REPORT

ON

"LIGHTWEIGHT ENCRYPTION SCHEME FOR RFID"

MASTER OF TECHNOLOGY
(Computer Science & Engineering)

By
Swarna Chaudhary
(2k12/CSE/22)

Under the Supervision of

Mr. Manoj Kumar
(Project Coordinator (CSE))

DELHI TECHNOLOGICAL UNIVERSITY
(formerly Delhi college of engineering), Delhi

CERTIFICATE

This is to certify that the project entitled by "**Lightweight Encryption Scheme for RFID"** submitted in the partial fulfilment , for the award of Degree of **Master of Technology (Computer Science & Engineering)** of **DELHI TECHNOLOGICAL UNIVERSITY** (formerly Delhi college of engineering), Delhi by **Swarna Chaudhary (2K12/CSE/22)** is carried out by her under my supervision. The matter embodied in this project has not been submitted earlier for award of any degree or diploma in any university/institution to the best of my knowledge & belief.

Date:

(Mr. Manoj Kumar)
Project Coordinator (CSE)

CANDIDATE'S DECLARATION

I hereby declare that this submission is my own work and that, to the best of my Knowledge and belief, it contains no material previously published or written by another person for material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or the other institute of higher learning, except where due acknowledgment has been made in the text.

Signature:

Name: Swarna Chaudhary
Roll No: 2K12/CSE/22
Date:

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to **Mr. Manoj Kumar, Computer Science & Engineering** for his generous guidance, help and useful suggestions, continuous encouragement and supervision throughout the course of present work.

Swarna Chaudhary
2K12/CSE/22

ABSTRACT

"LIGHTWEIGHT ENCRYPTION SCHEME FOR RFID"

Due to the tight cost and constrained resources of high volume consumer devices such as RFID tags, smart cards and wireless sensor nodes, it is desirable to employ lightweight and specialized cryptographic primitives for many security applications. It is widely believed that there is a trade-off between speed and security in cryptosystem design. No existing encryption algorithms are both fast enough for high-speed operation and sufficiently secure to withstand powerful cryptanalysis. Our objective is to design the ciphers that are suitable for security in Radio Frequency Identification (RFID) and other security applications with demanding area restrictions. In this work, we propose and analyze a generic construction of high-speed encryption schemes. Our solution is based on the fact that there exist secure lightweight hybrid cipher i.e. Hummingbird, which is the mainly designed for resource constrained environment like RFID technology, wireless sensors and other smart devices. The consumption of power is less and encryption speed is faster in Hummingbird. It is resistant to most of the cryptanalytic attacks common to block ciphers and stream ciphers such as linear and differential cryptanalysis.

But to make the scheme more secure than the existing ones, we combine a secure hybrid cipher with a super-fast stream cipher such that the resulting encryption scheme is more secure than the existing ones and is faster too. In this encryption scheme, we are generating the keys using hybrid cipher Hummingbird, in such a way that the output of Hummingbird is used as a key for superfast stream cipher scheme to encrypt the text.

In this thesis work, all architectures proposed, are simulated by means of MentorGraphics ModelSim tool in order to verify the correct designs and functionality. For the development (synthesis) Xilinx simulator tool is used. Specifically, a loop architecture has been used, where one basic round is used iteratively. The basic performance metrics are the area, power consumption associated with the implementation resulting throughput of each cipher. We have compared the results with the existing schemes. Hummingbird, AES and the variants of Hummingbird have been employed in the comparison. Furthermore, we also presented the software implementation of the scheme and compare the performance of the proposed scheme on both the platforms i.e. Hardware as well as the software platforms.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Motivation

Today's technologies are advancing every second of the day. The world is getting more and more compact with the new inventions and discoveries. The first era of computing in 1960s saw large mainframe computers which were common in large industries and with the US military and space program. These were large, expensive, error-prone, and very hard to use machines. In this Mainframe Era, many persons interact with one computer. With the Technological Advancement, the Mainframes got converted into desktops in the second era, where one person interacts with one computer and then the third era of computing is known as "Ubiquitous Computing", where one person can use many computers at a time.

Increasingly, everyday items are enhanced to pervasive devices by embedding computing capabilities, power and their interconnection leads to the concept of ***Ubiquitous computing*** which is widely believed to be the next paradigm in the Information technology.

Ubiquitous computing is a post-desktop model of human-computer interaction in which the information processing has been thoroughly integrated into everyday objects and activities.

The idea is that almost any device from clothing to tools to appliances to human body to cars to homes, can be embedded with chips to connect the device to the infinite network of other devices. People are demanding more and more from the technologies and the creator of these technologies never failed to please their necessities.

Low cost devices such as RFIDs, Sensor Network Nodes, and smartcards are crucial for building the next generation pervasive and ubiquitous networks. Such networks are envisioned to be extended in functionality and reach with wireless sensor nodes, RFIDs, and smartcards. Lightweight devices like mobile, pagers, ipad, Radio Frequency Identification

(RFID) are becoming popular day by day. They are becoming essential part of life. These devices are hardly ever switched off, if it's even possible to do so, and they are very closely connected to a specific person. People typically carry around multiple RFID tags (e.g. access cards, passport, product tags) .In general we can describe an RFID tag as a small, wireless device without any user interface.

Despite the wide range of applications of pervasive computing devices, they feature two common functionalities:

- the devices store a moderate amount of sensitive information (e.g. label of the node/product ID/name of person, or key information) and
- the devices transmit data through wireless networks.

It is required that sensitive information is stored and communicated securely over the network. Both operations require security features, such as confidentiality, authentication, integrity checks and privacy. However, smart devices have several limitations with regards to memory, resources and computation power, hindering the opportunity to apply well established standard cryptographic algorithms and techniques for authentication, confidentiality and data security issues which were originally designed to secure high-end systems with abundant power. Furthermore, the sharp increase in the number, diversity and strength of physical attacks which directly target the implementation, may have devastating consequences in a network setting, by creating a single point of failure.

Now a days, most of the transactions are being done using these handheld devices. Hence, security in wireless sensor networks is currently almost exclusively achieved through Lightweight Cryptography.

In this work, we have proposed a lightweight Encryption approach which is ***"As light as a feather, and as hard as dragon-scales"***. A strong focus is put on lightweight Encryption scheme that require as few area (measured in Gate Equivalents (GE)) as possible, takes less power and provides desired level of security.

A variety of implementation results both in software and hardware—using different design strategies and different platforms is presented.

1.2 Research Goals

In **t**his work, we tried to focus on the following challenges:-

- Design, analysis and implementation of lightweight Encryption Scheme which is as fast as stream cipher and as secure as Block cipher.
- Cryptographic hardware simulation using VHDL on Xilinx 6.1i for constrained domains
- Design and analysis of fast and compact cryptographic algorithms
- Wireless network security for low-resource devices
- Low-power crypto architectures
- Comparison of Software Based Encryption with Hardware Simulation in terms of performance.

1.3 Radio-frequency identification (RFID)

Radio-frequency identification (**RFID**) is the wireless non-contact use of radio-frequency electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects. The tags contain electronically stored information. Some tags are powered by and read at short ranges (a few meters) via magnetic fields (electromagnetic induction), and then act as a passive transponder to emit microwaves or UHF radio waves (i.e., electromagnetic radiation at high frequencies). Others use a local power source such as a battery, and may operate at hundreds of meters. Unlