is not uniform. Thus, although some of the proposals could be considered as having acceptable randomness under proportionality criterion, they cannot be considered the same under the P-value uniformity criterion.

Although certain configurations of the framework fail the STS assessment, we need to recall to the fact that STS assesses individual sequences (of 128 bits each), to determine their randomness/unpredictability as characterized by their performance with respect to each test. We also need to draw our attention to the fact that the overall performance of a system employing the framework depends on the unpredictability associated with the choice of the algorithm and the current time (as specified by the timestamp), which is established to be high in unpredictability as per the previous assessments.

7.3 Security Evaluation of the Proposals

We accomplish the evaluation of security of each individual algorithm and the framework as a whole with an evaluation using Scyther protocol analyser, and using qualitative security analysis of the proposals based on the security goals they satisfy and the attacks they prevent.

7.3.1 Security Analysis

We evaluate our proposals with respect to key security goals [1] (summarized in Table 2.1 found on page 13 of this thesis). These security goals are essential to communication systems and are expected to be satisfied by cryptosystems. In our assessment, we first discuss how each individual algorithm and the framework configurations accomplish these security goals in our discussion of results obtained using Scyther protocol analyser, and summarize the results in the section that follows.

We also assess the performance of each algorithm and the framework configurations against known attacks that are relevant in resource-constrained systems, classified under various network attacks by Mitrokotsa et al. [4] and Chaudhry et al. [5] (summarized in Table 7.7). Note that these attacks are a subset of attacks discussed in Table 2.2 [2, 3] (found on page 28 in this thesis), it is our belief that an algorithm that can prevent or foil these attacks can also successfully prevent the other attacks listed in Table 2.2.

7.3.2 Communication Protocol Analysis using Scyther

A secure algorithm or a framework also depends on its capability to not be prone to, or be capable of defending itself from, attacks during the communication. A way to assess this is to evaluate the security of the communication protocol. To accomplish this, we used a protocol analyser named Scyther [131, 132, 133, 134]. Scyther helps verify the security of a message being transmitted from the sender to the receiver and identifies whether there are any attacks that can be performed. Scyther is basically a "blackbox" protocol testing suite, which assumes that the adversary has full access to the channel, i.e. it employs the Dolev and Yao adversary model [131, 35]. The Dolev and Yao model assumes that the cryptography is perfect, messages are abstract terms

Table 7.7: Known network attacks [4] [5]

Attack	Description
Eavesdropping	Unwarranted ‘listening’ to a communication in progress
Replay attack	Attempts to replay previous messages, to gain access
Man-in-the-middle	An unauthorized entity acting as an intermediary, in the communication, who may or may not modify messages
Tracking	Unauthorized entity tracks responses by the entity to ‘profile’ activity of the associated user
Denial of Service (DoS)	Flooding the entity with information requests or overwhelming the system resources by other accesses
De-synchronization	Attempts to upset the synchronization of states in the system entities
Dropped frames	A type of Man-in-the-middle attack; Unauthorized entity selectively drops messages

and assumes full control for the adversary over the channel. The adversary entity in Scyther thus, attempts to perform various attacks in an attempt to determine how secure the protocol would be under such attack scenarios. We summarize the configuration used in Scyther assessment in Table 7.8. Note that the number of runs is set to “1” to indicate one communication session between the with the resource-constrained entity sending its data and the server acknowledging its receipt, followed by a seed update session.

We consider that it is sufficient to verify the security of the constituent algorithms, as a way to assess the security of the framework. This is because the framework is composed of the proposed algorithms, and the security of each constituent algorithm contributes to the security of the overall framework. This is also due to the fact that Scyther does not facilitate using random choices of algorithms within a framework, such as the one proposed in this thesis. Scyther assesses the security of each message that is communicated over the channel and checks the vulnerabilities of such messages, and since the working of our framework (and its security) is dependent on the random choices of one of the available algorithms in the framework, we believe that it will be sufficient to test the security of the constituent algorithms. Furthermore, since the framework randomly chooses one of its constituent algorithms, it further increases the unpredictability, and hence, security. However, we assess our framework configurations, $C4$, $C5$, $C6$ and $C7$, albeit without the random choices and with each algorithm chosen sequentially, i.e.

- in $C4$, GeM2 is used first, followed by Butterfly1 and HiveSec1;
- in $C5$, GeM2 is used first, followed by Butterfly1;
- in $C6$, GeM2 is used first, followed by HiveSec1;
- in $C7$, Butterfly1 is used first, followed by HiveSec1;

assuming that the random choice of algorithms in the framework will yield in the above choices.

To evaluate each algorithm (and the framework), we have a set of claims [132, 134], which are essentially the expected security behavior by each algorithm. These claims are — secrecy, aliveness, weak-agree, non-injective synchronization and non-injective agreement. We describe them below. Note that all claims, except secrecy, are common for all algorithms.

Claim-1: Secrecy:

This goal helps verify whether the parameters expected to be secret in the transmitted message, regardless of the algorithm employed, remains a secret as the message traverses the communication channel. Since we have fundamentally different algorithms, each algorithm ensures that some of its parameters remain a secret. However, Scyther imposes some restrictions in terms of the variable names that can be used. For example, θ_i is the message signature in Butterfly1. However, θ is a special symbol

Table 7.8: Parameter settings used for evaluation of the proposed algorithms using Scyther

Parameter	Value
Maximum number of runs	1
Matching type	Find all type flaws
Search pruning	Find all attacks
Maximum number of patterns per claim	10

Table 7.9: Algorithm parameters and their implementations in Scyther

GeM2		Butterfly1		HiveSec1	
Parameter	In Scyther as	Parameter	In Scyther as	Parameter	In Scyther as
$K1$	$K1$	s_j	sj	K_S	KS
$K2$	$K2$	K_i	ki	K_O	KO
asv	asv	K_T	kt	t_s	TS
$K2$	$K2$	m_i	mi	$message$	mi
asv	asv	t_i	ti	$Response$	$mResponse$
ID	ID	i	$i1$	$msign_I$	$msignI$
ack	ack	θ_i	$theta$	$msign_R$	$msignR$
		t_u	$tinew$		
		$theta_u$	$theta1$		

that Scyther does not support. Hence, we used the word *theta* as its replacement. We list the complete set of the parameters expected to be a secret by each algorithm and their corresponding Scyther implementation names in Table 7.9.

Claim-2: Alive:

Aliveness is a a justification of whether a communicating entity has been active during the communication, verified by the use of a common protocol. Specifically, communicating entities are said to be *alive* if the algorithm being evaluated has been employed to secure the previously communicated messages. This means that if Alice is the initiator of the communication for the i^{th} message in a sequence of messages with entity Bob, she will guarantee that Bob is *alive* if they had employed the same algorithm (and hence, the protocol) for the previous $(i-1)$ messages.

Claim-3: Weakagree:

This claim is used further support the claim of *aliveness*. Continuing our previous example, if Alice was the initiator in the communication of the i^{th} message using a particular algorithm (and hence, the protocol), Bob's response using the same protocol guarantees that the entities are in *weak agreement*, i.e. it suggests that Bob may have been using the same protocol for the previous messages.

Claim-4: Non-injective Synchronization (Nisynch):

When entities communicate, it is an expectation on the part of either entity that the other is honest and trusted. Thus, Alice would expect that the message reaches Bob without any errors or modifications and thus, enable Bob to synchronize his system state to that of Alice. Therefore, $Nisynch$ is a means to check the synchronization between the initiator and responder, specifically defined as a parameter that summarizes the use of the proposed algorithm by the entities and that they are synchronized during the said communication.

Claim-5: Non-injective Agreement (Niagree):
Synchronized entities further guarantee that the various variables used and the values associated with them are agreed upon. Therefore, $Niagree$ indicates that the entities agree upon few of the session parameters during a given session.

It is to be noted that the security claims discussed below and the secrecy claims summarized in Table 7.9 apply to the framework assessment as well. We discuss the results of the assessment of various configurations using Scyther in the following section.

7.3.3 Results: Protocol Analysis using Scyther

Configuration $C1$, GeM2

We summarize the results of evaluating GeM2 using Scyther protocol analyser in Table 7.10. Our analysis substantiates our claims that GeM2 offers good security. In the paragraphs that follow, we discuss the outcome for each of the claims and the corresponding security goal the claim will impact.

Claim-1: Secrecy: GeM2 uses encryption using continuously changing keys, which are randomly derived from previous keys. Use of synchronized PRNGs to generate the choices as well as the keys ensures that encryption keys are generated independently and maintained a secret, therefore assuring high unpredictability associated with the system states. This ensures secrecy and thus, helps us accomplish *confidentiality.*

Claims-2, 3 and 4: Alive, Weakagree and Non-injective Synchronization (Nisynch): Scyther validates our claims that the entities are running the same scheme (Weakagree) and all previous message sessions have used the proposed scheme (Alive). Validation of these claims further justifies that the communicating entities are synchronized in their continued use of GeM2. Their synchronization is facilitated by the parent and generation state identifiers, p and g, that are sent along with the encrypted message. This is a move to protect against *replay attacks* as well as *de-synchronization attacks*.

Claim-5: Non-injective Agreement (Niagree): GeM2 relies on the entities pre-sharing the initial key, IK, at deploy time. This is never shared openly or through encrypted messages. The generation of all future encryption keys using GeM2 depends on future states, derived through strategic linking beginning with IK, which makes it harder for an adversary to guess the exact current state of the system. This claim is verified by Scyther.

Authentication and Non-repudiation: GeM2 is proposed as a mutual authentication algorithm and we accomplish that goal through the use of the authentication synchronization vector, asv_i, as well as the parent and generation numbers. These parameters ensure that for a particular set of values of parent and generation, there is only one value of asv_i, thereby helping both entities verify the authenticity of the other. Furthermore, with the messages using encrypted IDs and pre-shared acknowledgement patterns, encrypted using keys generated in the currently synchronized state, GeM2 is able to naïvely accomplish non-repudiation as well (non-repudiation

Table 7.10: Results: Evaluation of the GeM2 protocol using Scyther

Claim	Initiator Status	Responder Status
Secret ID	✓N_{WB}^{*}	✓N_{WB}
Secret asv	✓N_{WB}	✓N_{WB}
Secret ack	✓N_{WB}	✓N_{WB}
Secret $K1$	✓$NAV^{\dagger}$	✓NAV
Secret $K2$	✓NAV	✓NAV
$Alive$	✓NAV	✓NAV
$Weakagree$	✓NAV	✓NAV
$Nisynch$	✓N_{WB}	✓N_{WB}
$Niagree$	✓N_{WB}	✓N_{WB}

* N_{WB} : No attacks, within bounds

† NAV : No attacks, verified

in this case, is by association of the resource-constrained entity with the server, which is assumed to be authentic/verified).

Forward and Backward security: To any unauthorized entity intercepting messages, GeM2 key generation process appears as a blackbox (pseudo)random number generator. The random choices for parent key refresh and for determining the number of child keys a parent key can have adds additional security to the system. Furthermore, the seed generator states are also updated continuously. In the event that an adversary were to gain knowledge of either a contiguous set of previously used keys or future keys, the adversary will still not have any knowledge of the internal PRNGs that are used to make said choices. This means that the system state remains unpredictable to an observer, thus maintaining security and preserving the privacy of the user.

Configuration $C2$, Butterfly1

Table 7.11 summarizes the results of evaluation of our communication protocol using Scyther. From our evaluation, we can see that all our claims were satisfied given the parameters of the experiment. In the paragraphs that follow, we discuss the outcome for each claim and the corresponding security goal the claim impacts.

Claim-1: Secrecy: With keys dynamically updated using timestamps, our protocol ensures that all secrecy claims are valid. This implies that the secrecy of the message, m_i and the updated seed, s', are both maintained a secret, thereby helping in achieve *confidentiality*. It has to be noted that even the message signatures, θ_i and θ_u are maintained a secret, since these are included in the second envelope encrypted with K_T, generated using seed s_j. This implies that the *integrity* of the message can be verified.

Claims-2 and 3: Alive and Weakagree: The protocol analyser validates our claims that the entities are running the same scheme (Weakagree) and all previous message sessions have used the proposed scheme (Alive).

Claim-4: Non-injective Synchronization (Nisynch): The analyser validates our claim that the protocol and the scheme ensure that the internal key generation states are synchronized in the communicating entities. This is possible due to the presence of timestamps, sequence numbers and message codes, which not only protect the system

Table 7.11: Results: Evaluation of the Butterfly1 communication protocol using Scyther

Claim	Initiator Status	Responder Status
Secret mi	✓$_{N^*_{WB}}$	✓$_{N_{WB}}$
Secret ti	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret i	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $i1$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $theta$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret sj	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret ki	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret kt	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $sjnew$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $tinew$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $theta1$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$Alive$	✓$_{NAV^\dagger}$	✓$_{N_{WB}}$
$Weakagree$	✓$_{NAV}$	✓$_{N_{WB}}$
$Nisynch$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$Niagree$	✓$_{N_{WB}}$	✓$_{N_{WB}}$

* N_{WB} : No attacks, within bounds

† NAV : No attacks, verified

against *replay attacks*, but also render *de-synchronization attacks* ineffective. This is supplemented by the multi-level enveloping technique employed by our scheme.

Claim-5: Non-injective Agreement (Niagree): The first seed, s_{init} is a central attribute in our scheme. This is never shared openly or even through encrypted messages. Its variant, s_j is used for generating multiple keys, and a new seed, s' is also communicated in an encrypted manner. This dependency on the initial seed protects the scheme operation since it makes it harder for an adversary to guess the particular state of a seed. This is an important pre-agreed parameter, and the analyser validates our claim that the entities can agree upon important parameters (such as s') during communication using this protocol.

Authentication and Non-repudiation: While using our scheme, the receiver can authenticate the communication initiating entity from the transmitted message, M_T, using η_t, pre-shared secret seed, s_j and the value of j to generate encryption keys to decrypt the received data. By computing θ_i, it is able to confirm the identity of the sender. Furthermore, the initiator can validate the responder on receipt of the acknowledgement message, when it decrypts ack_i to generate θ_i that it had previously sent. Thus, both entities can authenticate each other. Since the initial seed is pre-shared and never exchanged, only a legitimate entity will be able to generate specific values of θ_i, and the keys, K_i and K_T.

This implies that for each entity to be in continuous synchronization, neither of them can deny the messages they send. This, in a naïve manner, helps in accomplishing *non-repudiation*. Although symmetric cryptosystems cannot guarantee non-repudiation without use of asymmetric components or digital signatures [35], our approach facilitates non-repudiation by association. Since entities are communicating with a server with unlimited resources we assume the server to be authentic, and by verifying itself to the server, the resource-constrained entity guarantees that the messages sent by it are valid, if verifiable.

Forward and Backward secrecy: The use of timestamp to generate the encryption key, K_i, continuous changes to the seed, s_j, and hence the transfer key, K_T, helps in ensuring secrecy of data. Even if a contiguous set of keys were to be determined by an adversary, it would be challenging to determine either past keys or future keys, since the internal PRNG seed is not disclosed and the computed keys are dependent on the changing parameters. Thus, our Butterfly1 algorithm helps preserve forward and backward secrecy, adding to the overall security of the system.

Configuration $C3$, HiveSec1

We summarize the results of evaluating HiveSec1 using Scyther protocol analyser in Table 7.12. Our analysis substantiates our claims of offering security with the HiveSec1 concept. In the paragraphs that follow, we discuss the outcome for each of the claims presented in Section 7.3.2 and the corresponding security goal the claim will impact.

Claim-1: Secrecy: HiveSec1 uses two levels of encryption, and the keys used for both levels are dependent on two different timestamp values, i.e. initial timestamp,

Table 7.12: Results: Evaluation of the HiveSec1 communication protocol using Scyther

Claim	Initiator Status	Responder Status
Secret TS	✓$_{N^*_{WB}}$	✓$_{N_{WB}}$
Secret $msignI$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret mi	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret $mResponse$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
Secret KS	✓$_{NAV^\dagger}$	✓$_{NAV}$
Secret KO	✓$_{NAV}$	✓$_{NAV}$
$Alive$	✓$_{NAV}$	✓$_{NAV}$
$Weakagree$	✓$_{NAV}$	✓$_{NAV}$
$Nisynch$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$Niagree$	✓$_{N_{WB}}$	✓$_{N_{WB}}$

* N_{WB} : No attacks, within bounds

† NAV : No attacks, verified

t_{s0}, and the current timestamp, t_s. This, coupled with the random choices of parent/child seeds as proposed in HiveKey, ensures that the encryption keys keep changing for each message. This validates all our secrecy claims. This validation by Scyther implies that the secrecy of all messages and other parameters such as timestamp and message signature that are encrypted, remain a secret in the communication. This helps us achieve *confidentiality*.

Furthermore, the two message signatures, $msign_I$ and $msign_R$, contain different (randomly chosen) portions of the message as well as the key, thereby helping us accomplish *message (and encryption key) integrity* verification as well.

Claims-2 and 3: Alive and Weakagree: The protocol analyser validates our claims that the entities are running the same scheme (Weakagree) and all previous message sessions have used the proposed scheme (Alive).

Claim-4: Non-injective Synchronization (Nisynch): The analyser validates our claim that the protocol and the scheme ensure that the initiator and the responder states are synchronized. This is facilitated by the use of timestamps, session identifiers, session sequence numbers and message codes, in addition to a dynamic choice of

PRNG seeds using the HiveKey parent/child seed choice concept. This not only protects the system against *replay attacks*, but also render attempts at *de-synchronization* ineffective. This is further supplemented by the multi-level enveloping encryption employed by HiveSec1.

Claim-5: Non-injective Agreement (Niagree): The first set of parent seeds, $\{p_i\}$, in addition to the initial timestamp, t_{s0}, are agreed upon by the entities at deploy time. These are core elements in our algorithm. These are never shared openly or even, for that matter, through encrypted messages. The choice of a single seed (and hence, its neighbouring seeds) remains random and dependent on the current timestamp. Thus, using multiple pre-shared PRNG seeds with random choice of some of them makes it harder for an adversary to guess the state of the system, and hence, the keys used. Scyther validates our claim that the entities can agree upon important parameters in a communication employing HiveSec1.

Authentication and Non-repudiation: Entity authentication is facilitated by HiveSec1 through the message signatures, $msign_I$ and $msign_R$. Message signatures contain the current session, key, state and message parameters, in addition to the entity ID (ID_I and ID_R). These parameters, along with the timestamp, ensure that there is only one value of either message signature at a particular instant of time, for a particular entity. Thus, since both entities can validate the other, it helps HiveSec1 accomplish mutual authentication.

Furthermore, given that the parent seeds and initial timestamp are pre-shared (and updated when necessary, through specific special messages), only a legitimate initiator or responder will be able to generate specific values of encryption keys and message signatures at a given instance of time. Thus, neither entity will be able to deny any message that they had previously sent. This helps HiveSec1 accomplish non-repudiation as well (by association).

Forward and Backward security: HiveSec1 features multiple parent seeds, and maintains the number of child seeds and the choice of a child seed/its neighbours random, and as decided by the current timestamp. This helps us substantiate our claim that even with a knowledge of a continuous sequence of keys used previously, an adversary will not be able to generate future keys. The same holds true for a case when an adversary retrieves a sequence of keys used currently (or in the future) to derive keys used previously. Thus, HiveSec1 is able to achieve both forward and backward security, in addition to the other security goals.

Framework Configurations $C4$, $C5$, $C6$, $C7$

We summarize the results of evaluating the proposed framework (and various combination configurations) using Scyther protocol analyser in Table 7.13. Our analysis substantiates our claims of offering security with the framework. In the paragraphs that follow, we discuss the outcome for each of the claims presented in Section 7.3.2 and the corresponding security goal the claim will impact.

The framework configurations employ all, or combinations of, the algorithms proposed in our work, i.e. GeM2, Butterfly1 and HiveSec1. This means that the security of the framework is dependent on the security offered by the individual constituent algorithms, and at times, is improved by the framework. This is ascertained by the results summarized in Table 7.13. On comparing these results with the results of the individual algorithms summarized in Tables 7.10, 7.11 and 7.12, we observe that the results for the various framework configurations are consistent with the results of the individual algorithms. The framework configurations only provide an additional layer of unpredictability since one of the algorithms in these configurations are chosen at random based on several factors as discussed in Chapter 4. Next, we summarize our observations regarding each claim.

Table 7.13: Evaluation of the proposed framework using Scyther

Claim	$C4$		$C5$		$C6$		$C7$	
	In.[1]	Re.[2]	In.	Re.	In.	Re.	In.	Re.
Claim: *Secret*								
ID	✓ N^*_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	-	-
asv	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	-	-
ack	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	✓ N_{WB}	-	-

Continued on next page

Table 7.13 – continued from previous page

Claim	$C4$		$C5$		$C6$		$C7$	
	In.[1]	Re.[2]	In.	Re.	In.	Re.	In.	Re.
$K1$	✓$_{NAV^\dagger}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-
$K2$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-
mi	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
ti	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
i	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$i1$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$theta$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
sj	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$
ki	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$
kt	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$
$sjnew$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$
$tinew$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$theta1$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$
TS	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$msignI$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
mi	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
$mResponse$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	-	-	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$
KS	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$
KO	✓$_{NAV}$	✓$_{NAV}$	-	-	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$
Claim: $Alive$								
$Alive$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$
Claim: $Weakagree$								
$Weakagree$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$	✓$_{NAV}$
Claim: $Nisynch$								
$Nisynch$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$

Continued on next page

Table 7.13 – continued from previous page

Claim	$C4$		$C5$		$C6$		$C7$	
	In.[1]	Re.[2]	In.	Re.	In.	Re.	In.	Re.
Claim: $Niagree$								
$Niagree$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$	✓$_{N_{WB}}$

* N_{WB} : No attacks, within bounds

† NAV : No attacks, verified

- : Not Applicable for current configuration

1 : Initiator

2 : Responder

Claim-1: Secrecy: Parameters expected to be secret in each configuration remain a secret, including and primarily the key generation parameters and intended messages; a claim which holds true since none of these parameters are exchanged. As seen in the algorithm assessments, the parameters including timestamps, IDs, messages and the keys themselves are all ensured to be secret as shown in Table 7.13. This implies that the framework configurations accomplish *confidentiality.*

Claims-2 and 3: Alive and Weakagree: Scyther validates our claims that the entities are running the same configuration (Weakagree) and all previous message sessions have used the proposed scheme (Alive).

Claim-4: Non-injective Synchronization (Nisynch): Our claim that the initiator and responder states are synchronized in the framework configurations is also verified. The synchronization in these configurations is dependent on the internal states of the system, which requires that each algorithm states be synchronized. If any of the states are not synchronized, the entities can recognize the error, and act as specified by the deployment conditions. This guarantees protection against *replay attacks* and *de-synchronization* attacks.

Claim-5: Non-injective Agreement (Niagree): The initial deploy-time parameters, such as the initial seeds and initial timestamp, are never exchanged in the open. Each synchronized update of the system states imply that these parameters are automatically updated. Thus, the internal parameters that are essential in computing the key materials are always in agreement in both entities, as long as they are synchronized and authenticated. Scyther validates that the framework configurations are synchronized, and that they are in agreement.

Authentication and Non-repudiation: Each algorithm used in the framework facilitates authentication, using message signatures and the authentication synchronization vector. This ensures that the system states are synchronized (established by Scyther assessment) and that the entities are (mutually) authenticated. Furthermore, the use of pre-shared parameters and random choice of one of the available algorithms means that the sender of each message cannot deny that it was sent from that particular entity. Since keys are updated and potentially a different algorithm is chosen for each key generation/encryption, it means that the internal states of each entity (for each algorithm) are always updated to the latest (synchronized) version, as long as they are authenticated. This, in a naïve manner helps the framework accomplish non-repudiation (by association).

Forward and Backward security: To an observer without the knowledge of the internal states, the framework as a whole appears as a 'black-box' sequence generation engine. This means that it 'appears' to be a sequence generator that generates various keys and other parameters required to encrypt and sign messages. Unless the observing entity has a knowledge of the internal states and the algorithms chosen, knowledge of a contiguous set of keys either from the past or in the future would not yield useful information about the future keys or previously used keys, respectively. This is primarily due to the dependency on timestamp in the choice of algorithms as well as updates to internal states of the framework. This re-configurable or metamorphic property of our framework not only provides high security, but guarantees forward and backward secrecy as well.

7.3.4 Results: Security Analysis

From our discussions in the previous section, it is evident that the algorithms and the framework configurations pass the various security requirements, as specified by the security goals highlighted in Table 2.1. In Table 7.14, we summarize the performance

Table 7.14: Summary: Security goals satisfied by the algorithms and framework configurations

Security Goal	Status in configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
Confidentiality	✓	✓	✓	✓	✓	✓	✓
Integrity	✓	✓	✓	✓	✓	✓	✓
Authentication	✓	✓	✓	✓	✓	✓	✓
Non-repudiation	✓^{A}	✓^{A}	✓^{A}	✓^{A}	✓^{A}	✓^{A}	✓^{A}
Forward security	✓	✓	✓	✓	✓	✓	✓
Backward security	✓	✓	✓	✓	✓	✓	✓

A: Our proposals help accomplish non-repudiation in a naïve way, by association with and verification by a server

of each algorithm with respect to these essential security goals.

One aspect to note is that our proposals accomplish non-repudiation in a naïve way, meaning that although they don't employ the use of trusted third parties, the entities are implicitly capable of signing their messages. This is made possible by the presence of a resourceful entity, i.e. the server, in resource-constrained networks. With the server being one of the communicating entities and with the initial parameters being deployed on the devices without being exchanged, a verifiable parameter generated by either entity at any point in the communication is trustworthy, as the internal states are constantly updated and synchronized. This can be imagined to be similar to a scenario where the server delegates the signing authorization to the resource-constrained entity. Each successful verification of the resource-constrained entity by the server 'extends' this signing authorization (i.e. authorization by association). Furthermore, since our proposals ensure mutual authentication, the resource-constrained entity can verify the authenticity of the server itself, thereby acknowledging this signing authorization (and its subsequent renewals). This lets us claim that the proposals accomplish non-repudiation, i.e. neither entity can deny that a verified message originated at that system.

Several attacks applicable to resource-constrained wireless networks were presented in Table 7.7. Further to our discussions in Section 7.3.3 detailing results obtained using Scyther, we summarize the performance of our algorithms and framework configurations with respect to each attack.

As discussed in the Scyther results, our proposals are able to reduce the effects of or prevent replay attacks, de-synchronization and dropped frame attacks. Systems employing our algorithms and framework are also safe from tracking, an attack that has the potential to compromise the privacy of individuals and organizations connected to a particular resource-constrained entity. This is due to the continuous updates in system states. This ensures that the system outputs are constantly changing, despite in many cases the messages/replies remaining the same, implying that unauthorized entities will not be capable of monitoring a particular resource-constrained entity, or the individual/organization it is affiliated with.

Eavesdropping allows an unauthorized entity to monitor messages over a channel, with or without the intention of gaining access to the communication, i.e. with or without it 'graduating' into a man-in-the-middle (MITM) attack. By being an MITM, unauthorized entities will be able to monitor and modify any messages over the channel. This requires the entity to be able to gain access to the system communication, contingent on the eavesdropping attacks being successful and yielding meaningful information about the system states. Our algorithm proposals ensure that the internal key states are constantly changing, and without exchanging keys, keeps adversaries from being able to decode messages over the channel. An additional layer of complexity, and hence security, is introduced by the framework configurations, which choose one of the available algorithms at random. This ensures that the overall unpredictability of the system remains high, making the communications safe from yielding meaningful information when eavesdropped. This further implies that the ability of an adversary to gain entry into the system communication is significantly reduced, if at all, making the system safe from MITM attacks as well.

By being able to detect and prevent attacks presented in Table 7.7, we can claim that our proposals help secure systems against most attacks detailed in Table 2.2. This is because our proposals ensure different internal states in all entities involved, which means that for an adversary to impersonate an authorized node (as with Sinkhole attack), an adversary must have knowledge of the internal states of all associated devices. Any attempts to hijack a transmitted packet to a different route (as with

Table 7.15: Performance of the proposals with respect to known network attacks

Attack	Protection status in configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
Eavesdropping	✓	✓	✓	✓	✓	✓	✓
Replay attack	✓	✓	✓	✓	✓	✓	✓
Man-in-the-middle	$\checkmark^A$	$\checkmark^A$	$\checkmark^A$	$\checkmark^A$	$\checkmark^A$	$\checkmark^A$	$\checkmark^A$
Tracking	✓	✓	✓	✓	✓	✓	✓
Denial of Service	$\checkmark^B$	$\checkmark^B$	$\checkmark^B$	$\checkmark^B$	$\checkmark^B$	$\checkmark^B$	$\checkmark^B$
De-synchronization	$*$	✓	✓	✓	✓	✓	✓
Dropped frames	$*$	$\checkmark^C$	$\checkmark^C$	$\checkmark^C$	$\checkmark^C$	$\checkmark^C$	$\checkmark^C$

A: Continuous state updates ensure man-in-the-middle attack remains unfruitful

B: Programmed to stop responding in case of many requests, assuming these could be DoS attempts by flooding

C: Recognize attempts of dropped frames, since internal states are expected to update with each key generation; Also a mechanism to detect de-synchronization attempts

$*$: Can recover from such attempts for up to three previously stored key states, and can detect any further attempts for de-synchronization/dropped frames

Wormhole attack) would result in the targeted device not responding to the message also since the internal key states are different. Thus, even if one of the devices in the system is physically compromised by tampering or jamming, the functioning of the other devices will not be affected, thereby keeping the system secure. Furthermore, the design of our algorithms facilitate an entity 'demanding' that the other entity generate a particular key (using $\{p, g\}$ in GeM2, j in Butterfly1 and by using timestamp, t_s, in HiveSec1), which could help in the entities being able to generate groupwise keys for broadcast messages, although not explored in this thesis because our work is primarily on pairwise session keys between entities.

7.3.5 Behavior Under Attack

An attribute that is implicit in all our algorithm proposals is attack detection. This is facilitated by changing system states that are synchronized on entity authentication. However, this will not prevent adversaries from launching attacks. Although our proposals are aimed at improving system security through obscurity and constant updates, they will still be prone to some attacks such as denial of service. In such cases, our proposal can detect attacks using a threshold; however, they lack the ability to prevent them. The threshold is based on acceptable repeats of messages or acceptable invalid messages, i.e. messages that are not recognizable by the verified entities in the system. This threshold is set a value of 3 in the current implementations, i.e. if more than three repeated messages are received by either entity or if more than three invalid messages are recognized by either entity, the particular entity ceases communication. This mechanism facilitates detection of a potential attack and its action, i.e. ceasing communications, indicates the other entity that the other entity could be under attack. This measure is particularly useful for resource-constrained entities, which could be deployed in environments with varying security guarantees. Our algorithm and framework proposals detect attacks based on these conditions 100% of the time, guaranteeing attack detection under various scenarios.

The threshold value of 3 was set since GeM2 includes a scheme of storing the current key and two previous keys for synchronization purposes. This mechanism ensures that attempts to drop acknowledgement frames are tolerated, up to three times with the system being able to recover from such attacks. This threshold can be changed as required for the intended application.

However, this work on attack detection also led us to wonder if anything could be done in addition beyond merely detecting (and/or recovering from) attacks. We found our answer in the Bee Swarm approach described in Section 5.5.4 (on Page 90). Applicable to the HiveSec1 algorithm and the framework configurations employing HiveSec1, Bee Swarm is a technique by which the communicating entity *attacks the attacker*, i.e. generates and responds to detection of potential attacks with some meaningless random values with the same structure of an original message. The entity responds for a predetermined amount of time, or transmits a predetermined number of such meaningless messages, to the potentially invalid sender. This ensures a minimal flooding of responses to the potential invalid sender, as an attempt to misguide it regarding the system state. In our assessment, we had set the same threshold of attack detection for Bee Swarms as well, which meant that if the entity received 3 unexpected/invalid messages, it enables Bee Swarm module to respond with meaningless messages. One might wonder what would happen in a system employing the Bee Swarm module following a successful de-synchronization attack. In such cases, either communicating entity will recognize that something has gone awry, and will either attempt to recover from the issue or attempt to send an error message to the server (if the detecting entity is not the server). Either way, the entities can still detect that a potential attack is in progress and initiate corrective action.

Although Bee Swarm module is currently implemented only as a component of the HiveSec1 algorithm (and, thus, as part of the corresponding framework configurations), it is an independent module, which can be included as part of (and customized to work with) all the other algorithms and framework configurations as well.

7.4 Comparative Assessment

7.4.1 Overview

The algorithms and framework configurations proposed in this thesis are intended for application in resource-constrained wireless networks such as RFID tags and WBAN applications. Specifically, we consider their application in improving key management and authentication in RFID applications that could implement cryptographic suites [32, 33] facilitating the application of the various framework configurations or in other lightweight (Class-0/Class-1) applications that require minimalistic security, and in WBAN applications that support the use of symmetric cryptography (especially in master key pre-shared association) [66].

As discussed in Chapter 2, there have been several proposals for key management and authentication in these two domains. In this chapter, we consider three such proposals, namely, (a) block-wise key update and symmetric encryption approach for RFID proposed by Zhu and Khan [8], (b) hash (SHA-3)-based key generation and authentication approach for RFID proposed by Dong et al. [9], and (c) AES-based key generation approach for WBANs proposed by Liu and Kwak [7]. We chose these three approaches in particular for the following reasons — the proposal by Zhu and Khan implements a linked key update mechanism, a concept accomplished differently by GeM2 and Butterfly1; the approach by Dong et al. uses the recently standardized SHA-3 algorithm and using hashes is a common way to generate keys; and, the approach by Liu and Kwak uses AES based key generation, which is one of the concepts included in the IEEE 802.15.6 standard. We implemented them using Java, and use the results obtained to assess the key similarity using Sörensen's Similarity Index [129] and randomness properties using NIST STS [130]. We present a comparison between these results and the results obtained from the same analyses for our proposals (main key configurations only, discussed in Sections 7.2.4 and 7.2.5). We also discuss the security offered by these protocols and how these differ compared to our work.

7.4.2 Implementation Details

We implemented the functionality of the work by Zhu and Khan [8], Dong et al. [9], and by Liu and Kwak [7], using Java programming language [126] and the same development environment summarized in Table 7.1. We describe the details of these implementations in the paragraphs that follow.

Impl. 1: To implement the XTEA encryption mechanism that is a part of the key update cycle in the work by Zhu and Khan [8], we used the Java implementation of the XTEA cipher created by Mueller, available online on Google Code [135].

Impl. 2: We used the implementation of Keccak message digest algorithm (which NIST has published as SHA-3 [136]) made available online by Kocak on Github [137], as part of our implementation of the proposal by Dong et al. [9].

Impl. 3: We used the Cipher class [138], which is a part of the Java Cryptography Architecture [139] for the AES-CBC functionality required for the work by Liu and Kwak [7].

The initial key was set to $92EB8D6ECF7F808A$ in Impl. 3, while in Impl. 1 and Impl. 2, it was set to $92EB8D6ECF7F808A705D1A4566991AF0$. The initialization vector was set to $E36DC751D0433F05$ in Impl. 3. In all implementations, the seed for the PRNG was set to $12345678FEDCBA98$. Furthermore, in Impl. 2, we extracted the timestamp using the standard Java method, $System.currentTimeMillis()$. Our choices for the initial keys, vectors and seeds for PRNG were the same as used for implementations of our proposals discussed in Section 7.2, to ensure uniformity in initial conditions.

We extracted 10240 key sequences from these implementations, and used them to determine the SSI and randomness of the key sequences. We discuss the results obtained from this exercise in the next section.

7.4.3 Key Similarity

Table 7.16 summarizes the average SSI values for 10240 keys for the main key configurations in our proposals, and for those generated from Impl. 1, Impl. 2 and Impl. 3. The numbers represent the average SSI when keys are considered wholly (128 bits) or as four 32-bit blocks. Figure 7.22 illustrates these average SSI values, from which we can observe that our proposals perform at par, if not better than the other proposals. In terms of average full key SSI, all of our proposals ($C1$ — $C7$), outperform the others; an exception with the average full key SSI for $C3$ being very close to the average SSI of the keys generated by Dong et al.'s [9] proposal. When it comes to individual substrings, we observe the same behavior, i.e. our configurations perform at par, if not better than the other proposals (the average substring SSI being 0.3251). The exception to this is GeM2 (configuration $C1$), where we see that the average substring SSI is approximately 0.3326, a behavior we can attribute to the minimal linking of keys between successive key generations.

The keys generated in the proposal by Zhu and Khan [8] includes a scheme of updating the keys block-wise, which retains the other (non-updated) blocks of keys similar to the previous keys. This is the reason why the average SSI - both full key and substrings - are consistently higher (more so in the substrings where we see the

Table 7.16: Summary of similarity between keys

Configuration	Average SSI (SSI_{av})				
(Keys considered)	Full Key	Substring1	Substring2	Substring3	Substring4
C1 (K)	0.3040	0.3371	0.3315	0.3307	0.3312
C2 (K_i)	0.3809	0.3279	0.3237	0.3277	0.3252
C3 (K_S)	0.3816	0.3271	0.3246	0.3259	0.3254
C4 (K, K_i, K_S)	0.3514	0.3125	0.3067	0.3109	0.3103
C5 (K, K_i)	0.3437	0.3285	0.3274	0.3280	0.3288
C6 (K, K_S)	0.3415	0.3287	0.3271	0.3280	0.3263
C7 (K_i, K_S)	0.3812	0.3262	0.3249	0.3255	0.3263
Liu and Kwak [7]	0.3826	0.3256	0.3274	0.3253	0.3252
Zhu and Khan [8]	0.4110	0.6854	0.6866	0.6849	0.6877
Dong et al. [9]	0.3815	0.3267	0.3275	0.3239	0.3257

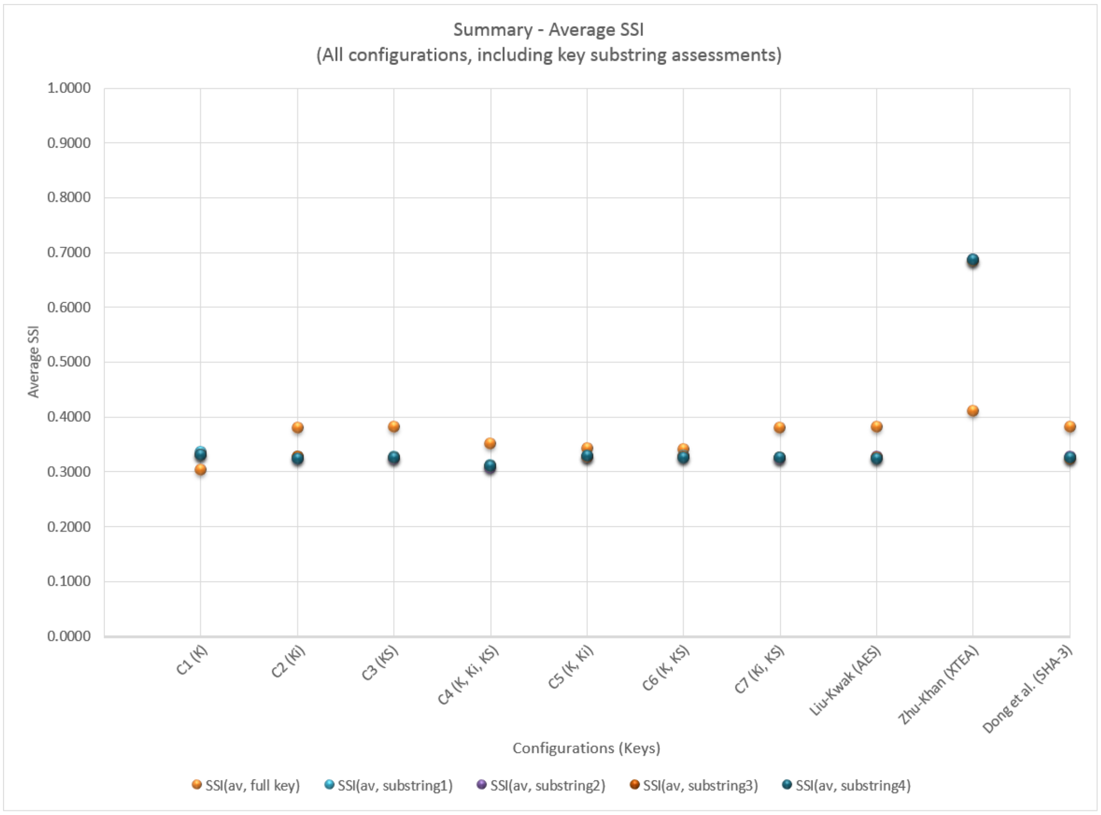


Figure 7.22: Variation in average SSI

average substring SSI being 0.6862). We illustrate the variation in similarity between successive keys (initial 108 comparisons), for the proposals by Liu and Kwak [7], Zhu and Khan [8], and Dong et al. [9] in Figures 7.23, 7.24 and 7.25, respectively.

With the average full key SSI for our proposals (main key configurations only) being in the range $0 \leq SSI_{av,full\_key} \leq 0.3900$, we can conclude that successive key sequences are somewhat dissimilar. Observing the variability of SSI in Figures 7.23, 7.24 and 7.25, and for our configurations in Figures 7.8 — 7.21, we can also infer that the variability in full key SSI is more in our configurations, with increased variability observed in the substring variability as well.

Our assessment of the similarity between keys was primarily to ascertain that our proposals generate variable keys that are dissimilar from the subsequent/previous keys, as a way to increase the overall unpredictability of the system implementing these proposals. One could argue that any scheme implementing a pseudorandom number generator and/or timestamps could inherit the randomness properties they possess. However, we need to ascertain the dissimilarity of the keys generated in such

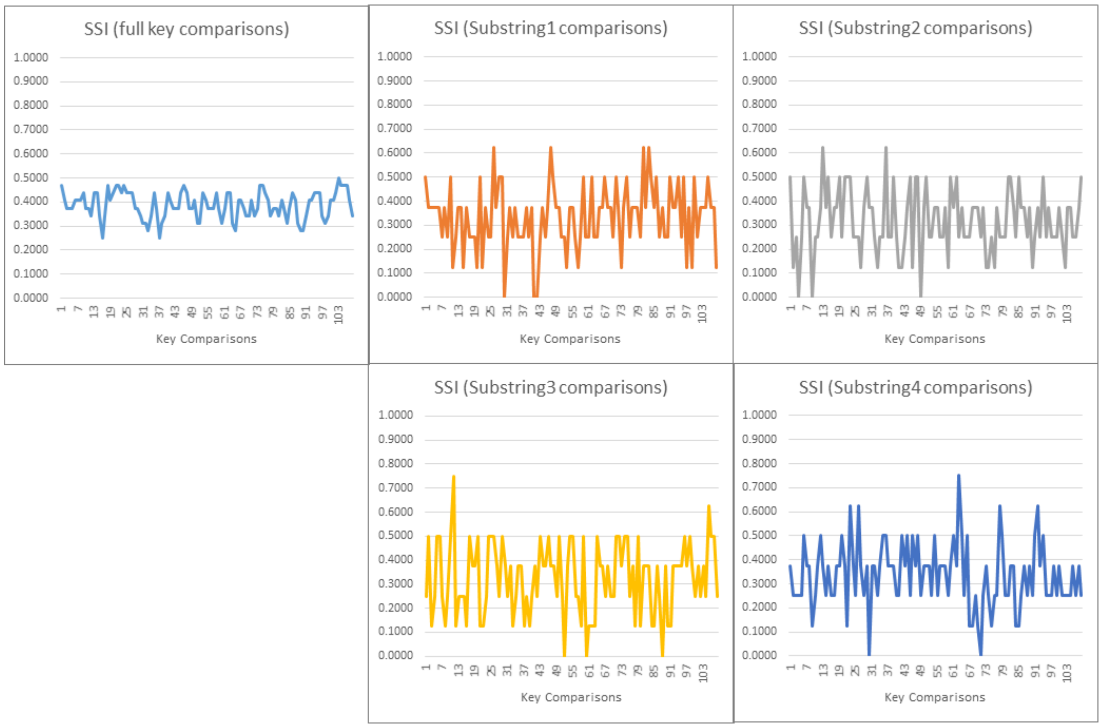


Figure 7.23: SSI Variation for the proposal by Liu and Kwak [7]

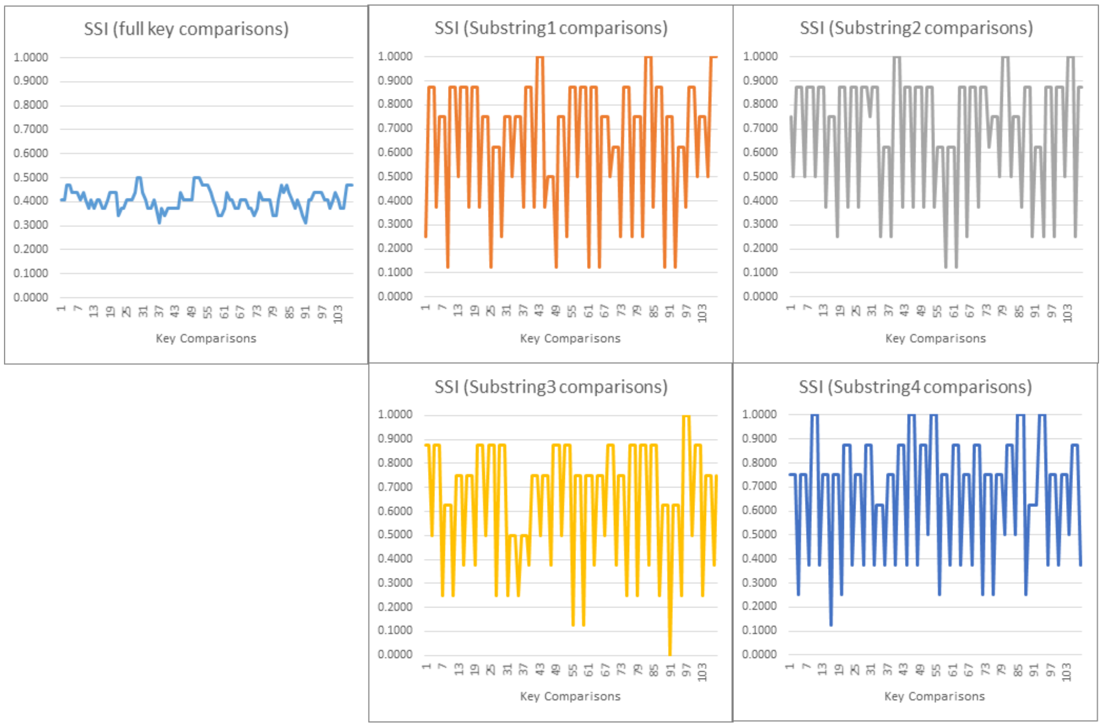


Figure 7.24: SSI Variation for the proposal by Zhu and Khan [8]

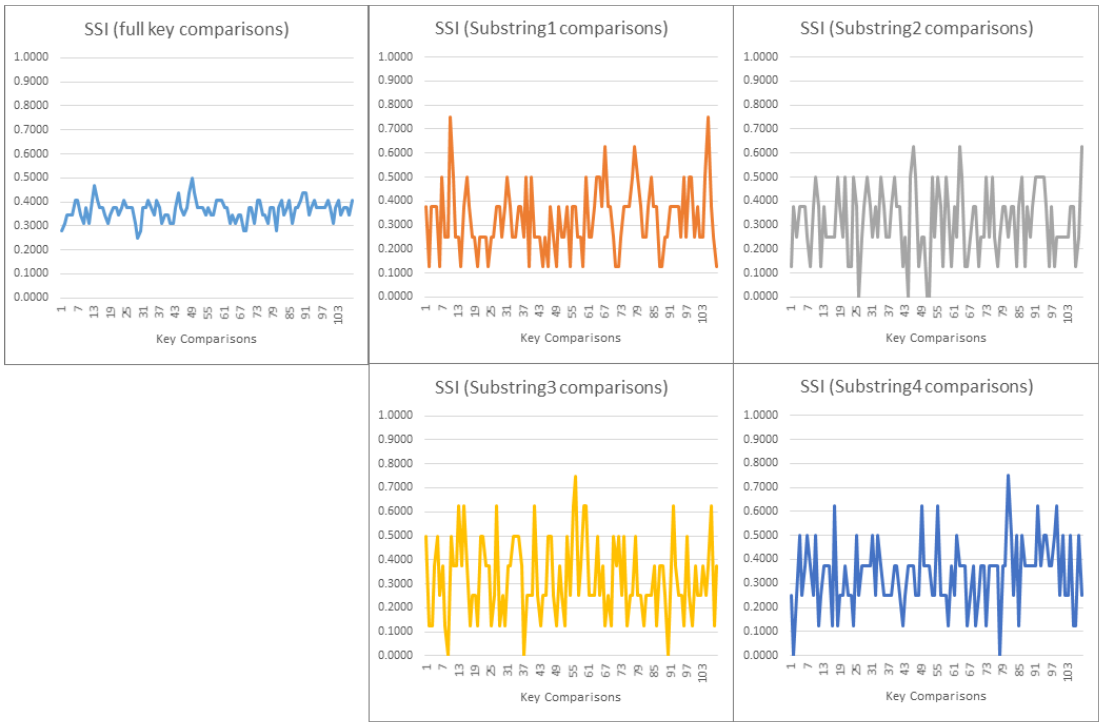


Figure 7.25: SSI Variation for the proposal by Dong et al. [9]

processes as a first step to assess the randomness of the key sequences, which could be impacted by the other logical operations that are used in the proposals. Our results help us conclude that the keys generated are indeed dissimilar, outperforming the keys generated by other proposals.

7.4.4 Key Randomness

As with our assessment discussed in Section 7.2.5, we limited the analysis of the key sequences generated by Zhu and Khan [8], Dong et al. [9], and by Liu and Kwak [7], to the total number of sequences (out of 10000) that passed each statistical test discussed in the previous section. To be consistent in our analysis, we performed the same tests described in Section 7.2.5 and considered a block size of 32 bits for the frequency test within a block.

Table 7.17 summarizes the number of sequences from each configuration that passed each statistical test, recalling that according to Equation (7.4) (on Page 138), the total number of sequences that need to pass a test to be considered as unpredictable or random are 9870 out of 10000 (or, 98.70%).

Table 7.17: NIST STS Assessment: Summary of results (Number of sequences out of 10000 that passed each test)

Configuration (Keys considered)	FM Test[1]	FB Test[2]	Runs Test	LR Test[3]	DFT Test[4]
C1 (K)	4998	5027	5140	5360	9933
C2 (K_i)	9895	9908	9906	9896	9862
C3 (K_S)	9908	9921	9907	9918	9857
C4 (K, K_i, K_S)	8288	8320	8342	8400	9856
C5 (K, K_i)	7555	7564	7623	7716	9885
C6 (K, K_S)	7405	7423	7482	7585	9876
C7 (K_i, K_S)	9885	9907	9880	9905	9837
Liu and Kwak [7]	9913	9914	9888	9912	9837
Zhu and Khan [8]	9891	9892	9892	9886	9832
Dong et al. [9]	9898	9926	9890	9894	9833

1 : Frequency (Monobit) Test

2 : Frequency Test within a Block

3 : Longest Runs of Ones in a Block

4 : Discrete Fourier Transform (Spectral Test)

As observed in Section 7.2.5, GeM2 (and the framework configurations it is a part of) seems to perform poorly with respect to the randomness within each key sequence. We recall that the randomness assessment by NIST STS is based on the number of 1 bits in the sequence measured against the theoretical probability of occurrence of a '1' bit in the sequence (50%). On the other hand, keys generated by Butterfly1 and HiveSec1 pass the requirement of sequences passing each test being more than 9870 for a sample of 10000 sequences. Based on the results in Table 7.17, we also observe that the sequences generated by Butterfly1 and HiveSec1 (and the framework configuration $C7$) perform at par to sequences generated using AES (Liu and Kwak), XTEA with blockwise key update (Zhu and Khan) and using SHA-3 (Dong et al.).

Our assessment of randomness in the key sequences is in addition to the similarity assessment, and is mainly to evaluate how random the bits in the key sequences generated by our algorithm proposals are, which has a direct bearing on the security of the system. The results lead us to infer that the mechanism of key updates used in GeM2, which involves changing the '0' bits in a parent key reduce the overall randomness (or the property of having nearly equal proportion of 0s and 1s in the sequence), although the scheme itself might be secure owing to the random key update mechanism and the random choices involved. We also note that the use of timestamps and combination of multiple parameters to generate keys ensures that the keys are both dissimilar and random. Furthermore, the framework configurations that implement these algorithms tend to improve on the overall randomness of the system, as is evident with the randomness assessment in this section. We can observe that although the overall randomness is lower in configurations $C5$ and $C6$, the average performance (which is a factor of the key sequences generated by all algorithms that are a part of that configuration, including GeM2) is better than in $C1$ (only GeM2). This goes to show that if the system supports, implementation of a multi-algorithm framework such as the one proposed in this thesis ensures sufficient randomness in the key sequences and improves the overall security offered by the system. The main contribution of such a framework, perhaps, is the random choice associated with the algorithms based simply on an internally updated timestamp value. This further adds to the overall unpredictability associated with the system, thereby increasing the security.

7.4.5 Security

Having assessed the performance of our algorithm proposals on their key similarity and key sequence randomness against the proposals by Zhu and Khan [8], Dong et al. [9], and by Liu and Kwak [7], we now discuss how our algorithms fare in terms of the overall security and ability to protect against attacks when compared to these approaches.

Liu and Kwak's protocol requires entities to transmit the random numbers used in the open, in addition to the mandatory source/destination IDs (addresses). This is a potential vulnerability because the session key generation is then only a factor of determining the master key, since the algorithm is known (AES) and the inputs to this algorithm are known (node addresses and the random numbers). The problem then reduces to a brute force attempt to decipher the master key, which might not be a challenging task given that one of the pre-association messages includes transmission of the random numbers and the source/destination IDs, and the response of encrypting a concatenation of these parameters, giving both the input and the output, leaving only the key to be determined. Zhu and Khan's protocol is a server initiated key update mechanism, in which the key update messages sent by the server in their approach includes the following — subkey to be updated, an XOR combination of the subkey and its index, encrypted using the system key, KEY_{SYS}. These messages are not accompanied by any vector to either verify the entity of the sender or to verify the integrity of the message. Although they include a mechanism to restore the old key in the event of a detected attack or non-verification of the server, it does involve the tag performing the system update first. A simple de-synchronization attack scenario for their protocol would be as follows — server/reader sends key update message to tag ($m0$); tag responds with encrypted acknowledgement pattern ($m1$) and generates verification pattern ($m2$). Following this, the attacker waits for the reader to send $m2$ and drops the packet, only to transmit a separate random number in its place. When $m2$ at the tag does not match with the received $m2$, the tag rolls back its update, but the server would have progressed to the next key state. This de-synchronizes the server and the tag states, disrupting future communication. The work by Dong et al. however, is contrary to both these approaches, featuring only hash functions and XOR operations, to exchange and agree on the key. Their approach also features linked key updates, in that the updates to keys are based on using previously synchronized keys for authentication, a threshold of one key stored for de-synchronization attempts.

We can see that the approaches by Liu and Kwak, and Zhu and Khan discussed here are prone to attacks such as brute force key determination, de-synchronization, selective forwarding and dropped frames. Their performance will be affected by these attacks, which could potentially lead to other vulnerabilities listed in Table 2.2. The work by Dong et al. may not be prone to these attacks, however, might not be scalable to other environments such as WBANs.

On the other hand, our proposals maintain all key states a secret, with only the state identifiers being communicated. All proposals include mechanisms to verify either the key, the message or both, by means of signatures, and further ensure that the system can prevent most attacks. Perhaps the biggest strength of our approaches is the increased unpredictability due to the different state parameters used and the presence of an overall framework to choose one of the algorithms at random.

7.4.6 Summary

In this section, we discussed the implementation of three key generation and authentication proposals in RFID and WBAN systems, and presented a discussion on how our algorithm proposals perform against them. We considered the same parameters we used to assess our approaches, i.e. key similarity, key sequence randomness and security analysis. From the results obtained, we can conclude that our algorithms perform better, if not at par with the other approaches considered for comparison. With the exception of GeM2, other algorithms also perform well when considering the randomness in key sequences. The encouraging aspect of these results is that the proposed framework works as expected, i.e. it improves the overall security by unpredictability in the system states and the algorithm choices. Having explored the security offered by our proposals in this section, we discuss resource utilization next, which is another critical factor for resource-constrained wireless networks. Our assessment of resource utilization is two-fold — assessment of resources used in a hardware (optimized) implementation and estimation of gates used in a generic (non-optimized) implementation.

7.5 FPGA Implementation

When security solutions need to be implemented on emerging wireless devices such as RFID tags, they need to be hard-wired, i.e. the logic needs to be implemented as part of the circuit of the tag. Although we have designed our algorithms and the framework in a way as to be able to ‘plugged-in’ with existing circuits, they would still need to be implemented as independent physical modules or as part of the larger circuit of the device. This requires us to assess the hardware resource requirement for each of our configurations.

One way to assess the resource utilization and to assist in implementing such digital logic solutions to hardware is to test the implementation on configurable integrated circuits called Field-Programmable Gate Arrays (FPGA). An FPGA is an integrated circuit with no specific function, which is mainly used to assist in the design of application specific integrated circuits (ASICs) [140, 113, 141]. FPGAs employ logic gates and memory elements (programmable/configurable elements), referred to as Configurable Logic Blocks (CLBs), that can be configured to accomplish the desired functionality using programmable interconnects. Once the design is tested and vetted, it can be used to (design and) manufacture custom ASICs for the particular application. The main difference, thus between FPGAs and ASICs is that FPGAs are re-programmable devices, while ASICs are not.

We used the Spartan-6 FPGA SP605 Embedded Kit [142] manufactured by Xilinx Inc. to implement our framework and its constituent algorithms. The SP605 kit includes in it the Xilinx XC6SLX45T-3FGG484 FPGA and ISE 13.4 Design Suite

Table 7.18: Features of a Slice in Spartan-6 [6]

Feature	SLICEX	SLICEL	SLICEM
6-Input LUTs	✓	✓	✓
8 Flip-flops	✓	✓	✓
Wide Multiplexers		✓	✓
Carry Logic		✓	✓
Distributed RAM			✓
Shift Registers			✓

Table 7.19: Logic resources in one CLB [6]

Slices	LUTs	Flip-flops	Arithmetic and Carry Chains(2)	Distributed RAM(1)	Shift Registers(1)
2	8	16	1	256 bits	128 bits

Notes:

1. SLICEM only.
2. SLICEM and SLICEL only.

Table 7.20: Spartan-6 (XC6SLX45T) logic resources [6]

Logic cells	43661
Total slices	6822
SLICEMs	1602
SLICELs	1809
SLICEXs	3411
Number of 6-input LUTs	27288
Maximum distributed RAM (Kb)	401
Shift registers (Kb)	200
Number of Flip-flops	54576

Embedded Edition IDE. Present day FPGAs, such as Spartan-6, have “Slices” as the main logical component of the CLBs. CLBs in Spartan-6 contain two slices, where a slice is composed of “four logic-function generators (or look-up tables, LUTs) and eight storage elements (or, Flip-Flops)” [6]. The slices in CLBs include one SLICEX and one of either SLICEL or SLICEM, the difference between the three being the inclusion of additional logical elements such as multiplexers, carry logic, distributed RAM and shift registers (The features of each type of slice is detailed in Table 7.18). Furthermore, “interconnect resources” are used to connect the various functional blocks in the FPGA. The Xilinx Synthesis Tool (XST), which is a part of the Xilinx ISE Design Suite optimizes much of the interconnections between logical blocks, allowing users the ability to optionally customize some resource routing, such as clocking. Table 7.19 lists the available logic resources in each CLB, and Table 7.20 summarizes the available resources in the Spartan-6 SP605 kit (with XC6SLX45T) that we used for our assessment.

To implement the functionality of our framework and its constituent algorithms, we used VHDL (Very High Speed Integrated Circuit – VHSIC – Hardware Description Language) [113]. VHDL is a hardware description language (HDL) that helps in modeling a digital system "at many levels of abstraction ranging from the algorithmic level to the gate level" [143]. Designers can either use concurrent or sequential modeling to describe a digital system. The VHDL description of a digital system includes one external view (referred to as the entity) and one (or more) internal views, where an external view is the set of interfaces the device would communicate with other associated devices and the internal view is the actual functionality of the system (referred to as the architecture).

In the next section, we summarize the implementation specifics of modeling our algorithms and framework configurations using VHDL, following which we present the results obtained by synthesizing and deploying them on Spartan-6 using Xilinx ISE.

7.5.1 Evaluation of Hardware Resource Utilization

The objective of our hardware assessment was to first verify the functionality of the system when deployed on FPGA (through simulation) and to assess the approximate resource utility of our approaches when deployed on a reconfigurable logical circuit such as the FPGA. Although our work is a security proposal and not primarily a hardware proposal, our evaluation is an initial attempt to understand the overall resource utility when our framework and the constituent algorithms are implemented on ASICs. FPGA assessment only provides an overview of the resources used by the proposals, and the implementations can be used as foundations for further optimization exercises and as a guideline for implementation on ASICs. In our assessment, we implemented all the algorithms and the framework configurations, each configuration listed in Table 7.2, using VHDL, on a different development environment (PC) than the one used for the proof of concept implementation. The characteristics of the PC used for this assessment are summarized in Table 7.21.

We implemented behavioral models our proposals using VHDL, using separate processes for key generation and serial output extraction. We implemented procedures to accomplish the following functionalities — (a) encryption (using XOR), (b)

Table 7.21: Characteristics / environment of the computer used for the VHDL implementation

Processor	Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz
System type	64-bit Operating System, x64-based processor
Operating System	Windows 7 - Service Pack 1 (build 7601)
Development Environment	Xilinx ISE 13.4
Target Family	Spartan6
Target Device	XC6SLX45T-3FGG484
Simulation Environment	Xilinx ISE Simulator (ISim)

pseudorandom number generation (using a modified implementation of the PRNG algorithm proposed by Melià-Seguí et al. [94, 144], where we used only one of the polynomials and implemented our own decoding logic to generate the pseudorandom number), and (c) Sequential implementation of SHA-1 message digest algorithm (implementation by Rainier, available online on Github [145]). We used XOR for encryption and SHA-1 for message digest only as a proof of concept in this case. We used the PRNG implementation proposed by Melià-Seguí et al. as it has been proposed as a lightweight PRNG for RFID systems. We would like to reiterate that our proposals are generic, and designed to work with any encryption, PRNG and message digest algorithms. We defined these procedures in a VHDL package, which facilitated using the same package definition for all our implementations. This was to ensure consistency in and re-use of procedure definitions.

For the sake of consistency with our proof of concept implementation discussed in Section 7.2, we used the following parameter initializations — for GeM2, the initial key was set to $92EB8D6ECF7F808A705D1A4566991AF0$, and initial seeds to compute the PRNG seed were set to 14930352 and 24157817. In all configurations where Butterfly1 is a component, the PRNG used to choose the value of the variable j at random, which decides the state of the seed (s_j), was initialized to $192BC333250CCCFF$, while the seed (s) itself was initially set to 12345678. In all configurations where HiveSec1 is a component, the six parent seeds of the 'seedhive' were initialized to $21365448FEA32DE0$, $F40925AB6B446781$, $E82745AEF95112dDC$, $67A8366EFC8CC294$, $48656CBCEAA36291$ and $998163AFE2A88A0A$, while the seed for the PRNG when used to choose the value of s_e was set to $12345678FEDCBA98$.

Furthermore, we synchronized all data processing and input/output in our implementations with the clock ($33MHz$) available on the Spartan-6 FPGA. This was an exercise in optimizing the usage of available resources and to synchronize serial input/output and data processing cycles. We used serial (bit-wise) input/output in our implementation since each configuration/algorithm has different number of bits that are output and because the FPGA has a maximum of 296 input-output blocks available for use.

Our assessment was aimed at establishing that the presence of only lightweight logical operations in our schemes would make them lightweight additional modules, to systems that may already have implementations for encryption and hash operations. This prompted us to use XOR as the encryption function, and available implementations for PRNG and the SHA-1 hash algorithm to realize the functionality and assess the resource utility. We summarize the resource utilization results in the next section.

7.5.2 Results

Our first task after implementing our algorithms and framework configurations using VHDL was to assess the functionality. For this, we simulated the algorithms using the Xilinx ISE Simulator (ISim). Waveforms generated by simulation of each configuration, representing the inputs and outputs, can be found in Appendix B.

The estimation of hardware used by an algorithm helps us assess the logic blocks (i.e. LUTs, Flip Flops, etc.) that are utilized in the implementation. This helps in approximating resources that would be required on an ASIC implementation of the same, thereby becoming a contributor to its size, utility and cost. When we deployed our algorithms on the Spartan-6 FPGA, the PAR (Xilinx Place and Route) tool generated a summary of device utilization, listing the available resources and the amount of resources used by the particular implementation.

We present three results in this section obtained from this exercise:

- HDL synthesis summary (Table 7.22) — This summarizes the logic gates used in the implementation. This provides the un-optimized number of standard basic gates, and logical blocks such as registers, comparators and multiplexers required during the initial step of the HDL synthesis process. Synthesis is a process where the Xilinx Synthesis Tool converts the abstract VHDL model of each configuration into an implementation in terms of logic gates [146].

- Advanced HDL Synthesis summary (Table 7.23) — This provides the optimized resource usage. This is the result of further optimization by XST, where it reduces the logic blocks used by 'trimming' Flip-Flops and latches, reducing redundant registers, setting constant values (assumed in our implementations for accomplishing the server/reader/WBAN hub aspect of the communication), and converts the logic gate design obtained in the previous step into one that uses registers and LUTs [146].

- Device utilization summary (Table 7.24) — This summarizes the approximate number of slices/slice resources that are used in an optimized execution of the various configurations, possibly affected by the synchronization with the clock, taking into account assumed inputs, with optimizations for constant values.

Table 7.22 provides the logical resources that are estimated by XST in the initial synthesis step. This accounts for all the resources needed for a parallel/concurrent implementation of the various operations in each configuration. This takes into account logical/arithmetic operations such as XOR, addition, comparisons, condition checks (implemented using multiplexers), and registers used for storage and to represent signals that carry data from one sub-process to another in each implementation.

One thing that stands out in Table 7.22 is the low resource use by our Butterfly1 proposal; an attribute that can be observed in the advanced HDL synthesis summary (Table 7.23) as well. This 'curious case of Butterfly1' can be attributed to the simplicity in the design of the algorithm. This helps us accomplish the desired functionality, while retaining unpredictability, and using less resources on hardware. This enables us to claim that not only is Butterfly1 secure in terms of the randomness in keys generated (Section 7.2.5) or its ability to mitigate attacks (Section 7.3), but it is also truly not resource intensive. This makes it a prime candidate for implementation on resource-constrained devices. This non-resource intensiveness property can also be observed in HiveSec1, which consumes slightly more resources than Butterfly1. However, GeM2, which requires storing previous states for synchronization purposes, requires more storage and addition operations, which is mainly due to the Fibonacci-like mechanism of seed updates.

Table 7.22: FPGA Implementation: HDL Synthesis Summary

Parameter	Configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
Adders/Subtractors							
128-bit adder	1	0	0	1	1	1	0
32-bit adder	304	0	1	611	305	611	307
4-bit adder	2	0	0	2	2	2	0
Registers							
1-bit register	1	3	3	1	1	1	1
2-bit register	0	0	0	2	2	2	2
4-bit register	2	0	0	4	4	4	0
32-bit register	0	0	1	4	2	3	4
128-bit register	4	0	0	5	5	4	1
296-bit register	2	0	0	0	0	0	0
358-bit register	0	0	2	0	0	0	0
530-bit register	0	1	0	1	1	0	1
Comparators							
4-bit comparator greater	1	0	0	1	1	1	0
Multiplexers (MUX)							
128-bit 2-to-1 MUX	2	0	0	2	2	2	0
4-bit 2-to-1 MUX	3	0	0	4	4	4	0
1-bit 2-to-1 MUX	0	0	1	0	0	0	0
XORs							
352-bit XOR2	0	1	1	2	1	1	2
128-bit XOR2	4	4	1	9	9	5	5
128-bit XOR3	4	4	1	1	0	1	01
32-bit XOR2	31	0	0	62	31	62	31
32-bit XOR3	42	0	0	84	42	84	42
32-bit XOR4	51	0	0	102	51	102	51
1-bit XOR9	1	2	1	4	4	1	2
1-bit XOR3	0	0	0	0	0	0	1
1-bit XOR2	0	8	0	0	0	0	0

Table 7.23: FPGA Implementation: Advanced HDL Synthesis Summary

Parameter	Configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
Macro Statistics							
Accumulators (128-bit)	1	0	0	1	1	1	0
Adders/Subtractors							
128-bit adder	1	0	0	1	1	1	0
32-bit adder	0	0	0	1	1	1	1
4-bit adder	1	0	0	1	1	1	0
XORs (128-bit)	0	0	0	1	1	1	1
Counters (32-bit,Up counter)	0	0	0	1	1	1	1
Comparators (4-bit)	1	0	0	1	1	1	0
Multiplexers (4-bit 2-to-1)	2	0	0	2	2	2	0
Registers (Flip-Flops)	135	3	3	143	143	143	7

The resources required by the framework configurations are largely dependent on the resources consumed by the individual algorithms that they include. While the resources required by configurations $C4$, $C5$ and $C6$ are higher than $C7$, we can observe that this is decided by the requirements of GeM2 in the former, and reduced in the latter due to the requirements of Butterfly1 and HiveSec1.

XST optimization is mainly to ensure that the resources used by the circuit design is faster and consumes less logical elements. This optimization is an implicit step in Synthesis and Implementation stages, which provides an estimate of required resources. The optimization is notable in Table 7.23, where we see that the large number of adders and registers, especially in the configurations employing GeM2, are considerably reduced (as compared to Table 7.22). This takes into account the reduction in redundant logical blocks and re-use of the various available resources on the FPGA for accomplishing the required functionality in each configuration.

Table 7.24: FPGA Implementation: Summary of results (resources used in an optimized implementation)

Parameter	Configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
Device Utilization Summary							
Slice Registers (available:54,576)	9	0	3	17	17	17	10
Slice LUTs (available:27,288)	8	0	1	19	19	19	9
Occupied Slices (available:6,822)	4	0	3	10	10	10	7
MUXCYs used (available:13,644)	8	0	0	16	16	16	8
LUT Flip Flop pairs (available)	6 (10)	0 (3)	1 (3)	13 (22)	13 (22)	13 (22)	6 (13)
Bonded IOBs (available:296)	3	3	3	5	5	5	5
BUFG/BUFGMUXs (available:16)	1	0	1	1	1	1	1
Approximate Logic Cell and ASIC Gate Equivalent							
Logic Cells (available:43,661)	26	1*	19	64	64	64	45
ASIC Gate Equivalent	390	15*	285	960	960	960	675

* Logical blocks occupied by the Butterfly1 implementation (after it is optimized and packed by the Xilinx PAR tool) is less than the resources of 1 complete slice

Following synthesis, we used the Xilinx PAR tool to deploy our designs on the FPGA. This tool uses the optimized design generated by the Synthesis tool and maps the design to specific circuit elements on the FPGA. This is done by using specific sections on the FPGA as LUTs and registers, and by using other elements such as the clock source and input/output pins for other aspects of the circuit. The PAR tool provides the summary (Table 7.24) of design elements on the FPGA used for implementing said designs.

We observe from Table 7.24 that the resources (slice registers, slice LUTs and MUXCYs) utilized by our implementations are significantly lower ($\leq 1\%$) than the available resources on the Xilinx Spartan-6 FPGA. Note that MUXCY is a multiplexer that is a component of a slice that helps in choosing one of the available inputs based on the prevailing conditions. LUT Flip Flop pairs are allocated for each design and we observe that of the total allocated, the utilization remains less than 60%. Note that the number of available LUT Flip Flop pairs are allocated based on the total slices occupied by the deployed design. Bonded IOBs are input-output blocks that are assigned to our implementations during placement and routing. We note that the total bonded IOBs in our implementations are less than 2% for all configurations, while the clocking resources (global clock buffer, BUFG, or multiplexed global clock buffer, BUFGMUX) used are less than 7%. The main reason why less resources are used in our algorithms is the re-use of procedures for encryption and pseudorandom number generation. This reduces the overall resource utility, which is reflected both in the 'Macro statistics' in Table 7.23 and 'Device utilization summary' in Table 7.24.

An aspect that stands out among the results in Table 7.24 is the efficiency in resource utility by our Butterfly1 proposal (observed in HDL synthesis as well, Tables 7.22 and 7.23). This is largely possible owing to the re-use concept that is central to this approach. By re-using PRNG and other operations for accomplishing various functions in the protocol, the strain on the hardware is remarkably less. Following Butterfly1 in terms efficiency are configurations $C3$ (HiveSec1) and $C7$ (framework with Butterfly1 and HiveSec1). We must note that although the occupied slice estimate for Butterfly1 is listed as 0, it is an approximate value rounded to the nearest integer by Xilinx ISE. This just goes to show that the logical blocks occupied by the Butterfly1 implementation (after it is optimized and packed by the Xilinx PAR tool) is less than the resources of 1 complete slice. The results further support the concept that re-using circuit components for various functions reduce the overall strain on the device. In contrast, the configurations featuring GeM2 have a larger utilization of the available resources, even though the utilization is less than 1% of the available resources. In terms of registers (Flip-Flops), however, the utilization by these configurations is higher, something that can be attributed to the storage of three (two previous and one current) system states for synchronization purposes.

The number of occupied slices in the design report (Table 7.24) translates approximately to the number of logic cells that the design might occupy when deployed as part of a larger system-on-chip (SOC), or as an independent ASIC. A logic cell is a "logical equivalent of a classic four-input LUT and a Flip Flop" [147]. This is a way to measure the device capacity. On the FPGA we have used, Spartan-6 XC6SLX45T, there are a total of 43661 logic cells available, which translates to approximately 1.6 logic cells per LUT [6], or approximately 6.4 logic cells per slice (given that there are 6822 slices and 27288 LUTs in total). To compute the approximate logic cell equivalent in Table 7.24, we consider the formula, 1 *slice* $\approx$ 6.4 *logic cells*, since a slice is a unit that encompasses LUTs and Flip Flops (we round the result to the nearest integer).

Furthermore, it is to be noted that the number of gates in an ASIC design helps us approximate the area that the design might require on the ASIC. It has been estimated that, 1 *logic cell* $\approx$ 15 *ASIC gates* [148]. This helps us approximate the required number of ASIC gates for each configuration. ASIC gate equivalents or NAND gate equivalents of the logic blocks denote the approximate number of NAND gates required to realize the circuit (measured in terms of NAND gates because it is functionally complete, i.e. all logic blocks such as AND, OR, etc. can be realized using NAND gates) [113]. We note that all our configurations have an approximate gate count of less than 1000 gates, noting that these counts include key generation function, encryption function (XOR) and message digest function (SHA-1), thereby being a complete encryption scheme. These gate count estimates are very much within the range of 200 - 3000 gates, which is suggested to be the available gates for security solutions on resource-constrained devices [149, 150, 151]. Even if one were to implement other encryption algorithms (in place of XOR) or a different message digest algorithm (in place of SHA-1), the gate count estimates would not be significantly affected by our key generation algorithms (and framework). To provide some perspective for these gate equivalents, we have included the gate equivalents of two algorithms included in the cryptographic suite specifications as part of ISO/IEC

Table 7.25: Gate Count Estimates

$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$	PRESENT [75]	Grain128 [78]
390	15	285	960	960	960	675	1570	1857

Table 7.26: Logic Circuit Estimation (generic)

Logic	Configurations						
Circuit	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
AND	n-bit	-	n-bit	n-bit	n-bit	n-bit	n-bit
OR	-	-	n-bit	n-bit	-	n-bit	n-bit
NOT	n-bit	1-bit	-	n-bit	n-bit	n-bit	n-bit
XOR	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit
Addition	n-bit	$\frac{n}{2}$-bit	n-bit	n-bit	n-bit	n-bit	n-bit
Division	-	-	$\frac{n}{2}$-bit	$\frac{n}{2}$-bit	-	$\frac{n}{2}$-bit	$\frac{n}{2}$-bit
MUX	γ	γ	γ	γ	γ	γ	γ
PRNG	α	α	α	α	α	α	α
Hash	β	-	β	β	β	β	β
TOTAL	$4n + x_1$	$\frac{3n+2}{2} + x_2$	$\frac{9n}{2} + x_1$	$\frac{11n}{2} + x_1$	$4n + x_1$	$\frac{11n}{2} + x_1$	$\frac{11n}{2} + x_1$

$x_1 : \ \alpha + \beta + \gamma$; and, $x_2 : \ \alpha + \gamma$

29167-1:2014 [33], namely PRESENT [75] and Grain128 [78], and presented the comparison in Table 7.25. Note that all results presented in Table 7.25 are for a key size of 128 bits. We chose to include the gate equivalents reported by the respective publications as a way to compare our results with what has been included as part of the standard specification. This lets us ascertain that the overhead of our proposals on resource-constrained devices or other devices will be considerably less as compared to conventional approaches.

FPGA implementation, as observed here, gives an estimate of the total resources that might be used by the various configurations under consideration. If one were to consider a non-optimized design, i.e. a straightforward implementation of the logic circuits in our algorithm proposals, one could estimate an approximate number of logic gates required to realize the desired functionality. For this purpose, we summarize (generically, for a key size of n bits) the various functions and logical operations that are required to realize the functionality of our proposals in Table 7.26. This is a generic estimate, different from the initial synthesis summary presented in Table 7.22, which lists the required logical elements without considering the potential for re-use of certain elements.

Note that with details about number of slice registers and slice LUTs used, the Xilinx XST and PAR tools provide an estimate of the resource utilization after synthesis and following the placement of the design on the FPGA, respectively. However, with some aspects in our implementations being constant, and XST optimizing the use of registers and LUTs, the device utilization summary might not be a very accurate representation of the resources used/required for ASIC implementations of our proposals. Although we have provided an estimate of the number of gates for each configuration, these gate count estimates must be taken with a grain of salt (something also considered to be an ‘unreliable’ source of resource estimates [153, 154]), as other design constraints in actual ASIC implementations (such as the way a temporary clocking signal can be deployed on an RFID tag) might necessitate the use of more gates/logical elements. These are approximate values and can only be used as initial estimates, prior to a comprehensive assessment of actual ASIC resource usage.

A Note on Energy and Memory Consumption

An aspect that has been included in the HDL Synthesis summary in Table 7.22 is memory consumption. XST summarizes the optimized memory consumption by the various algorithm configurations in terms of register usage. This is a way to estimate the number of registers that would be required when implementing our approaches on ASICs, and hence, the memory overhead that they would impose on the device. The summary of results presented in Table 7.24 indicate that our approaches are memory efficient, although one needs to consider that the results presented here are from an optimized implementation, following a stage of removal of redundant registers and registers containing constant values (note that some registers contained constant values in our implementation since we had to assume some of the functionality of the resource-constrained device).

It should be noted that our FPGA implementation was only for the security proposals presented in this thesis, and without the actual functionality that may be present in either an RFID tag or a WBAN sensor. Thus, if one were to consider a

Table 7.27: Memory Consumption Estimation (generic, in bits)

Parameter	Configurations						
	$C1$	$C2$	$C3$	$C4$	$C5$	$C6$	$C7$
ID	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit
Message	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit
Keys	$4 \times n$-bit	$2 \times n$-bit	$2 \times n$-bit	$8 \times n$-bit	$6 \times n$-bit	$6 \times n$-bit	$4 \times n$-bit
Seeds	$2 \times n$-bit	n-bit	$6 \times n$-bit	$9 \times n$-bit	$3 \times n$-bit	$8 \times n$-bit	$7 \times n$-bit
Timestamp	–	$1 \times t$-bit	$2 \times t$-bit	$4 \times t$-bit	$2 \times t$-bit	$3 \times t$-bit	$4 \times t$-bit
p	3×8-bit	–	–	3×8-bit	3×8-bit	3×8-bit	–
g	3×8-bit	–	–	3×8-bit	3×8-bit	3×8-bit	–
i	–	1×8-bit	–	1×8-bit	1×8-bit	–	1×8-bit
j	–	$1 \times n$-bit	–	$1 \times n$-bit	$1 \times n$-bit	–	$1 \times n$-bit
MC	–	1×2-bit	1×2-bit	2×2-bit	1×2-bit	1×2-bit	2×2-bit
δ	–	–	n-bit	n-bit	–	n-bit	n-bit
$n_{\delta,ideal}$	–	–	$\frac{n}{2}$-bit	$\frac{n}{2}$-bit	–	$\frac{n}{2}$-bit	$\frac{n}{2}$-bit
SSN	–	–	1×8-bit	1×8-bit	–	1×8-bit	1×8-bit
$pattern_{asv}$	n-bit	–	–	n-bit	n-bit	n-bit	–
asv	n-bit	–	–	n-bit	n-bit	n-bit	–
ack	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit	n-bit
TOTAL	$11n + 48$	$8n + 10$	$\frac{29n}{2} + 10$	$\frac{57n}{2} + 68$	$17n + 58$	$\frac{47n}{2} + 58$	$\frac{41n}{2} + 20$

generic design of our approaches, the various parameters/components that would contribute to the overall memory consumption would include all the permanent storage parameters (such as ID), variable parameters (such as keys, timestamps), and state variables. We summarize such parameters and the estimated memory consumption for each configuration under test in Table 7.27. Note that although messages could have variable length, for this assessment, we have assumed the message length as n bits. We have also assumed the length of the timestamp to be n bits.

Hardware and storage resource requirements of an algorithm impact the overall energy consumed by a device implementing the algorithm since the device has to expend energy in order to perform the required operations. Typically, a resource-constrained device (such as an RFID tag or a WBAN sensor) would be in the off or sleep state, and when active, it might need to expend energy to send/receive messages, access memory blocks for data retrieval/storage and for any additional processing (e.g. security, signal amplification, etc.) [155, 156]. The resources available on the device (including the modules responsible for the electrical power), thus, are directly responsible to the amount of energy requirement of the device.

The International Bureau of Weights and Measures defines 1 watt (W) of electrical power as the "power which in one second gives rise to energy of 1 Joule (J)" [157]. Given that electrical power (P) is defined as the amount of work done by an electric charge per unit time (Equation (7.6)), the amount of energy generated per watt is computed as specified in Equation (7.7).

$$P = V \cdot I \tag{7.6}$$

$$E = P \cdot t = V \cdot I \cdot t \tag{7.7}$$

Here, V is the electric potential or voltage (measured in volts), I is the electric current (measured in amperes, A) and t is the time (measured in seconds).

If n is the number of bits of data of message, M, to be transmitted and ω is the data rate (measured in bits per second, bps), the total time (t_{T_x} in seconds) required for transmission of M can be computed as:

$$t_{T_x} = \frac{n}{\omega} \tag{7.8}$$

If the resource-constrained device transmits ν such data packets, then, the total transmission time will be:

$$t_{T_x,total} = \frac{n}{\omega} \times \nu \tag{7.9}$$

This is the total time for transmitting/relaying ν data packets, considering that the time required for packet assembly (following sensing in case of WBANs) is negligible. Following packet assembly, the time required for performing the *security action*[1] as decided by our framework or one of the constituent algorithms can be considered to be $t_{security}$. Thus, the total time required by a resource-constrained entity to perform the security action and transmission can be given by:

$$t_{total} = t_{security} + t_{T_x,total} + t_{rw} = t_{security} + \frac{n\nu}{\omega} + t_{rw} \tag{7.10}$$

Here, t_{rw} represents any minimal time required for data read/write operations from/to memory.

Using the value for the total time from Equation (7.10) in Equation (7.7), we can derive the total energy consumed by the resource-constrained entity as summarized by Equation (7.11).

$$E_{RC} = V \cdot I \cdot (t_{security} + \frac{n\nu}{\omega} + t_{rw}) \tag{7.11}$$

Several factors can impact the total time, t_{total}, and consequently, the energy, E_{RC}, in a real-time resource-constrained application. This would first of all be impacted by the type of resource-constrained application (i.e. RFID, WBAN, etc.), in addition to any other application specific restrictions (such as limits on the maximum packet size, data rate or available electric power). If a single packet ($\nu = 1$) of size, $n = 128$ bits, were to be written to the memory by an Impinj® Monza® X-8K Dura RFID tag (which draws a nominal $80\mu A$ of current at $1.6V$ during write operation) [158], the energy consumed for this write operation (assuming $t_{rw} = 3.6ms$) can be calculated as:

$$E_{RFID,write} = 1.6 \times 80 \times 10^{-6} \left(t_{security} + \frac{128 \times 1}{400 \times 10^3} + 3.6 \times 10^{-3} \right) J$$

$$E_{RFID,write} = 0.128 \times 10^{-3} \left(t_{security} + 0.32 \times 10^{-3} + 3.6 \times 10^{-3} \right) J$$

$$\therefore E_{RFID,write} = 0.128 \left(t_{security} + 3.92 \times 10^{-3} \right) mJ$$

[1] A security action constitutes the choice of algorithm, key generation, encryption and authentication vector/message signature generation in case of our framework.

Similarly, if the same packet were to be transmitted by a WBAN sensor employing the IEEE 802.15.4 standard specification for communication, having PHY layer properties for North America, i.e. with a maximum data rate of $40kbps$ and a transmission power of $1mW$ (min) [159] (assuming $t_{rw} = 3.6ms$), the energy can be computed as:

$$E_{WBAN,T_x} = 1 \times 10^{-3} \left(t_{security} + \frac{128 \times 1}{40 \times 10^{-3}} + 3.6 \times 10^{-3} \right) J$$

$$\therefore E_{WBAN,T_x} = \left(t_{security} + 6.8 \times 10^{-3} \right) mJ$$

Even though the energy consumption by the resource-constrained device is directly proportional to $t_{security}$, we expect that the overhead imposed by the $t_{security}$ component, which represents the time required to generate keys, encrypt/decrypt data and generate authentication parameters, will be low due to the operations in our approaches being simple logical/arithmetic operations. However, the exact impact will depend on the type of application and any application-specific restrictions on the resource-constrained entity.

7.6 Summary

Our evaluation was mainly aimed at establishing the security offered by the various configurations listed in Table 7.2, and to estimate the resources required for implementing them on ASICs. Our assessment of security and resource utility was preceded by verification of the concept and assessment of the ability of the framework configurations to choose its constituent algorithms, at random, with equal probability. This was mainly to ensure that by using the mechanism to select the algorithm (described in Chapter 4), the framework is not biased towards any specific algorithm included as its constituent. Our assessment enabled us to conclude that the choices of algorithms are reasonably spread out, i.e. algorithms having almost an equal probability of being chosen. An aspect associated with our framework configurations is also the possibility of the same algorithm being chosen for consecutive key generation processes, which puts the onus on ensuring continuous key updates on each individual algorithm. With our assessment of key uniqueness, we were able to establish that the main keys used for encryption are always updated with each new message, which when combined with possible choice of one in n available key generation algorithms, provides sufficient unpredictability to the system, thereby increasing the overall security.

To assess the security of the key sequences themselves, we employed two assessment strategies — first, we evaluated the similarity between keys, to determine how similar consecutive keys were. This assessment showed that our proposals are able to generate keys that are less similar to other keys in their vicinity (i.e. preceding/succeeding keys). Following this assessment was our attempt to verify the randomness of each key sequence using NIST Statistical Test Suite (STS). This assessment illustrated the ability of our algorithms to generate keys with high randomness properties, although keys generated by our GeM2 proposal, and framework configurations it is a part of, were not successful in this assessment. This is primarily due to the manner in which keys are updated in GeM2, i.e. mutations or changes are introduced to the '0' bits in parent keys, keeping child keys related to the parent, but not to each other, therefore reducing the proportionality between '1' bits and '0' bits in a key sequence and thus, its randomness. The NIST STS assessment also illustrated the ability of the framework configurations to increase the overall randomness, despite the inclusion of GeM2. This, and the standalone randomness of the framework when GeM2 is not considered, displayed the ability of our framework to add an additional 'layer' of unpredictability over that associated with each individual algorithm. This helps us justify our claim that our framework increases the overall security of the system.

Our assessment also included evaluating the security of the communication protocol and the behavior of each configuration with respect to known attacks on resource-constrained wireless networks. These helped us assess the ability of devices employing our algorithms to operate in synchronous manner, ensuring message confidentiality, along with other security goals such as message (and key, in HiveSec1) integrity verification, mutual authentication and non-repudiation by association. The satisfaction of these security goals implies that our proposals are able to, in most cases ward off and in the other cases mitigate the effects of possible attacks on resource-constrained wireless networks. The security offered is the result of the employing continuous key update mechanisms in each algorithm and hence, the framework, and due to the ability of these key generation mechanisms to generate authentication parameters. Furthermore, the ability of BeeSwarm in HiveSec1 to attack the attacker, albeit in its own limited manner, adds to the overall security and becomes a last-ditch attempt to mitigate attacks in case of persistent attacks.

However, the security offered by our proposals could also be impacted by the required operations (arithmetic and logical) and storage, which led us to evaluate the hardware resource utilization of each configuration. We did so by deploying behavioral models of the configurations on the Xilinx Spartan-6 FPGA. Although this has been suggested to not be a reliable way to assess resource utility by algorithms, it serves as a way to estimate the required logic blocks for each configuration, becoming an initial step toward deployment of these proposals as part of ASICs. This hardware resource utility assessment enabled us to justify that the resources required by our proposals are significantly less as compared to algorithms suggested for use by the ISO/IEC 29167-1:2014 [33] standard specification for RFID systems. Furthermore, with the proposals being largely composed of logical operations, with very minimal use of arithmetic operations, we can also claim that our proposals consume less resources as compared with the more sophisticated security proposals, such as AES and RSA (suggested by IEEE 802.15.6 [66] specification for WBAN systems).

Thus, our assessments justify our claims of security with low resource utility for all algorithms and framework configurations. With low resource utility, the associated implementation costs will be reduced, allowing manufacturers to be able to include additional functionality for the applications. This, along with increased security, makes our proposals ideal candidates for resource-constrained devices, possibly reducing the need for the trade-off between security, functionality and longevity, in such applications.

Chapter 8

Discussion

8.1 Proposed Framework: A Summary

We set out to propose a reconfigurable framework for security in resource-constrained wireless networks that would provide a mechanism of dynamic key and other parameter updates, as an attempt to increase security in these devices through unpredictability. Each individual algorithm proposed in our work accomplishes the objectives of modularity, dynamic (and continuous) key generation, facilitating mutual authentication, and being deterministic only to authorized entities. These algorithms are designed to work as standalone security solutions in various resource-constrained applications. They also form critical components of our proposed security framework.

Our framework combines these algorithms that bring with them varying levels of unpredictability[1]. With its own ability to choose one of the available algorithms at random while maintaining a constant length of the transmitted messages, the framework increases the uncertainty associated with the system, thereby increasing the security.

Our research was motivated by a need to remove key exchange messages, while facilitating resource-constrained devices to be able to dynamically choose from a set of available algorithms for various aspects of security. This ability of dynamically being able to choose algorithms is similar to the framework reconfiguring itself with each message. The reconfigurable behavior makes our framework *metamorphic*, giving an illusion as though the framework is changing its structure at random. In the sections that follow, we discuss the various aspects of our framework and its constituent algorithms, describing how they impact our research goals and hypotheses.

[1]Level of unpredictability of an algorithm, in this context, is determined using the number of internal states that are updated without parameter exchange. For example, HiveSec1 has a higher level of unpredictability since each parent seed has the ability to generate up to 6 or 18 child seeds depending on the timestamp. This is closely followed by GeM2 having internal PRNG seeds updates based on the linear recurrence equation (Equation (5.4)) that are never exchanged, and Butterfly1 with its modifications decided at random to an internal PRNG seed.

8.1.1 Algorithm choice

With the decision to choose algorithms based on the timestamp, its own ID and a constantly incrementing number, all algorithms that form a part of the framework have an equal chance of being chosen for their specific function. This was observed in our evaluation of the algorithm choices in Section 7.2.2. When we considered a large number of key sequences ($n = 10240$), we found that the average probability of the framework choosing each constituent algorithm is nearly equal, although when consecutive key generations are considered, we observe that there could be times when the framework chooses the same algorithm for more than one communication. This is observed more when the choice of algorithms reduces (configurations $C5$, $C6$ and $C7$) as compared with configuration, $C4$. This does not affect the security of the system in any way, since the independent algorithms possess the ability to continually update the various parameters, such as encryption keys and authentication vectors. This ensures that even if a few consecutive algorithm choices could be the same, the security still remains high owing to changing states. Furthermore, with the algorithm choices being decided by the bits of the timestamp, this behaviour need nor be what will be observed in another test instance. This, in addition to the results in Section 7.2.2, confirms our hypothesis ($H1$) that in a large number of trials, the algorithms have an equal chance of being chosen and ensure high security of the system.

8.1.2 Dynamic key generation

When we consider our framework, dynamic key generation is more of an inherited attribute, mainly dependent on the ability of the constituent algorithms to generate keys dynamically. Another aspect to consider in evaluating the dynamic nature of keys generated by our framework are the type of keys, and how frequently they are expected to be updated. For example, long-term session keys, such as the outer envelope key in HiveSec1, K_O, or the transfer key in Butterfly1, K_T, might only change depending on session updates or the value of the Butterfly1 seed update parameter, j, that is chosen at random. This does not affect the security of the system since these are keys used to encrypt an already encrypted set of data elements. These keys need not be frequently updated, since their updates are ensured to be at random. The main keys, on the other hand, such as K in GeM2, K_i in Butterfly1 and K_S in HiveSec1 are expected to change with each key generation, as proposed in the description of the algorithms. This expected behavior was consistent with the results discussed in Section 7.2.3, where we found that the individual algorithms are able to generate new keys for each key generation.

Another aspect we observed in our evaluation of dynamic key generation was that the delay duration (chosen at random between 0 and 2 seconds) led to less than 100% unique key generation by Butterfly1 and HiveSec1 (Table 7.4 on Page 121). This is possible due to the timestamp being a main factor in determining the keys chosen. This behavior remains consistent when we observe the nature of the main keys dynamically generated by the framework configurations. We set the delay to a random choice between 0 and 2 seconds (both 0 and 2 included) to observe the effect of multiple messages being transmitted within a fraction of a second. Despite this possibility being very minimal in resource-constrained applications, we wanted to explore how the framework (and the algorithms) would behave in such a scenario. The impact of this behavior is very minimal, if at all, with less than 0.1% of the keys generated being the same. In a long-term communication between entities, this effect will be negligible, and possibly even non-existent depending on the random choice of algorithms by our framework. The results obtained by our assessment of the dynamic nature of generating main encryption keys in the individual algorithms and the framework support our hypothesis, $H2$.

8.1.3 Key unpredictability

Unpredictability is an attribute that is perhaps central to our framework. This is an aspect that defines the security of our proposals, since security of a published cryptographic technique is dependent on the nature of the keys used. This is defined by how related keys are to preceding and subsequent keys, and the randomness of each individual key sequence. We chose two methods to assess the unpredictability of the key sequences generated by our proposals based on this. First, we assessed the similarity between consecutive keys using the Sörensen's Similarity Index (SSI) [129], which was a way of quantifying how similar (and therefore, dissimilar) keys were to their preceding and subsequent keys. Following this, we assessed the average randomness of key sequences based on statistical tests that are part of the NIST

Statistical Test Suite (STS) [130]. The randomness assessment was mainly to ensure that the key sequences were both sufficiently random and dissimilar to subsequent keys. The unpredictability of our proposals therefore, can be assessed by the behavior of the key sequences they generate with respect to these tests.

The keys generated by our proposed algorithms, and hence the framework, are not very similar to preceding/subsequent keys. We use the contentious word 'very' in the previous sentence to indicate that the similarity is quantified by the characters present in the keys, something limited by the set of available hexadecimal characters. With each hexadecimal character having a $\frac{1}{16}$ chance of being chosen for each character position in a generated key, we evaluated the similarity considering both full keys (of length 128 bits) and key blocks (four 32-bit blocks from the 128-bit key)[2]. Our expectation was for the average similarity index values to be in the vicinity of 0.30 for keys to be reasonably dissimilar. We obtained an average full-key SSI of 0.3549 for the main keys and 0.3632 for transfer keys (not including s_j), and an average key block SSI of 0.3251 for the main keys and 0.4356 for the transfer keys not (including s_j). Although the full keys (main and transfer keys) are seemingly dissimilar with SSI values in the vicinity of 0.3500, we observe that the average key block similarity is increased, mainly due to the reduced variation in the block-wise SSI values of K_T and configurations using K_T. Though this seems to reduce the block-wise dissimilarity between keys, the security of the system employing only the Butterfly1 proposal or any framework configuration using Butterfly1 would not be affected, since the data encryption key is derived using timestamp, which increases the overall uncertainty, reducing the impact of low dissimilarity between key blocks.

Our assessment of the randomness of key sequences (Section 7.2.5) further supports the unpredictability of most of our proposals. While Butterfly1, HiveSec1 and the framework configuration employing only these algorithms pass the NIST STS criterion for randomness in the tests considered, a somewhat surprising observation was with regard to GeM2. Although the major improvement in GeM2 was with regard to 'unlinking' the keys to an extent, the randomness tests still showed that the keys generated by this proposal might not be as random or unpredictable as expected.

[2]Block-wise key assessment was required to assess the effort that would be required for an adversary who has the knowledge of one block of the key to guess the remaining blocks, and therefore, the full key.

Revisiting Section 5.3, we observe that though GeM2 removes the linking between keys, the concept of generating keys requires that mutation patterns be applied to bits that are '0' in the parent keys, which means that GeM2 reduces the number of '0' bits in the new key generated. This affects GeM2's performance in the tests included in NIST STS because they primarily depend on key sequences having an equal proportion of '0' bits and '1' bits, with the tests verifying the possibility of a bit being chosen at random in the sequence. Though a majority (on average, 51.32%) of the key sequences generated by GeM2 fail the STS randomness tests (except the DFT test for periodic elements), the positive from this exercise is the ability of the framework (configurations $C4$, $C5$ and $C6$) to reduce its effect on the overall randomness of the keys generated by the framework. We observe that the presence of other algorithms improves the overall randomness of the keys generated by the framework configurations, and are able to reduce any effect of an algorithm that might be classified as failed by the STS assessment.

Considering the various results obtained in our study on the unpredictability of the keys generated by the framework (and its constituent algorithms), we can say that the behavior is satisfactory. This is because although our framework configurations present increased unpredictability due to the available algorithm choices, the behavior of the overall proposal is affected by GeM2. GeM2 includes mechanisms to update various parameters internally, adding to the overall uncertainty of the system, however, the sequences do not perform as expected in terms of proportionality of '1' bits and '0' bits. Therefore, although results obtained for Butterfly1, HiveSec1 and the framework employing only these algorithms support hypothesis $H3$, GeM2 and framework configurations that it is a part of, fail some of the tests. The positive deductions from this exercise, as discussed earlier, include the ability of the framework to reduce the effects of one algorithm that might perform poorly by combining the positives of other algorithms, and to introduce sufficient uncertainty in the generated key sequences (and implicitly state updates in the algorithms) to improve security of the system.

8.1.4 Attack detection

Securing algorithms by cryptographic mechanisms is only one part of securing communication between two entities. Although the keys generated by our framework maintain a level of unpredictability, they assist in encrypting data. Also important in securing communication between entities is the ability of an algorithm to provide the means to detect data modifications (due to errors or unauthorized modifications) and the means to mitigate any attacks. An implicit component in our proposals is the generation of authentication parameters (asv in GeM2, θ_i in Butterfly1 and message signatures in HiveSec1), which assist in entities verifying the messages sent and the entity sending the message. Our proposals facilitate non-repudiation by association, which is determined by their ability to mutually authenticate each other and being able to verify the integrity of the messages communicated, and supported by the fact that the initial parameters are never exchanged.

This, however, will not prevent unauthorized entities from attempting to disrupt the communication or attacking the system in other possible ways. The facilities to verify the authenticity of the entities and the message itself helps in determining attacks such as data modification, replay, de-synchronization, etc. as discussed in Section 7.3, while the presence of the BeeSwarm protocol as part of the framework (and HiveSec1) helps entities mitigate possible attacks in progress.

Our framework therefore, is not only able to detect attacks (true for all individual algorithm proposals as well), but is also able to attack an attacker as an attempt to mitigate any possible attack (such as replay or DoS) in progress, to the extent allowed by the resource constraints on the devices. Our assessment of security using the Scyther protocol analyser and the security analysis exercise, thus support our hypothesis ($H4$) that our proposals are able to detect attacks.

8.1.5 Resource utilization

One of the aspects necessary in resource-constrained applications such as RFID and WBAN systems is longevity. Longevity is mainly determined by how long a particular tag or sensor can function without the need for battery replacement or other maintenance. This is essential since their deployment in real-time scenarios and their adoptability by users is dependent on their ease of use and reduced need for frequent maintenance. Although our proposals are designed using simple logical operations (XOR, AND, OR, NOT) and minimal use of mathematical functions (such as addition and division), their deployment would also be contingent on them requiring less resources on the ASIC on which they will be deployed. Our exercise on evaluating the resources required (Section 7.5) substantiates our claims that the resources used by our framework and the individual constituent algorithms are not excessive, supporting their deployment in integrated circuits manufactured for a particular resource-constrained application environment. Although the results obtained in our assessment support our hypothesis ($H5$), it encourages us to further continue further optimization and evaluation of our proposals, to reduce the resources necessary even further.

8.2 Implications for Practice

In practice, the type of security solution deployed in an application environment depends on the type of data being communicated, whether the data are sensitive, and are impacted by the resources (storage, computation and cost) that are available. We have discussed two application scenarios and the possible applicability of our proposals in those contexts in Chapter 6. However, the security solution deployed in an application will entirely depend on a manufacturer's perception of the security needs in that application and on the impact on the manufacturing costs various available solutions will have. Another aspect that will impact this decision is longevity (as discussed in Chapter 2).

Our proposals are lightweight and include various mechanisms to maintain unpredictability of the overall system state, which increases security, and their reduced resource utilization (Section 7.5) is an added advantage. Using our solutions (framework and individual algorithms) for key generation and authentication in practical applications also promises accomplishing several security goals at minimal computational (and thus, resource) costs, which would mean that manufacturers will have additional resources for increasing the functionality of the overall application.

An aspect that needs to be considered when discussing practical use of our proposals is scalability; specifically, whether our algorithms can be used in scenarios that have more sensors or more RFID tags and how they would impact the cost in each case. Our proposals rely on uncertainty that is granted by continuously updating states and in case of the framework, also on algorithms being chosen at random. In a WBAN scenario, one possible example to consider is the use case discussed in Chapter 6. Practical WBAN deployments will have limited sensors that record a specific health parameter, somewhat similar to the application we considered. Our algorithms can therefore be deployed in practical WBAN sensors and will not cause much impact on their performance.

When we consider RFID applications, however, we need to consider the type of tags and type of data being communicated by the tags. Would our proposals help in keeping the same response by a tag different at different instances of time? Yes, as discussed previously. But, one would need to determine the appropriate type of framework or algorithms for use depending on the function of the tag, whether the data stored on the tag is sensitive and whether the application environment is expected to include mobile readers. One way of using the algorithms in a diverse environment, including both fixed and mobile readers and varying functions for the tag, is discussed in Chapter 6. One would still need to consider the number of tags that are present in the organization. The number of tags, the communication protocol and thus, the overhead on the server to be able to determine one tag out of the set of available tags would definitely be an aspect to consider prior to deployment. Theoretically, a server can be assumed to be a machine with limitless resources and the ability to perform computations to deduce the correctness of a tag's response. However, practically, one would need to consider how many such parallel computations the server can perform. The presence of state identifiers and tag identifiers (η_t in Butterfly1) help in potentially limiting the search space to identify and authenticate a particular tag. Another aspect that then impacts the function is the ability of the mobile or stationary reader to resolve its location and of the server to be able to send timestamp for the tag to perform computations. This task is simpler when the tag is held very close to the reader (as though through a 'tapping' action of the tag on the reader), but the tag identity resolution complexity increases with increase in the number of available tags in a particular 'zone' of an organization. Nevertheless, the search space is still reduced owing to the resolution of the reader's location and can be improved by storing the locations of all deployed tags.

The needs of an application both in terms of resources and security are entirely dependent on the environment in which it will be deployed. We can see that our proposals may be adapted for deployment in practical scenarios without many changes or the need to be concerned about the computational resources necessary; for, the resource-constrained entity needs to authenticate and communicate with only one server, and the server theoretically possesses the capacity to communicate with many resource-constrained entities.

8.3 Benefits and Challenges

Having discussed the performance of our proposals and their practical applicability, we now present some of the benefits and potential challenges that might necessitate future investigation.

First and foremost, the concept we have proposed employs three fundamentally different algorithms, with different design philosophies, working in the same ecosystem. This serves as a means to show how such diverse systems can impact the security of a system. Metamorphism makes the system change its structure and behavior based on a parameter dependent on the current context (e.g. time and entity ID). This dynamically changing structure and behavior of our framework is perhaps its biggest strength, which increases the overall uncertainty associated with the system.

Furthermore, our proposals are independent of PRNG/Hash/Encryption proposals. They are intended to be deployed as additional (lightweight) module(s), to make each of these or a combination into a secure one, able to accomplish multiple security goals, including but not limited to confidentiality, integrity verification, mutual authentication, as well as non-repudiation by association. This is in keeping with the "modular" design of our proposals, which ensures replacement of any of the modules without affecting the design of the overall system. This is the primary reason why we have a framework that dynamically chooses between algorithms — in order to make high security (through obscurity and unpredictable choices, of course) available to resource-constrained devices.

With algorithms being chosen at random and several different state identifiers in the entities, our framework helps in being able to assert that the message was sent by a legitimate entity belonging to a given organizational/application network. Successful communication is contingent on verification of the senders (and receivers), and therefore any intermediaries, because state updates occur with each transmitted message. If an intermediary is not authenticated at the server, either entity can detect its attempts to modify data or replay previous frames, and can take necessary action (including invoking the BeeSwarm protocol).

Our framework and the random choice of the constituent algorithms ensure that the security remains high and introduces uncertainty/unpredictability detrimental to the ability of a potential adversary retrieving any one or multiple keys. Although the individual algorithms may sometimes necessitate updates to seeds, the security is never compromised because of the constant length of the transmitted message (Chapter 6), the constantly changing internal states and the choice of one of the algorithms at random.

One of the aspects we have claimed in several places of this thesis is non-repudiation. When employing symmetric cryptosystems, entities are considered to not be able to guarantee non-repudiation. This necessitates the use of either asymmetric algorithms for select phases of communication, or the use of certificate/trusted authorities [35]. Despite this, we claim that our proposals are able to accomplish non-repudiation by association. This is because of the following reasons — (a) the shared initial secrets (between the server and resource-constrained entities) and any further key materials are never revealed, and (b) our framework uses the ID of the entity as one of the parameters to choose the algorithm to be used for a particular communication. This means that if an authenticated resource-constrained entity generates a specific parameter using our proposed approaches, then it could only mean that it was able to do so having gone through the exact state updates as the server. This further means that the resource-constrained entity will not be able to deny that it was the source of a verified/authenticated message. This security is also ensured because the choice of the algorithm is always dependent on the previously synchronized value of the timestamp, t_0, which could either be updated when Butterfly1 is chosen or HiveSec1 is chosen, making 'guessing' or 'cracking' the parameters challenging for an unauthorized entity.

Furthermore, our proposals facilitate on demand authentication or key generation, i.e. each entity can demand that the other reach a specific state to generate keys and authentication parameters. This is a feature that can be made more explicit when customizing the proposals for application in a specific domain.

Although there are several benefits of applying our proposals in resource-constrained wireless networks, there are some challenges that restrict their application and necessitate future investigation. Perhaps the first is the inability of GeM2 to be able to generate sequences that can be characterized as random by the tests in NIST STS. We have been able to identify one of the reasons for this behavior as the manner in which the parent keys are linked to the possible child keys generated from it, which reduces the number of ‘0’ bits in the new key sequence. This will necessitate future investigation into different mechanisms of generating keys.

Another possible challenge is in the practical deployment of the algorithm in an environment with many (in thousands or hundreds-of-thousands of) sensors or tags. This would necessitate proper documenting of the approximate locations of the tags and the ability of the readers to resolve their approximate location to achieve successful communication. This might not be a challenge in WBANs, since there are only a limited number of sensors that can be deployed on a human body, but, it might be a factor that might need to be considered in RFID systems. Although theoretically it is possible for a server to be able to resolve the identity of the tags using the techniques specified in our thesis, there could be other practical factors that might hinder performance.

An aspect that goes hand-in-hand with the practical deployment considerations is physical device compromise. As discussed in Section 5.6, should a particular deployment of the framework or any constituent algorithm be configured to perform as part of a cluster, physical device compromise would yield all security parameters, including internal state parameters, which could compromise the overall system security. Making such resource-constrained devices tamper-proof is therefore a very important contributor to data security, although it is beyond the purview of our work.

8.4 Summary

The framework and the constituent algorithms proposed in this thesis have the potential to be deployed in a variety of application scenarios, even when several aspects of the considered application have varying security requirements. The performance of our proposals in terms of key unpredictability and randomness is comparable to some of the proposals in literature that employ standard algorithms such as AES [7] and SHA-3 Hashing algorithm [9]. Furthermore, when compared to the suggested algorithms for the security suite [75, 78], the resources required by our proposals for deployment and operation are considerably low, making them ideal for deployment in resource-constrained wireless networks.

In this chapter, we have presented a general discussion about the behavior of our proposals under different conditions, identifying some of the strengths and research challenges, and noting some of the aspects that can be improved in the future. In the next chapter, we conclude our thesis, summarizing our work and presenting some potential considerations for future research.

Chapter 9

Conclusions and Future Work

9.1 Concluding Remarks

Motivated by the need for security proposals that consume less resources, while providing high security by means of increased unpredictability, our work proposed a metamorphic (or, reconfigurable) security framework. We draw inspiration for our work from the manner in which a chameleon changes its colour in response to the colour of its surroundings. Since resource-constrained devices would require circuits to be pre-defined at deploy time, our framework is based on using multiple (modular) security solutions that help in accomplishing reconfigurability in its operation. Our framework uses a synchronized value of the timestamp, an incrementing integer and the ID of the resource-constrained entity for choosing one of n_{Alg} available algorithms.

However, just a framework that chooses one of n_{Alg} algorithms might not be sufficient to guarantee high unpredictability. Therefore, we also propose three algorithms for security, with a main focus on dynamic key generation and authentication parameter computation. The three algorithms, namely, GeM2, Butterfly1 and HiveSec1, are inspired by concepts from biology and chaos theory. We draw inspiration from gene mutation and transfer in our design of GeM2, while Butterfly1 is inspired by the Butterfly Effect. HiveSec1 on the other hand, draws inspiration from the symmetry in the structure of beehives and functioning of bee swarms. Each algorithm is based on a fundamentally different concept and our framework is a means of integrating them to provide a mechanism for key generation (and thereby facilitating accomplishment of several security goals), with high unpredictability. The unpredictability is due to the varying internal states of each algorithm, and their ability to generate keys independently at the sender and receiver without having to exchange keys or other parameters. This dynamic and continuous key generation (rather than key exchange) makes for an application environment with increased security due to the uncertainty surrounding the choice of the next algorithm to be used for key generation and the implicit unpredictability of the parameters resulting thereof.

Our assessment largely supports our claims of high unpredictability and increased security, although our GeM2 proposal performs less than optimally when subjected to randomness tests in the NIST Statistical Test Suite. Nevertheless, the effect of any sub-optimal performance by one algorithm in the framework is mitigated by the presence of other algorithms in the framework, as observed in Chapter 7. Our results (presented in Chapter 7 and discussed in Chapter 8) support our claims of increased security through unpredictability, while requiring less resources for ASIC implementations of our algorithms and the framework. The benefits and strengths of our framework (and constituent algorithms) seem to outweigh the limitations and research challenges, while the challenges themselves become interesting opportunities warranting future investigation. Our framework and the constituent algorithms can be deployed in a variety of application domains that necessitate internal state updates and independent key generation at the entities, of which we presented two examples in Chapter 6 and discussed the applicability of our proposals in those scenarios.

While our framework and the constituent algorithms are designed to work as a cohesive unit or as standalone algorithms as proposed, there is still opportunity to extend, adapt and improve the proposals for achieving better security and higher applicability, as discussed next.

9.2 Future Work

Perhaps the first scope for improvement is in exploring the possibility for improving the randomness and unpredictability in the key sequences generated by GeM2. This would include first identifying the exact cause of failure in the sequences that fail to pass the randomness tests in NIST STS. This would be followed by exploring possible ways to improve GeM2.

Though reconfigurability and modularity are inherent concepts in our approach, it will also be worth exploring how modular and reconfigurable frameworks such as the one proposed in our thesis can be made, i.e. an ideal reconfigurable framework would include random choices between different internal modules of the framework. An example of this could be choosing key generation module from Butterfly1 while message signature from HiveSec1, and potentially even AES for encryption with the standards pushing for the technology to be able to support sophisticated encryption algorithms. Such high modularity and reconfigurability could of course result in significant modifications to the algorithm choice logic and to the manner in which the circuit could be implemented in hardware (on ASICs). This could mean that the hardware resources necessary to ensure such high modularity may be higher than what is required now, which is naturally an aspect to evaluate in the future.

Furthermore, another interesting aspect to explore is the application of this metamorphic framework for key generation with standard symmetric algorithms such as AES [35], to be able to generate the main key, which is then used to compute the keys for each round of encryption. This would be an interesting application to explore since it would mean that the entities would be able to generate main keys independently and possibly, on demand, leading to an increase in the overall unpredictability of the system. This would also lead to us possibly exploring the use of our framework (and constituent algorithms) in other, non-resource-constrained application domains as well.

Although we presented the BeeSwarm approach to mitigate perceived attacks by attacking the attacker, we will have to evaluate its practical use. The first aspect to explore in this would be to evaluate the amount of time for which or the number of frames that the entity can transmit. Despite this depending on the resources available for a specific application, it would be worth exploring an approximate (ideal) number of frames or the amount of time for which transmission of the meaningless frames has to occur, in order to mitigate an attack.

One final aspect to explore is drawing inspiration from other natural processes that could be capitalized in the framework, either in the algorithm choice logic or introducing an additional security module that could potentially result in more robustness and increased security, owing to an added element of uncertainty.

9.3 Final Thoughts

Our work has centred around achieving reconfigurability and unpredictability using simple functions, which results in reduced resource usage and increased security. Our approach of drawing inspirations from other domains such as biology/natural systems and chaos theory allows us to create such simple solutions. While our focus has been in developing our proposals for application in resource-constrained wireless networks, the design of each constituent algorithm and the framework are generic in nature, which means that they could be extended and adapted for use in other (non-resource-constrained) application environments as well.

In a world that is adopting more mobile technology solutions with each passing day and that is encouraging creation of more solutions that require these devices to talk among each other (in the Internet of Things paradigm), the need for distributed and independent (device-specific) security management will continue to increase. Our proposals are a step in addressing such security requirements, with each device employing them possessing the ability to manage their own security parameters and demand vectors to be generated by any other device wanting to communicate with it. We have proposed a new framework and three new standalone algorithms for accomplishing security through dynamic choices between algorithms in the framework, during PRNG seed updates, and during key/message signature generation in each algorithm, which improves the overall security of the system considerably due to the increased uncertainty. Our proposals are not only suited for application in resource-constrained devices/applications, but can be used in other non-resource-constrained applications as well. Furthermore, the implicit features of our proposals facilitate customization/adaptation for their use in several different application domains that may be useful for researchers and manufacturers of resource-constrained wireless devices.

Appendix A

Assessment Using NIST Statistical Test Suite: Detailed Results

In this Appendix, we provide detailed results obtained during our assessment of the proposed algorithm configurations and the comparative assessment with the work by Zhu and Khan [8], Dong et al. [9], and Liu and Kwak [7], using NIST Statistical Test Suite (STS) [130]. Tables A.1 – A.16 give the results for uniformity of P-values and the proportion of passing sequences. Summary numbers from these results have been presented in Table 7.6 and Table 7.17.

Table A.1: Uniformity of P-values and Proportion of Passing Sequences ($C1$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
5427	480	722	485	565	582	0	679	691	369	0.0000	*	4998/10000	*	Frequency
5419	539	484	595	567	313	668	469	514	432	0.0000	*	5027/10000	*	BlockFrequency
5319	542	634	520	515	536	353	575	595	411	0.0000	*	5140/10000	*	Runs
5386	534	525	559	510	618	394	550	475	449	0.0000	*	5360/10000	*	LongestRun
1995	665	0	2954	0	1669	0	0	2717	0	0.0000	*	9933/10000	*	FFT

Table A.2: Uniformity of P-values and Proportion of Passing Sequences ($C2(K_i)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
923	934	1474	973	1097	1229	0	1330	1358	682	0.0000	*	9895/10000		Frequency
949	1069	881	1185	1051	738	1316	815	1132	864	0.0000	*	9908/10000		BlockFrequency
1044	1054	1124	1006	944	1032	729	1101	1128	838	0.0000	*	9906/10000		Runs
942	1046	1021	1052	1047	1219	768	1078	905	922	0.0000	*	9896/10000		LongestRun
1166	1147	0	2147	0	2508	0	0	3032	0	0.0000	*	9862/10000	*	FFT

Table A.3: Uniformity of P-values and Proportion of Passing Sequences ($C2(K_T)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value	Proportion	Statistical Test
409	155	6697	471	390	485	0	570	520	303	0.000000 *	9906/10000	Frequency
433	303	245	394	409	324	6790	258	468	376	0.000000 *	9906/10000	BlockFrequency
237	395	596	320	400	392	6642	322	535	161	0.000000 *	10000/10000 *	Runs
325	6681	370	471	496	377	316	288	331	345	0.000000 *	9917/10000	LongestRun
539	213	0	751	0	780	0	0	7717	0	0.000000 *	10000/10000 *	FFT

Table A.4: Uniformity of P-values and Proportion of Passing Sequences ($C3(K_S)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value	Proportion	Statistical Test
907	915	1451	934	1050	1237	0	1366	1446	694	0.000000 *	9908/10000	Frequency
941	1019	930	1157	1080	745	1288	789	1111	940	0.000000 *	9921/10000	BlockFrequency
992	1020	1176	1022	919	1057	687	1094	1159	874	0.000000 *	9907/10000	Runs
910	1075	1036	1074	1017	1206	766	1096	895	925	0.000000 *	9918/10000	LongestRun
1228	1169	0	2149	0	2475	0	0	2979	0	0.000000 *	9857/10000 *	FFT

Table A.5: Uniformity of P-values and Proportion of Passing Sequences ($C3(K_O)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
907	978	1429	997	1068	1166	0	1341	1378	736	0.000000	*	9885/10000		Frequency
944	1080	922	1171	1017	698	1369	846	1062	891	0.000000	*	9930/10000	*	BlockFrequency
1013	1033	1065	1015	1030	1022	700	1110	1186	826	0.000000	*	9909/10000		Runs
964	1025	1004	1035	1039	1211	800	1112	911	899	0.000000	*	9904/10000		LongestRun
1220	1137	0	2114	0	2405	0	0	3124	0	0.000000	*	9804/10000	*	FFT

Table A.6: Uniformity of P-values and Proportion of Passing Sequences ($C4(K, K_i, K_S)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
2412	759	1244	768	941	996	0	1116	1172	592	0.000000	*	8288/10000	*	Frequency
2384	860	719	986	907	622	1154	727	896	745	0.000000	*	8320/10000	*	BlockFrequency
2406	867	998	863	797	842	602	927	971	727	0.000000	*	8342/10000	*	Runs
2415	810	843	911	832	1026	627	974	763	799	0.000000	*	8400/10000	*	LongestRun
1448	995	0	2462	0	2120	0	0	2975	0	0.000000	*	9856/10000	*	FFT

Table A.7: Uniformity of P-values and Proportion of Passing Sequences ($C4(K, K_T, K_O)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
2229	508	3013	634	711	767	0	819	858	461	0.000000	*	8299/10000	*	Frequency
2240	623	506	742	670	468	3009	553	634	555	0.000000	*	8318/10000	*	BlockFrequency
2178	656	769	620	594	675	2627	667	721	493	0.000000	*	8365/10000	*	Runs
2214	2784	625	696	641	758	498	612	586	586	0.000000	*	8416/10000	*	LongestRun
1288	675	0	1937	0	1670	0	0	4430	0	0.000000	*	9917/10000		FFT

Table A.8: Uniformity of P-values and Proportion of Passing Sequences ($C5(K, K_i)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
3092	696	1138	731	794	916	0	1021	1078	534	0.000000	*	7555/10000	*	Frequency
3134	794	673	924	815	505	1034	640	800	681	0.000000	*	7564/10000	*	BlockFrequency
3042	813	916	764	767	807	561	832	858	640	0.000000	*	7623/10000	*	Runs
3094	818	768	846	737	916	609	836	672	704	0.000000	*	7716/10000	*	LongestRun
1580	932	0	2556	0	2079	0	0	2853	0	0.000000	*	9885/10000		FFT

Table A.9: Uniformity of P-values and Proportion of Passing Sequences ($C5(K, K_T)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
2801	307	3867	465	491	514	0	606	622	327	0.000000	*	7564/10000	*	Frequency
2824	408	323	503	480	314	3937	353	471	387	0.000000	*	7572/10000	*	BlockFrequency
2664	482	583	399	437	465	3686	445	565	274	0.000000	*	7666/10000	*	Runs
2768	3762	438	506	480	490	358	403	386	409	0.000000	*	7724/10000	*	LongestRun
1218	453	0	1853	0	1217	0	0	5259	0	0.000000	*	9964/10000	*	FFT

Table A.10: Uniformity of P-values and Proportion of Passing Sequences ($C6(K, K_S)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
3183	663	1044	772	811	914	0	1021	1069	523	0.000000	*	7405/10000	*	Frequency
3250	783	661	922	799	484	1011	629	815	646	0.000000	*	7423/10000	*	BlockFrequency
3220	816	912	743	746	761	521	838	824	619	0.000000	*	7482/10000	*	Runs
3229	761	770	804	777	915	582	826	641	695	0.000000	*	7585/10000	*	LongestRun
1541	960	0	2584	0	2091	0	0	2824	0	0.000000	*	9876/10000		FFT

Table A.11: Uniformity of P-values and Proportion of Passing Sequences ($C6(K, K_O)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
3231	726	1030	711	779	932	0	1040	1040	511	0.000000	*	7406/10000	*	Frequency
3201	833	710	853	818	479	973	609	852	672	0.000000	*	7427/10000	*	BlockFrequency
3218	824	842	786	769	744	500	829	864	624	0.000000	*	7467/10000	*	Runs
3241	801	751	807	767	852	598	831	662	690	0.000000	*	7573/10000	*	LongestRun
1523	931	0	2579	0	2125	0	0	2842	0	0.000000	*	9889/10000		FFT

Table A.12: Uniformity of P-values and Proportion of Passing Sequences ($C7(K_i, K_S)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
945	926	1447	957	1076	1205	0	1298	1429	717	0.000000	*	9895/10000		Frequency
984	1035	877	1110	1040	788	1336	857	1092	881	0.000000	*	9907/10000		BlockFrequency
1077	992	1154	1040	941	1039	720	1105	1065	867	0.000000	*	9880/10000		Runs
929	1078	1026	1054	1022	1199	729	1099	899	965	0.000000	*	9905/10000		LongestRun
1254	1116	0	2165	0	2470	0	0	2995	0	0.000000	*	9837/10000	*	FFT

Table A.13: Uniformity of P-values and Proportion of Passing Sequences ($C7(K_T, K_O)$)

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
677	558	4185	696	737	811	0	950	928	458	0.000000	*	9901/10000		Frequency
707	652	580	729	739	528	4143	544	785	593	0.000000	*	9897/10000		BlockFrequency
661	701	768	700	636	724	3753	705	815	537	0.000000	*	9951/10000	*	Runs
647	3914	688	760	752	818	566	657	571	627	0.000000	*	9918/10000		LongestRun
839	697	0	1457	0	1588	0	0	5419	0	0.000000	*	9909/10000		FFT

Table A.14: Uniformity of P-values and Proportion of Passing Sequences — Keys generated using the proposal by Liu and Kwak [7]

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
966	873	1373	969	1117	1260	0	1325	1420	697	0.000000	*	9913/10000		Frequency
975	1025	900	1163	983	745	1353	847	1104	905	0.000000	*	9914/10000		BlockFrequency
1005	1046	1135	1039	950	1014	737	1062	1140	872	0.000000	*	9888/10000		Runs
947	994	1001	1061	1052	1199	748	1112	910	976	0.000000	*	9912/10000		LongestRun
1211	1202	0	2122	0	2366	0	0	3099	0	0.000000	*	9837/10000	*	FFT

Table A.15: Uniformity of P-values and Proportion of Passing Sequences — Keys generated using the proposal by Zhu and Khan [8]

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
893	893	1452	924	1093	1243	0	1346	1434	722	0.000000	*	9891/10000		Frequency
1018	1021	906	1111	1004	760	1267	895	1126	892	0.000000	*	9892/10000		BlockFrequency
1016	963	1144	1013	940	1047	716	1123	1168	870	0.000000	*	9892/10000		Runs
960	1014	965	1064	1011	1218	809	1119	883	957	0.000000	*	9886/10000		LongestRun
1282	1165	0	2087	0	2395	0	0	3071	0	0.000000	*	9832/10000	*	FFT

Table A.16: Uniformity of P-values and Proportion of Passing Sequences — Keys generated using the proposal by Dong et al. [9]

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value		Proportion		Statistical Test
943	941	1411	977	1127	1177	0	1345	1399	680	0.000000	*	9898/10000		Frequency
952	1084	882	1154	1075	699	1308	784	1098	964	0.000000	*	9926/10000		BlockFrequency
1027	1033	1112	985	949	1040	751	1063	1183	857	0.000000	*	9890/10000		Runs
967	1057	1012	1039	1057	1217	772	1051	886	942	0.000000	*	9894/10000		LongestRun
1222	1165	0	2099	0	2405	0	0	3109	0	0.000000	*	9833/10000	*	FFT

Appendix B

FPGA-based Simulation of our Proposals

In this Appendix, we present the outputs from simulation of our proposed algorithm configurations using Xilinx ISim simulator included in the Spartan-6 FPGA SP605 Embedded Kit [142]. In our simulations, the input and output were both serial, i.e. bit-wise, accumulated into a register prior to processing and prior to output, respectively. The signal, *enable*, was used to start a particular key generation cycle, culminating in the output of the encrypted bit sequence illustrated as the output in each illustration. In the simulation of the framework configurations, $C4 - C7$, the parameter *algorithm*[1:0] is a 2 bit number representing the algorithm choice. Table B.1 summarizes the algorithm and their corresponding algorithm number, represented by *algorithm*[1:0].

Table B.1: Algorithms and Algorithm Numbers (represented by *algorithm*[1:0]) used in the Simulation

Configuration	Algorithm number (Bit representation)	Algorithm
C4	0 (00) or 3 (11)	GeM2
	1 (01)	Butterfly1
	2 (10)	HiveSec1
C5	0 (00) or 3 (11)	GeM2
	1 (01) or 2 (10)	Butterfly1
C6	0 (00) or 3 (11)	GeM2
	1 (01) or 2 (10)	HiveSec1
C7	0 (00) or 1 (01)	Butterfly1
	2 (10) or 3 (11)	HiveSec1

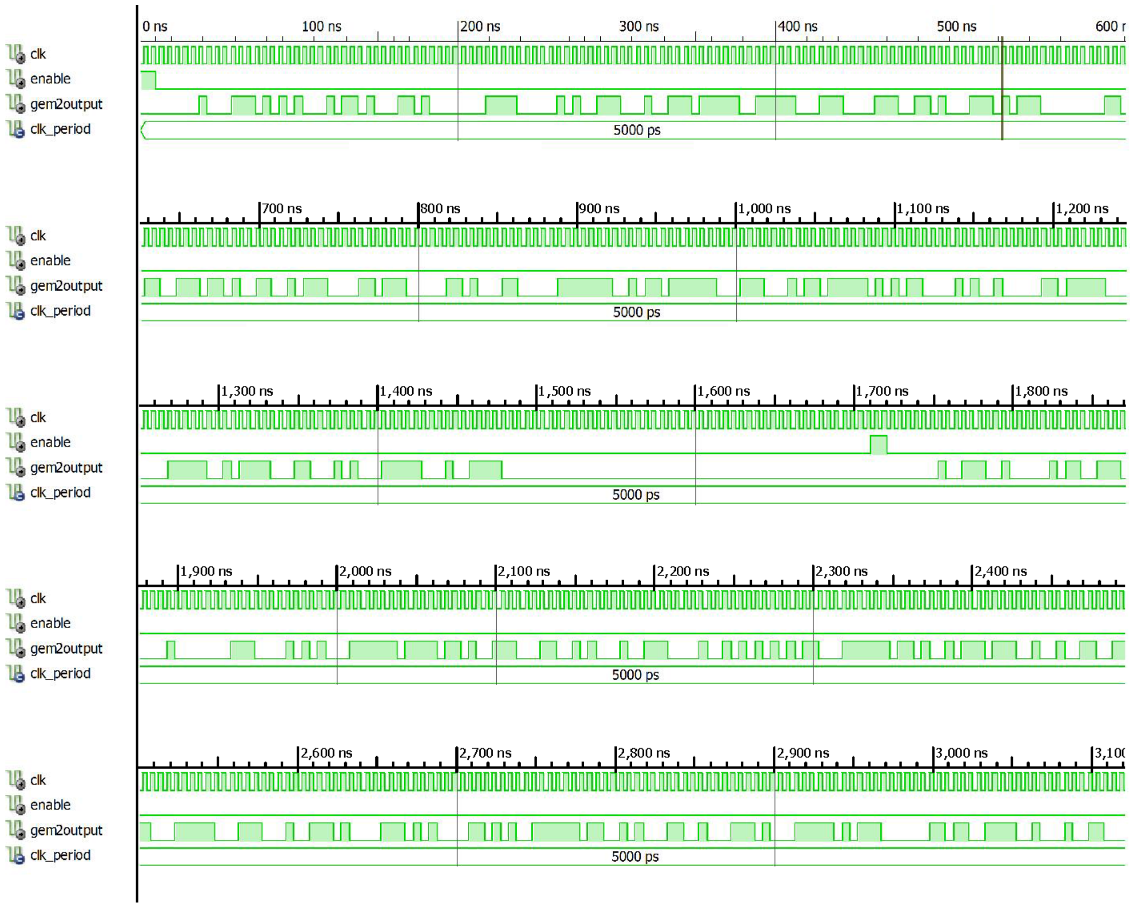


Figure B.1: Simulation of GeM2 using Xilinx ISim

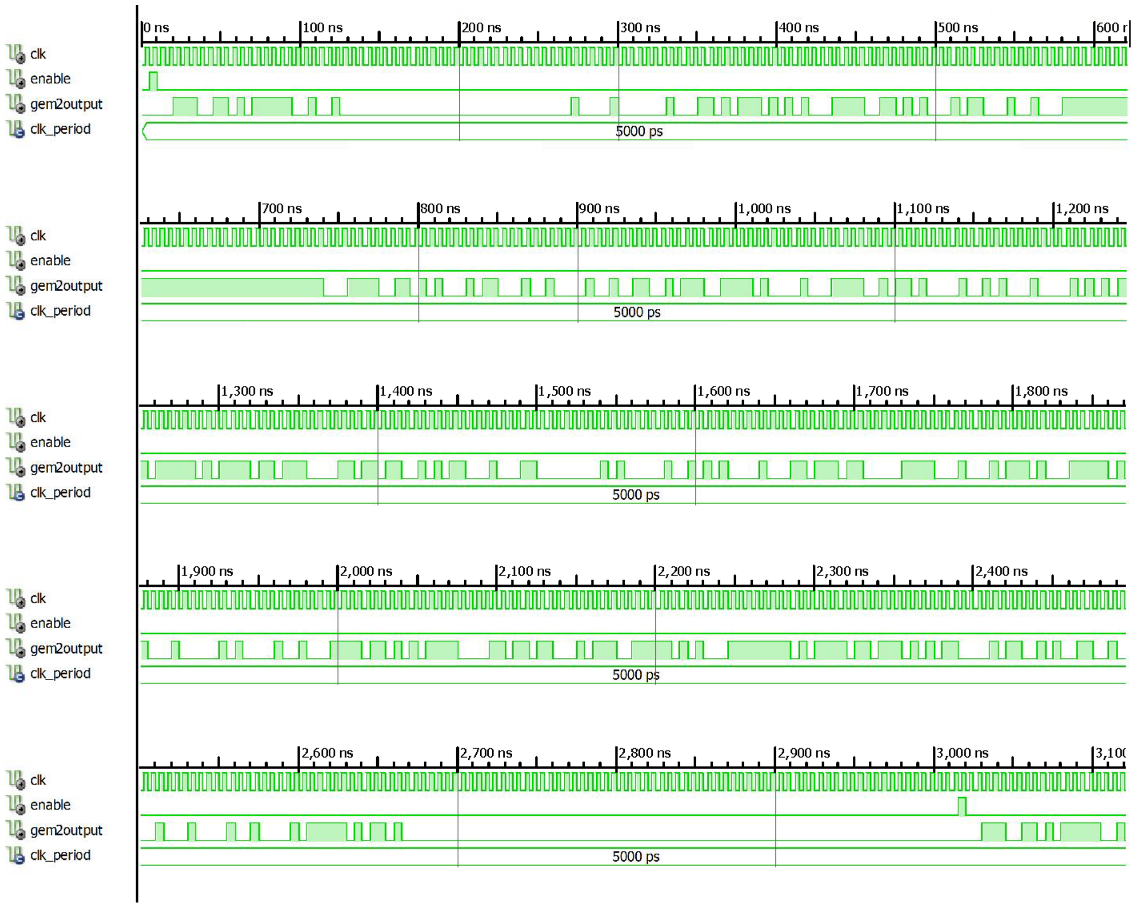


Figure B.2: Simulation of Butterfly1 using Xilinx ISim

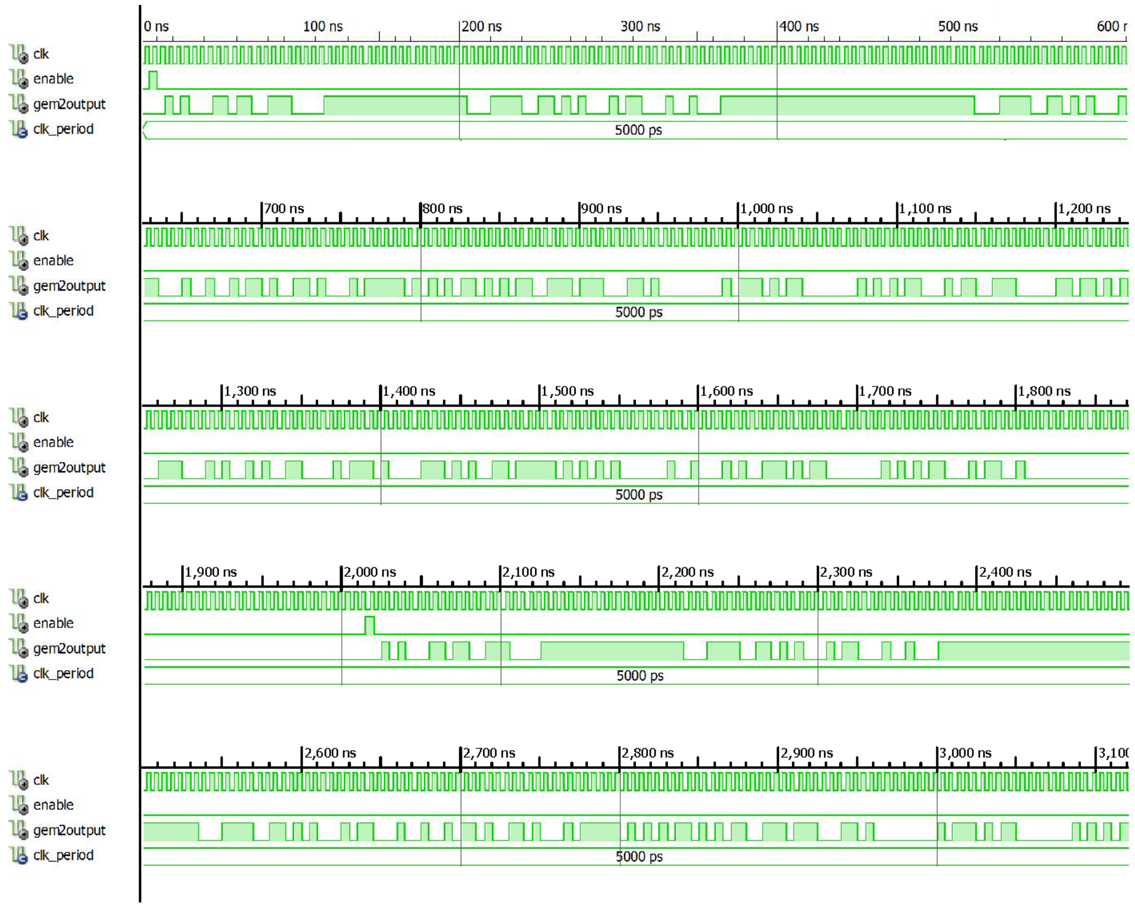


Figure B.3: Simulation of HiveSec1 using Xilinx ISim

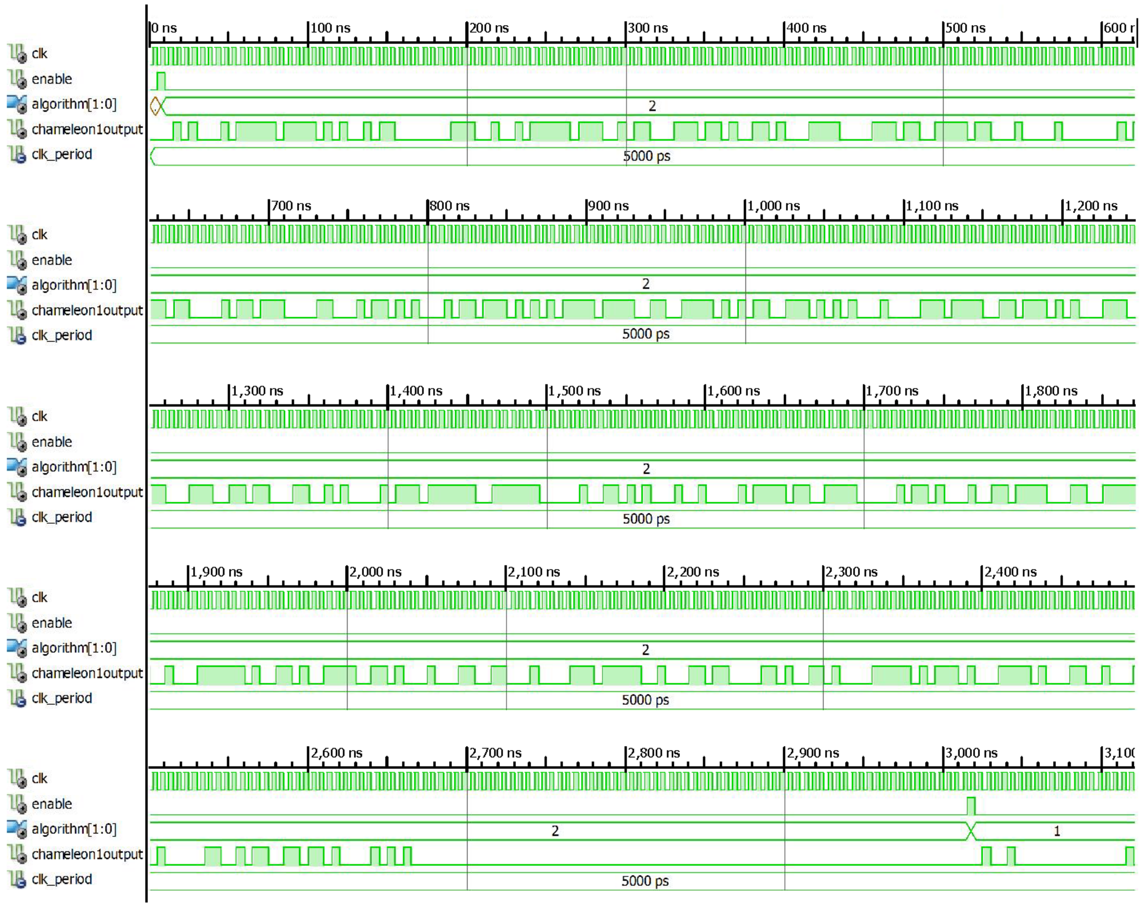


Figure B.4: Simulation of Framework (GeM2, Butterfly1, HiveSec1) using Xilinx ISim

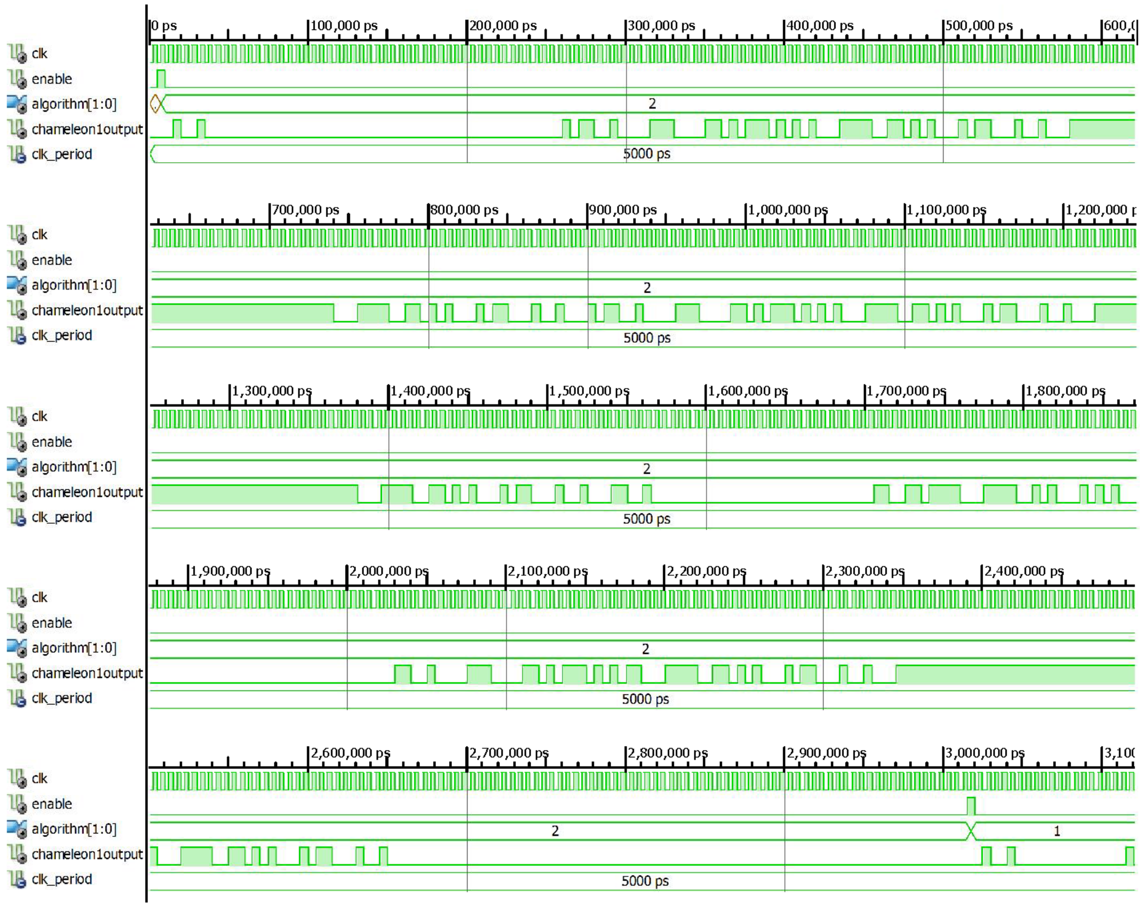


Figure B.5: Simulation of Framework (GeM2, Butterfly1) using Xilinx ISim

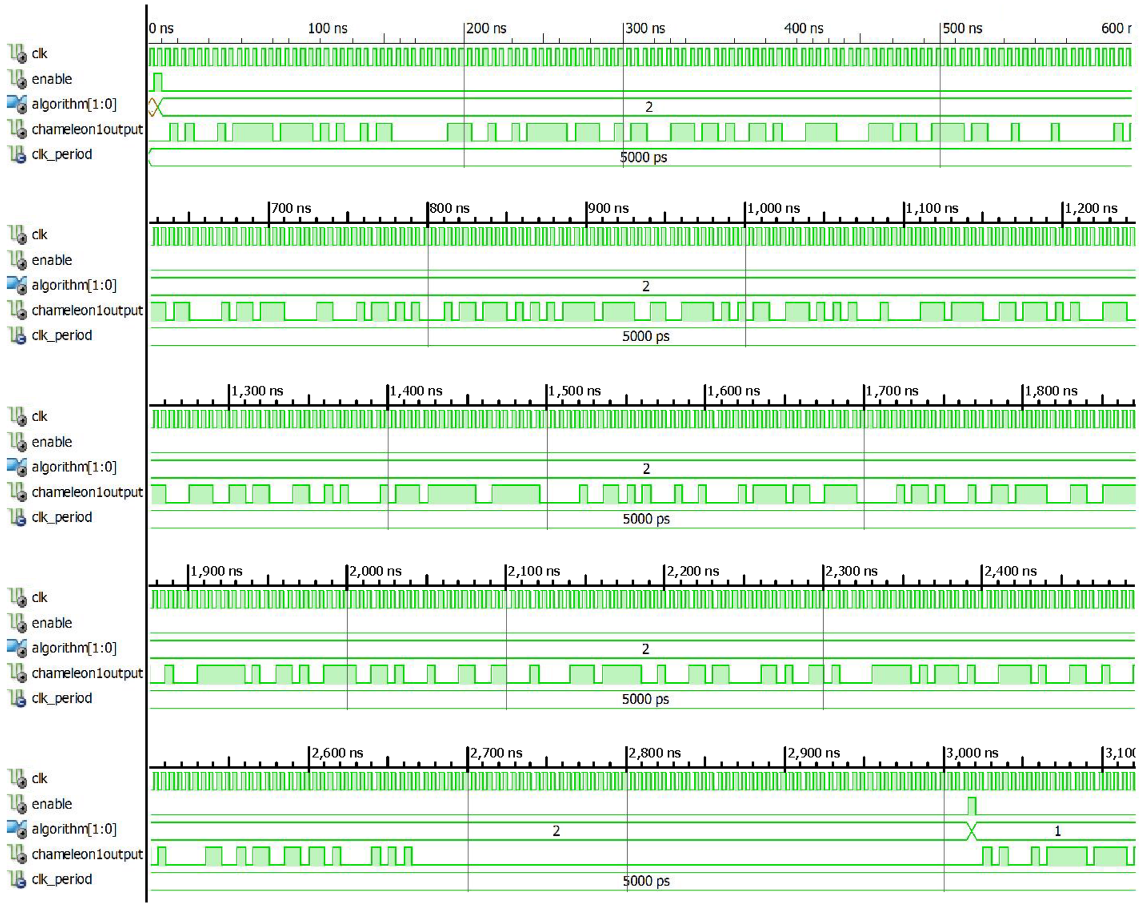


Figure B.6: Simulation of Framework (GeM2, HiveSec1) using Xilinx ISim

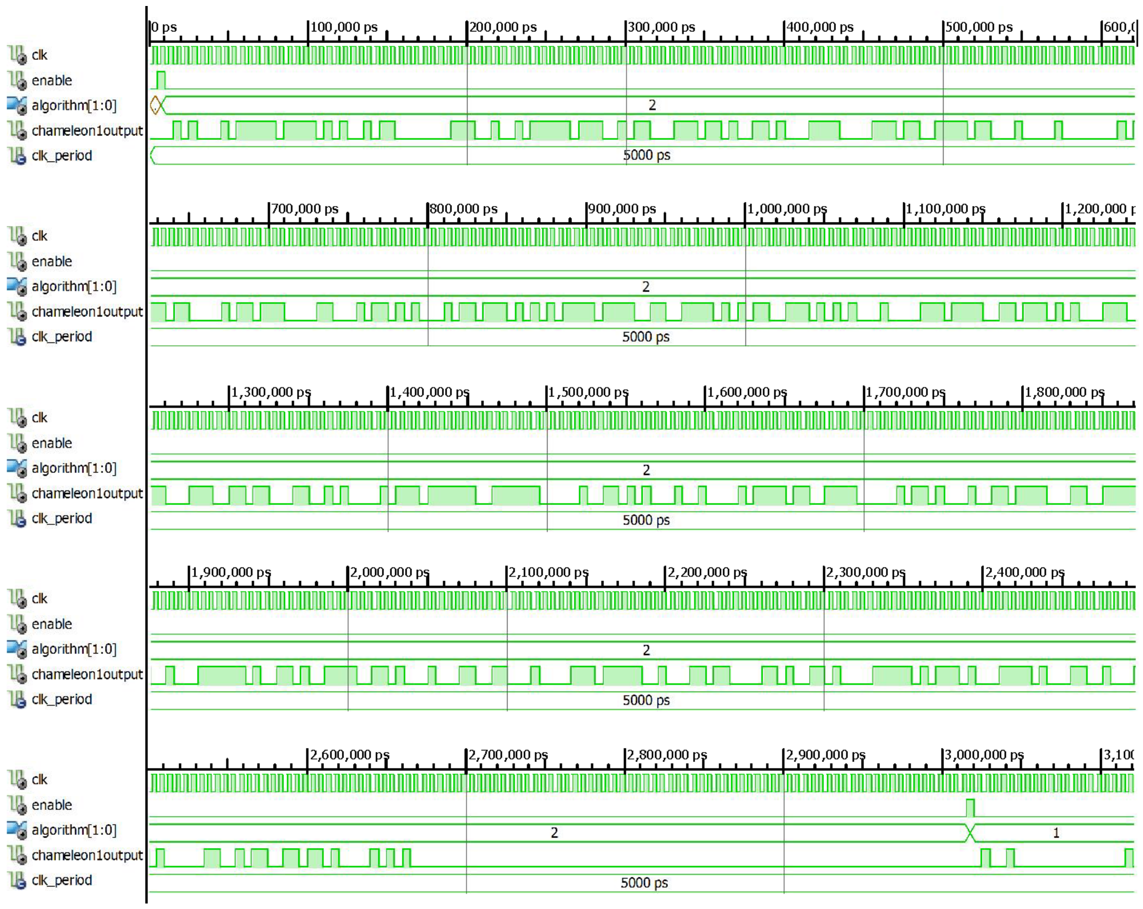


Figure B.7: Simulation of Framework (Butterfly1, HiveSec1) using Xilinx ISim

Appendix C

Logic Circuit Utilization Estimation

We estimated the logical circuit blocks that might be necessary for a conventional (non-optimal, non-FPGA) design of our proposals. We list the requirements for each configuration in Tables C.1 — C.7. Here, n is the size of the key in bits.

Table C.1: Logic Circuit Estimation: Configuration, $C1$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
AND	n bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	n bit $\times 1$
MUX (Multiplexer)	$2:1 \times 1$

Table C.2: Logic Circuit Estimation: Configuration, $C2$

Logic Circuit	Count
PRNG	n bit $\times 1$
Addition	$\frac{n}{2}$ bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	1 bit $\times 1$
MUX (Multiplexer)	$2:1 \times 1$

Table C.3: Logic Circuit Estimation: Configuration, $C3$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
Division	$\frac{n}{2}$ bit $\times 1$
AND	n bit $\times 1$
OR	n bit $\times 1$
XOR	n bit $\times 1$
MUX (Multiplexer)	$2:1\times 1$ $6:1\times 1$ $18:1\times 1$

Table C.4: Logic Circuit Estimation: Configuration, $C4$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
Division	$\frac{n}{2}$ bit $\times 1$
AND	n bit $\times 1$
OR	n bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	n bit $\times 1$
MUX (Multiplexer)	$2:1\times 1$ $6:1\times 1$ $18:1\times 1$

Table C.5: Logic Circuit Estimation: Configuration, $C5$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
AND	n bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	n bit $\times 1$
MUX (Multiplexer)	$2:1\times 1$

Table C.6: Logic Circuit Estimation: Configuration, $C6$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
Division	$\frac{n}{2}$ bit $\times 1$
AND	n bit $\times 1$
OR	n bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	n bit $\times 1$
MUX (Multiplexer)	$2:1 \times 1$ $6:1 \times 1$ $18:1 \times 1$

Table C.7: Logic Circuit Estimation: Configuration, $C7$

Logic Circuit	Count
PRNG	n bit $\times 1$
Hash	p bit $\times 1$
Addition	n bit $\times 1$
Division	$\frac{n}{2}$ bit $\times 1$
AND	n bit $\times 1$
OR	n bit $\times 1$
XOR	n bit $\times 1$
NOT (Invert)	1 bit $\times 1$
MUX (Multiplexer)	$2:1 \times 1$ $6:1 \times 1$ $18:1 \times 1$

Appendix D

Publications

D.1 Published

[PB1] R. V. Sampangi and S. Sampalli, “Tag-Server Mutual Authentication Scheme based on Gene Transfer and Genetic Mutation”, Proceedings of the 2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA 2012), July 2012, Ottawa, Canada [11].

[PB2] R. V. Sampangi and S. Sampalli, “RFID Mutual Authentication Protocols based on Gene Mutation and Transfer”, in the Special issue on RFID & Internet of Things of the Journal of Communications Software and Systems (JCOMSS), Vol. 9, No. 1, March 2013 [12].

[PB3] R. V. Sampangi and S. Sampalli, “HiveSign: Dynamic Message Signatures for Resource-Constrained Wireless Networks”, Proceedings of the 10th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet) 2014, September 2014, Montreal, Canada [14].

[PB4] R. V. Sampangi and S. Sampalli, “RFID Encryption Scheme Featuring Pseudorandom Numbers and Butterfly Seed Generation”, Proceedings of the 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM) 2014, September 2014, Split, Croatia [13].

D.2 Manuscripts Under Review

[PR1] R. V. Sampangi and S. Sampalli, “HiveSec: Security in Resource-Constrained Wireless Networks Inspired by Beehives and Bee Swarms”.

[PR2] R. V. Sampangi and S. Sampalli, “Butterfly Encryption Scheme for Resource-Constrained Wireless Networks”.

[PR3] R. V. Sampangi and S. Sampalli, “Metamorphic Framework for Key Management and Authentication in Resource-Constrained Wireless Networks”.

Appendix E

Copyright Permissions

Tag-Server Mutual Authentication Scheme based on Gene Transfer and Genetic Mutation [11].

RFID Mutual Authentication Protocols based on Gene Mutation and Transfer [12].

JCOMSS

Croatian Communications and Information Society,
Ruđera Boškovića 32
21000 Split
Croatia

Split, February 10, 2015.

PERMISSION STATEMENT

Following the request of Raghav Vemagal Sampangi, Croatian Communication and Information Society (CCIS), as a copyright holder, grants permission with no charges for author's materials that are published in the Journal of Communications Software and Systems (ISSN: 1845-6421), vol. 9, no. 1, March 2013, pp. 44-56.
Permission is granted for reuse in the Dalhousie University library, and Libraries and Archives Canada.

JCOMSS editorial chair
Dr. sc. Nikola Rožić

[redacted]

HiveSign: Dynamic Message Signatures for Resource-Constrained Wireless Networks [14].

ASSOCIATION FOR COMPUTING MACHINERY, INC. LICENSE TERMS AND CONDITIONS

Feb 09, 2015

This is a License Agreement between Raghav V. Sampangi ("You") and Association for Computing Machinery, Inc. ("Association for Computing Machinery, Inc.") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Association for Computing Machinery, Inc., and the payment terms and conditions.

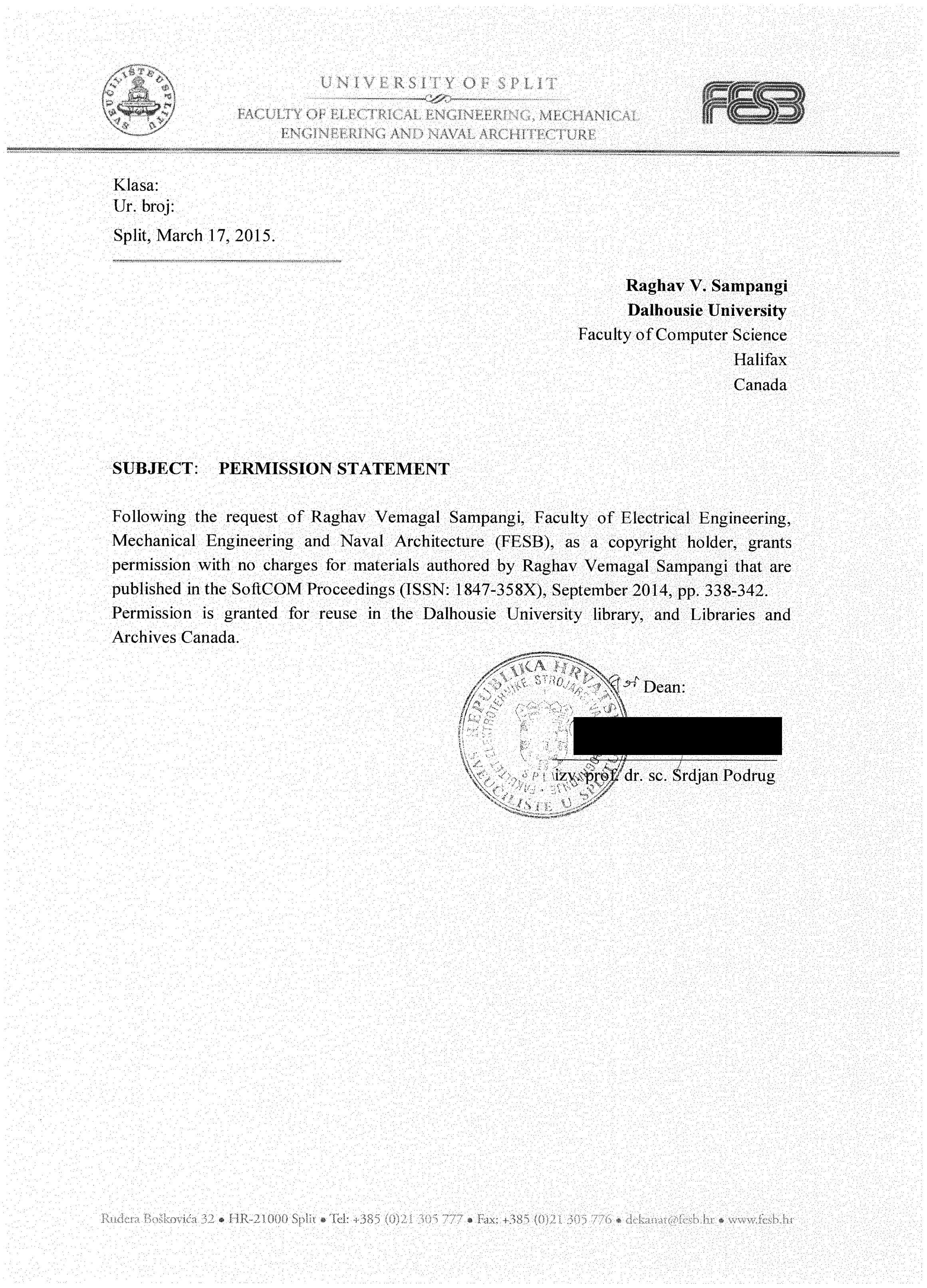
License Number	3565081504871
License date	Feb 09, 2015
Licensed content publisher	Association for Computing Machinery, Inc.
Licensed content publication	Proceedings of the 10th ACM symposium on QoS and security for wireless and mobile networks
Licensed content title	HiveSign: dynamic message signatures for resource-constrained wireless networks
Licensed content author	Raghav V. Sampangi, et al
Licensed content date	Sep 21, 2014
Type of Use	Thesis/Dissertation
Requestor type	Author of this ACM article
Is reuse in the author's own new work?	Yes
Format	Print and electronic
Portion	Full article
Will you be translating?	No
Order reference number	None
Title of your thesis/dissertation	Biomimetic Metamorphic Framework for Security in Resource-Constrained Wireless Networks
Expected completion date	May 2015
Estimated size (pages)	270
Billing Type	[redacted]
Credit card info	[redacted]
Credit card expiration	[redacted]
Total	8.00 USD

Terms and Conditions

Rightslink Terms and Conditions for ACM Material

1. The publisher of this copyrighted material is Association for Computing Machinery, Inc. (ACM). By clicking "accept" in connection with completing this licensing transaction, you

RFID Encryption Scheme Featuring Pseudorandom Numbers and Butterfly Seed Generation [13].



UNIVERSITY OF SPLIT

FACULTY OF ELECTRICAL ENGINEERING, MECHANICAL ENGINEERING AND NAVAL ARCHITECTURE

FESB

Klasa:
Ur. broj:
Split, March 17, 2015.

Raghav V. Sampangi
Dalhousie University
Faculty of Computer Science
Halifax
Canada

SUBJECT: PERMISSION STATEMENT

Following the request of Raghav Vemagal Sampangi, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture (FESB), as a copyright holder, grants permission with no charges for materials authored by Raghav Vemagal Sampangi that are published in the SoftCOM Proceedings (ISSN: 1847-358X), September 2014, pp. 338-342.
Permission is granted for reuse in the Dalhousie University library, and Libraries and Archives Canada.

Dean:

[redacted]

izv. prof. dr. sc. Srdjan Podrug

Ruđera Boškovića 32 • HR-21000 Split • Tel: +385 (0)21 305 777 • Fax: +385 (0)21 305 776 • dekanat@fesb.hr • www.fesb.hr

Bibliography

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. Prentice Hall, 2010.

[2] A. Mitrokotsa, M. R. Rieback, and A. S. Tanenbaum, "Classifying RFID attacks and defenses," *Information Systems Frontiers*, vol. 12, no. 5, pp. 491–505, 2010.

[3] S. Javadi and M. Razzaque, "Security and privacy in wireless body area networks for health care applications," in *Wireless Networks and Security*, ser. Signals and Communication Technology, S. Khan and A.-S. Khan Pathan, Eds. Springer Berlin Heidelberg, 2013, pp. 165–187. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36169-2_6

[4] A. Mitrokotsa, M. R. Rieback, and A. S. Tanenbaum, "Classifying RFID attacks and defenses," *Information Systems Frontiers*, vol. 12, no. 5, pp. 491–505, 2010.

[5] J. Chaudhry, U. Qidwai, R. Rittenhouse, and M. Lee, "Vulnerabilities and verification of cryptographic protocols and their future in wireless body area networks," in *2012 International Conference on Emerging Technologies (ICET)*, Oct 2012, pp. 1–5.

[6] *Spartan-6 FPGA Configurable Logic Block: User Guide*, Xilinx Inc., February 2010.

[7] J. Liu and K. S. Kwak, "Hybrid security mechanisms for wireless body area networks," in *2010 Second International Conference on Ubiquitous and Future Networks (ICUFN)*, June 2010, pp. 98–103.

[8] G. Zhu and G. Khan, "Symmetric key based RFID authentication protocol with a secure key-updating scheme," in *2013 26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2013, pp. 1–5.

[9] Q. Dong, J. Zhang, and L. Wei, "A SHA-3 based RFID mutual authentication protocol and its implementation," in *2013 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC)*, Aug 2013, pp. 1–5.

[10] R. V. Sampangi, S. Dey, S. R. Urs, and S. Sampalli, "A security suite for wireless body area networks," *International Journal of Network Security and its Applications (IJNSA)*, vol. 4, no. 1, pp. 97–116, 2012. [Online]. Available: http://arxiv.org/abs/1202.2171

[11] R. Sampangi and S. Sampalli, "Tag-server mutual authentication scheme based on gene transfer and genetic mutation," in *2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA)*, 2012, pp. 1–8.

[12] R. V. Sampangi and S. Sampalli, "RFID mutual authentication protocols based on gene mutation and transfer," *Journal of Communications Software and Systems*, vol. 9, no. 1, p. 44, Mar. 2013.

[13] R. V. Sampangi and S. Sampalli, "RFID encryption scheme featuring pseudo-random numbers and butterfly seed generation," in *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sept 2014, pp. 1–5.

[14] R. V. Sampangi and S. Sampalli, "HiveSign: Dynamic message signatures for resource-constrained wireless networks," in *Proceedings of the 10th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, ser. Q2SWinet '14. New York, NY, USA: ACM, 2014, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/2642687.2642699

[15] R. Want, "An introduction to RFID technology," *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25–33, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2006.2

[16] EPCglobal, "Tag Class Definitions," *EPCglobal*, p. 1, 2007.

[17] H.-Y. Chien, "Sasi: A new ultralightweight RFID authentication protocol providing strong authentication and strong integrity," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 337–340, 2007.

[18] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, T. Bessell, M. Rutten, and S. Jha, "Wireless sensor networks for battlefield surveillance," in *in Proc. of the Land Warfare Conference*, 2006.

[19] IEEE 802.15. IEEE 802.15: WIRELESS PERSONAL AREA NETWORKS (PANs). IEEE Standards Association. [Online]. Available: http://standards.ieee.org/about/get/802/802.15.html

[20] T. O'Donovan, J. O'Donoghue, C. Sreenan, D. Sammon, P. O'Reilly, and K. O'Connor, "A context aware wireless body area network (ban)," in *3rd International Conference on Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009*, April 2009, pp. 1–8.

[21] D. T. H. Lai, B. Santhiranayagam, R. K. Begg, and M. Palaniswami, Eds., *Healthcare Sensor Networks*. CRC Press, 2011.

[22] H. Karppanen, "Ischaemic heart disease," *Drugs*, vol. 28, no. 1, pp. 17–27, 1984. [Online]. Available: http://dx.doi.org/10.2165/00003495-198400281-00003

[23] F. Pipkin and E. Symonds, "Pregnancy-induced hypertension," in *Prostaglandins and their Inhibitors in Clinical Obstetrics and Gynaecology*, M. Bygdeman, G. Berger, and L. Keith, Eds. Springer Netherlands, 1986, pp. 337–366. [Online]. Available: http://dx.doi.org/10.1007/978-94-011-6734-5_16

[24] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. C. Leung, "Body area networks: A survey," *Mob. Netw. Appl.*, vol. 16, no. 2, pp. 171–193, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1007/s11036-010-0260-8

[25] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester, "A survey on wireless body area networks," *Wirel. Netw.*, vol. 17, no. 1, pp. 1–18, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1007/s11276-010-0252-4

[26] B. Defend, K. Fu, and A. Juels, "Cryptanalysis of two lightweight RFID authentication schemes," in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07.*, March 2007, pp. 211–216.

[27] J. Yu, G. Khan, and F. Yuan, "Xtea encryption based novel RFID security protocol," in *2011 24th Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2011, pp. 000 058–000 062.

[28] K. Osaka, T. Takagi, K. Yamazaki, and O. Takahashi, "An efficient and secure RFID security method with ownership transfer," in *2006 International Conference on Computational Intelligence and Security*, vol. 2, Nov 2006, pp. 1090–1095.

[29] Y.-Q. Gui, J. Zhang, and H. K. Choi, "An improved RFID security method with ownership transfer," in *2011 International Conference on ICT Convergence (ICTC)*, Sept 2011, pp. 594–596.

[30] I. Vajda and L. Buttyn, "Lightweight authentication protocols for low-cost RFID tags," in *In Second Workshop on Security in Ubiquitous Computing Ubicomp 2003*, 2003.

[31] W. Trappe and L. C. Washington, *Introduction to Cryptography with Coding Theory.* Pearson Prentice Hall, 2006.

[32] "EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID," *EPCglobal Specifications*, p. 152, 2013.

[33] "ISO/IEC 29167-1:2014 – Information technology – Automatic identification and data capture techniques – Part 1: Security services for RFID air interfaces," *International Standard*, p. 10, Aug 2014.

[34] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov 1976.

[35] C. Paar and J. Pelzl, *Understanding Cryptography.* Springer Berlin Heidelberg, 2010.

[36] J. Wilcox, *Solving the Enigma: History of the Cryptanalytic Bombe.* Center for Cryptologic History, National Security Agency, 2006.

[37] M. Li, W. Lou, and K. Ren, "Data security and privacy in wireless body area networks," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 51–58, February 2010.

[38] Y. Luo, Q. Chai, G. Gong, and X. Lai, "A lightweight stream cipher wg-7 for RFID encryption and authentication," in *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, Dec 2010, pp. 1–6.

[39] F. Miao, L. Jiang, Y. Li, and Y.-T. Zhang, "A novel biometrics based security solution for body sensor networks," in *2nd International Conference on Biomedical Engineering and Informatics, 2009. BMEI '09.*, Oct 2009, pp. 1–5.

[40] H. Ko and C. Ramos, "A study on the encryption algorithm for RFID tag (SEED : 8 rounds x 64 bits block)," in *International Conference on Convergence and Hybrid Information Technology, 2008. ICHIT '08.*, Aug 2008, pp. 672–677.

[41] P. Israsena, "Design and implementation of low power hardware encryption for low cost secure RFID using tea," in *2005 Fifth International Conference on Information, Communications and Signal Processing*, 2005, pp. 1402–1406.

[42] J. Yu, G. Khan, and F. Yuan, "XTEA encryption based novel RFID security protocol," in *2011 24th Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2011, pp. 000 058–000 062.

[43] Z. Jeddi, E. Amini, and M. Bayoumi, "A novel authenticated encryption algorithm for RFID systems," in *2013 Euromicro Conference on Digital System Design (DSD)*, Sept 2013, pp. 658–661.

[44] M. Liu and Y. Wang, "RFID system information security based on chaotic encryption," in *2011 Third International Conference on Multimedia Information Networking and Security (MINES)*, Nov 2011, pp. 499–502.

[45] T. Hongsongkiat and P. Chongstitvatana, "AES implementation for RFID tags: The hardware and software approaches," in *2014 International Computer Science and Engineering Conference (ICSEC)*, July 2014, pp. 118–123.

[46] Y. Lin, K. Kang, and Y. Shi, "Research on encryption model based on AES and ECC in RFID," in *2013 International Conference on Computer Sciences and Applications (CSA)*, Dec 2013, pp. 9–13.

[47] M.-J. Saarinen, "The bluejay ultra-lightweight hybrid cryptosystem," in *2012 IEEE Symposium on Security and Privacy Workshops (SPW)*, May 2012, pp. 27–32.

[48] C. Piao, Z. Fan, C. Yang, and X. Han, “Research on RFID security protocol based on grouped tags and re-encryption scheme,” in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security (WCNIS)*, 2010, pp. 568–572.

[49] C. Tan, H. Wang, S. Zhong, and Q. Li, “IBE-Lite: A lightweight identity-based cryptography for body sensor networks,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 926–932, Nov 2009.

[50] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, “Universal re-encryption for mixnets,” in *Topics in Cryptology CT-RSA 2004*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed. Springer Berlin Heidelberg, 2004, vol. 2964, pp. 163–178.

[51] K. Huey, W. Ismail, and M. Rahman, “Fingerprint-based mutual authentication RFID protocol,” in *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, Sept 2011, pp. 1–4.

[52] S. W. Jung and S. Jung, “HRP: A hmac-based RFID mutual authentication protocol using PUF,” in *2013 International Conference on Information Networking (ICOIN)*, Jan 2013, pp. 578–582.

[53] A. Juels, “Minimalist cryptography for low-cost RFID tags (extended abstract),” in *Security in Communication Networks*, ser. Lecture Notes in Computer Science, C. Blundo and S. Cimato, Eds., vol. 3352. Springer Berlin Heidelberg, 2004, pp. 149–164.

[54] M. Akgun and M. Caglayan, “PUF based scalable private RFID authentication,” in *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, Aug 2011, pp. 473–478.

[55] J. Liu, Z. Zhang, X. Chen, and K. S. Kwak, “Certificateless remote anonymous authentication schemes for wireless body area networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 332–342, Feb 2014.

[56] T. Kovacevic, T. Perkovic, and M. Cagalj, “Lira: A new key deployment scheme for wireless body area networks,” in *2013 21st International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sept 2013, pp. 1–6.

[57] C. C. Tan, H. Wang, S. Zhong, and Q. Li, “Body sensor network security: An identity-based cryptography approach,” in *Proceedings of the First ACM Conference on Wireless Network Security*, ser. WiSec ’08. New York, NY, USA: ACM, 2008, pp. 148–153. [Online]. Available: http://doi.acm.org/10.1145/1352533.1352557

[58] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on computing*, vol. 15, no. 2, pp. 364–383, 1986.

[59] P. Peris-Lopez, E. San Millan, J. van der Lubbe, and L. Entrena, "Cryptographically secure pseudo-random bit generator for RFID tags," in *2010 International Conference for Internet Technology and Secured Transactions (ICITST)*, 2010, pp. 1–6.

[60] R. Katti and S. Srinivasan, "Efficient hardware implementation of a new pseudo-random bit sequence generator," in *IEEE International Symposium on Circuits and Systems, 2009. ISCAS 2009.*, 2009, pp. 1393–1396.

[61] M. Rahman, "A novel scalable key management protocol for wireless sensor networks," April 2013. [Online]. Available: http://hdl.handle.net/10222/21683

[62] C.-L. Chen and Y.-Y. Deng, "Conformation of EPC class 1 generation 2 standards RFID system with mutual authentication and privacy protection," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 8, pp. 1284 – 1291, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197608001814

[63] M. Hakeem, K. Raahemifar, and G. Khan, "A novel key management protocol for RFID systems," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, July 2013, pp. 1107–1113.

[64] Q. Cai, Y. Zhan, and J. Yang, "The improvement of RFID authentication protocols based on r-rapse," *Journal of Networks*, vol. 9, no. 1, 2014.

[65] EPCglobal. EPCglobal. GS1. [Online]. Available: http://www.gs1.org/epcglobal

[66] "IEEE Standard for Local and metropolitan area networks - Part 15.6: Wireless Body Area Networks," *IEEE Std. 802.15.6-2012*, pp. 15 – 172, 2012.

[67] "Draft Protocol Specification for a 900 MHz Class 0 Radio Frequency Identification Tag," *EPCglobal Specifications*, pp. 1–49, 2003.

[68] SkyRFID Inc. (2014) RFID Gen 2 - What is it? - Smart RFID! SkyRFID Inc. [Online]. Available: http://skyrfid.com/RFID_Gen_2_What_is_it.php

[69] C. Bolan, "A review of the electronic product code standards for RFID technology," in *Proceedings of the Seventh International Network Conference (INC2008)*, 2008, pp. 171–178. [Online]. Available: http://www.cscan.org/openaccess/?id=179

[70] B. Fabian and O. Günther, "Security challenges of the epcglobal network," *Commun. ACM*, vol. 52, no. 7, pp. 121–125, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1538788.1538816

[71] D. Engels, Y. S. Kang, and J. Wang, "On security with the new gen2 RFID security framework," in *2013 IEEE International Conference on RFID (RFID)*, April 2013, pp. 144–151.

[72] "ISO/IEC 18031:2011 – Information technology – Security techniques – Random bit generation," *International Standard*, p. 142, Nov 2011.

[73] ISO (International Organization for Standardization). Standards catalogue. ISO (International Organization for Standardization). [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_tc_browse.htm?commid=45332&includesc=true&published=on&development=on

[74] "Advanced encryption standard (AES)," *Federal Information Processing Standards Publication 197* , Nov 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[75] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds. Springer Berlin Heidelberg, 2007, vol. 4727, pp. 450–466. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74735-2_31

[76] "ISO/IEC 29167-11:2014 – Information technology Automatic identification and data capture techniques Part 11: Crypto suite PRESENT-80 security services for air interface communications," *International Standard*, p. 10, Jul 2014.

[77] E. Barker, D. Johnson, and M. Smid, "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)," *NIST Special Publication 800-56A*, Mar 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf

[78] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128a: A new version of grain-128 with optional authentication," *Int. J. Wire. Mob. Comput.*, vol. 5, no. 1, pp. 48–59, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1504/IJWMC.2011.044106

[79] "Digital signature standard (dss)," *Federal Information Processing Standards Publication 186-4* , Jul 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[80] *Radio Frequency Identification: Security and Privacy Issues*, ser. Lecture Notes in Computer Science, 2014.

[81] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," 1979. [Online]. Available: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf

[82] P. L. Montgomery, “Modular multiplication without trial division,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr 1985. [Online]. Available: http://cseweb.ucsd.edu/classes/fa06/cse246/montgomery.pdf

[83] S. Ullah, M. Mohaisen, and M. A. Alnuem, “A review of ieee 802.15.6 mac, phy, and security specifications,” *International Journal of Distributed Sensor Networks*, vol. 2013, pp. 1–12, Nov 2013.

[84] N. Koblitz, “Elliptic curve cryptosystems,” *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan 1987.

[85] “Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality,” *NIST Special Publication 800-38C*, May 2004.

[86] B. A. Forouzan, *Data Communications and Networking.* McGraw-HilI Forouzan networking series, 2007.

[87] S. Saleem, S. Ullah, and K. S. Kwak, “Towards security issues and solutions in wireless body area networks,” in *2010 6th International Conference on Networked Computing (INC)*, May 2010, pp. 1–4.

[88] S. Saleem, S. Ullah, and H. S. Yoo, “On the security issues in wireless body area networks,” *International Journal of Digital Content Technology and its Applications*, vol. 3, no. 3, pp. 178–184, September 2009. [Online]. Available: http://www.aicit.org/JDCTA/ppl/22.pdf

[89] Atmel Corporation. (2009) Avr411: Secure rolling code algorithm for wireless link. [Online]. Available: http://www.atmel.com/images/atmel-2600-avr411-secure-rolling-code-algorithm-for-wireless-link_application-note.pdf

[90] Microchip Technology Inc. (2001) Keeloq code hopping encoder. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/21143b.pdf

[91] N. Döttling, D. Lazich, J. Müller-Quade, and A. S. de Almeida, “Vulnerabilities of wireless key exchange based on channel reciprocity,” in *Proceedings of the 11th International Conference on Information Security Applications*, ser. WISA’10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 206–220. [Online]. Available: http://dl.acm.org/citation.cfm?id=1949945.1949964

[92] A. Miyaji and M. S. Rahman, “Kimap: Key-insulated mutual authentication protocol for RFID,” *CoRR*, vol. abs/1209.5388, 2012.

[93] H. Martin, E. San Millan, L. Entrena, P. Lopez, and J. Castro, “Akari-x: A pseudorandom number generator for secure lightweight systems,” in *2011 IEEE 17th International On-Line Testing Symposium (IOLTS)*, 2011, pp. 228–233.

[94] J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti, “Multiple-polynomial lfsr based pseudorandom number generator for EPC gen2 RFID tags,” in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 3820–3825.

[95] D. Molnar, A. Soppera, and D. Wagner, “A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags,” in *Proceedings of the 12th International Conference on Selected Areas in Cryptography*, ser. SAC’05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 276–290.

[96] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The keccak sponge function family. [Online]. Available: http://keccak.noekeon.org/

[97] W. Choi, S. Kim, Y. Kim, Y. Park, and K. Ahn, “PUF-based encryption processor for the RFID systems,” in *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, 2010, pp. 2323–2328.

[98] L. Shi, M. Li, S. Yu, and J. Yuan, “BANA: Body area network authentication exploiting channel characteristics,” *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 1803–1816, September 2013.

[99] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1137/060651380

[100] Z. Zhang, H. Wang, A. Vasilakos, and H. Fang, “Ecg-cryptography and authentication in body area networks,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 6, pp. 1070–1078, Nov 2012.

[101] K. Venkatasubramanian, Venkatasubramanian, A. Banerjee, and S. Gupta, “EKG-based key agreement in body sensor networks,” in *IEEE INFOCOM Workshops 2008*, April 2008, pp. 1–6.

[102] M. Mana, M. Feham, and B. A. Bensaber, “Trust key management scheme for wireless body area network,” *International Journal of Network Security*, vol. 12, no. 2, pp. 75–83, 2011.

[103] M. Mana, M. Feham, and B. A. Bensaber, “SEKEBAN (secure and efficient key exchange for wireless body area network),” *International Journal of Advanced Science and Technology*, 2009.

[104] T. A. Pham, M. Hasan, and H. Yu, “A RFID mutual authentication protocol based on AES algorithm,” in *2012 UKACC International Conference on Control (CONTROL)*, Sept 2012, pp. 997–1002.

[105] M. Kuroda, Y. Tamura, R. Kohno, and O. Tochikubo, "Empirical evaluation of zero-admin authentication for vital sensors in body area networks," in *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2008. EMBS 2008.*, Aug 2008, pp. 2349–2352.

[106] W. Drira, E. Renault, and D. Zeghlache, "A hybrid authentication and key establishment scheme for WBAN," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, June 2012, pp. 78–83.

[107] J.-S. Cho, S.-S. Yeo, and S. K. Kim, "Securing against brute-force attack: A hash-based RFID mutual authentication protocol using a secret value," *Comput. Commun.*, vol. 34, no. 3, pp. 391–397, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2010.02.029

[108] E.-J. Yoon, "Improvement of the securing RFID systems conforming to EPC class 1 generation 2 standard," *Expert Systems with Applications*, vol. 39, no. 1, pp. 1589 – 1594, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417411010153

[109] S. Ahamed, E. Hoque, F. Rahman, F. Kawsar, and T. Nakajima, "Ya-srap: Yet another serverless RFID authentication protocol," in *2008 IET 4th International Conference on Intelligent Environments*, July 2008, pp. 1–8.

[110] J. Zhou, Y. Xu, and X. Li, "Reconfigurable and scalable security module of active RFID for security-sensitive applications," in *2010 The 2nd IEEE International Conference on Information Management and Engineering (ICIME)*, April 2010, pp. 135–140.

[111] C. Li, J. Zhou, Y. Jiang, C. Chen, Y. Xu, and Z. Luo, "A reconfigurable and scalable architecture for security coprocessor," in *2010 the 5th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, June 2010, pp. 1826–1831.

[112] A. K. Jones, R. Hoare, S. Dontharaju, S. Tung, R. Sprang, J. Fazekas, J. T. Cain, and M. H. Mickle, "An automated, fpga-based reconfigurable, low-power RFID tag," *Microprocess. Microsyst.*, vol. 31, no. 2, pp. 116–134, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2006.03.002

[113] C. H. Roth, Jr., *Digital Systems Design Using VHDL.* PWS Publishing Company, 1998.

[114] J. F. Vincent, O. A. Bogatyreva, N. R. Bogatyrev, A. Bowyer, and A.-K. Pahl, "Biomimetics: its practice and theory," *J. R. Soc. Interface*, vol. 3, no. 9, pp. 471–482, Mar. 2006.

[115] A. J. F. Griffiths, J. H. Miller, D. T. Suzuki, R. C. Lewontin, and W. M. Gelbart, *An Introduction to Genetic Analysis.* W. H. Freeman, 2000.

[116] G. S. L. Center. Tour of basic genetics. Learn.Genetics. [Online]. Available: http://learn.genetics.utah.edu/content/basics/

[117] Fibonacci number. Wolfram MathWorld. [Online]. Available: http://mathworld.wolfram.com/FibonacciNumber.html

[118] Linear recurrence equation. Wolfram MathWorld. [Online]. Available: http://mathworld.wolfram.com/LinearRecurrenceEquation.html

[119] D. Poulin. A rough guide to quantum chaos. Department of Physics and Institute for Quantum Computing, University of Waterloo. [Online]. Available: http://www.iqc.ca/publications/tutorials/chaos.pdf

[120] U. Maurer, "A universal statistical test for random bit generators," *Journal of Cryptology*, vol. 5, no. 2, pp. 89–105, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00193563

[121] R. C. Bishop, "On separating predictability and determinism," *Erkenntnis*, vol. 58, no. 2, pp. 169–188, 2003. [Online]. Available: http://www.igpp.de/english/tda/pdf/DeterminismPrediction.pdf

[122] S.-S. Shen, H.-R. Liao, S.-H. Lin, and J.-H. Chiu, "A novel stream cipher with hash function for the RFID device," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2011, pp. 532–536.

[123] F. Aguilar, H. Autrup, S. Barlow, L. Castle, R. Crebelli, W. Dekant, K.-H. Engel, N. Gontard, D. Gott, S. Grilli, R. Gurtler, J. C. Larsen, C. Leclercq, J.-C. Leblanc, F. X. Malcata, W. Mennes, M. R. Milana, I. Pratt, I. Rietjens, P. Tobback, and F. Toldra, "Scientific opinion of the panel on food additives, flavourings, processing aids and materials in contact with food (afc) on a request from the commission on the safety in use of beeswax," *The EFSA Journal*, pp. 1–28, 2007.

[124] M. J. Sommeijer, "Beekeeping with stingless bees: a new type of hive," *Bee World*, vol. 80, no. 2, pp. 70–79, 1999.

[125] R. V. Sampangi, "Enhancing security and reliability in wireless body area networks for remote health monitoring," M. Tech. Project Report, International School of Information Management, University of Mysore, 2011.

[126] Java. Oracle Corporation. [Online]. Available: http://www.java.com/en/

[127] Eclipse. The Eclipse Foundation. [Online]. Available: https://eclipse.org/

[128] SHA1-Online. SHA1 usage implementation in java: sha1 of a text string and file's sha1 checksum verification. [Online]. Available: http://www.sha1-online.com/sha1-java/

[129] T. Sorensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons," *Biologiske Skrifter Kongelige Danske Videnskabernes Selskab*, vol. 5, no. 4, pp. 1–34, 1957.

[130] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, National Institute of Standards and Technology (NIST).

[131] C. Cremers. The Scyther tool. Department of Computer Science, University of Oxford. [Online]. Available: http://users.ox.ac.uk/~coml0529/scyther/

[132] G. Lowe, "A hierarchy of authentication specifications," in *Proceedings., 10th Computer Security Foundations Workshop, 1997.*, 1997, pp. 31–43.

[133] G. Hollestelle, "Systematic analysis of attacks on security protocols," Master's thesis, Technische Universiteit Eindhoven, 2005. [Online]. Available: http://alexandria.tue.nl/extra1/afstversl/wsk-i/hollestelle2005.pdf

[134] C. Cremers, *Scyther User Manual*, Department of Computer Science, University of Oxford. [Online]. Available: https://github.com/cascremers/scyther/blob/master/gui/scyther-manual.pdf

[135] Thomas Mueller. An implementation of the XTEA block cipher algorithm. H2 Group. [Online]. Available: https://code.google.com/p/h2database/source/browse/trunk/h2/src/main/org/h2/security/XTEA.java

[136] E. Barker, D. Johnson, and M. Smid, "SHA-3 standard: Permutation-based hash and extendable-output functions," *Federal Information Processing Standards Publication*, May 2014. [Online]. Available: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf

[137] Osman Kocak. The Keccak digest algorithm. [Online]. Available: https://github.com/kocakosm/pitaya/blob/master/src/org/kocakosm/pitaya/security/Keccak.java

[138] Class Cipher. Oracle Corporation. [Online]. Available: http://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html

[139] Java Cryptography Architecture (JCA) Reference Guide. Oracle Corporation. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html

[140] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992.

[141] (2014) Field programmable gate array FPGA. Xilinx Inc. [Online]. Available: http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[142] *Getting Started with the Spartan-6 FPGA SP605 Embedded Kit*, Xilinx Inc., June 2010.

[143] J. Bhasker, *A VHDL Primer.* P T R Prentice Hall, 1999.

[144] J. Melia-Segui, J. Garcia-Alfaro, and J. Herrera-Joancomarti, "J3gen: A prng for low-cost passive RFID," *Sensors*, vol. 13, no. 3, pp. 3816–3830, 2013.

[145] J. Rainier. SHA1 Sequential Implementation. [Online]. Available: https://github.com/JarrettR/FPGA-Cryptoparty/blob/master/decrypt%20-%20sequential/main-s.vhd

[146] (2013) Xst user guide for virtex-6, spartan-6, and 7 series devices. Xilinx Inc. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst_v6s6.pdf

[147] *7 Series FPGA Configurable Logic Block: User Guide*, Xilinx Inc., November 2014.

[148] (2014) All programmable low-end portfolio product selection guide. Xilinx Inc. [Online]. Available: http://www.xilinx.com/publications/prod_mktg/low-end-portfolio-product-selection-guide.pdf

[149] P. Peris-lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador, and A. Ribagorda, "RFID systems: A survey on security threats and proposed solutions," in *PWC 2006.* Springer, 2006, pp. 159–170.

[150] A. Juels and S. Weis, "Authenticating pervasive devices with human protocols," in *Advances in Cryptology CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed. Springer Berlin Heidelberg, 2005, vol. 3621, pp. 293–308. [Online]. Available: http://dx.doi.org/10.1007/11535218_18

[151] M. O. (nee McLoone), "Low-cost SHA-1 hash function architecture for RFID tags," in *Proceedings of the Workshop on RFID Security 2008 (RFIDsec 08*, 2008, pp. 1–11.

[152] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices security for 1000 gate equivalents," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, G. Grimaud and F.-X. Standaert, Eds. Springer Berlin Heidelberg, 2008, vol. 5189, pp. 89–103. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85893-5_7

[153] I. Kuon and J. Rose, *Quantifying and Exploring the Gap Between FPGAs and ASICs.* Springer Science+Business Media, 2009.

[154] D. Brand and C. Visweswariah, “Inaccuracies in gate-level power estimation,” 1996. [Online]. Available: http://vlsicad.ucsd.edu/courses/ece260b-w05/pdf/RC20520.pdf

[155] J. Elias and A. Mehaoua, “Energy-aware topology design for wireless body area networks,” in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 3409–3410.

[156] M. Salajegheh, S. Clark, B. Ransford, K. Fu, and A. Juels, “CCCP: Secure remote storage for computational RFIDs,” in *USENIX Security Symposium*, 2009, pp. 215–230.

[157] International Bureau of Weights and Measures (BIPM), *The International System of Units (SI).* International Bureau of Weights and Measures (BIPM), 2006.

[158] *Impinj® Monza® X-8K Dura Datasheet*, Impinj, Inc., 2014.

[159] J. Y. Khan and M. R. Yuce, *Wireless Body Area Network (WBAN) for Medical Applications*, ser. New Developments in Biomedical Engineering, D. Campolo, Ed. InTech, 2010. [Online]. Available: http://www.intechopen.com/books/new-developments-in-biomedical-engineering/wireless-bodyarea-network-wban-for-medical-applications

[160] J. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, “Ieee 802.15.4: a developing standard for low-power low-cost wireless personal area networks,” *Network, IEEE*, vol. 15, no. 5, pp. 12–19, Sept 2001.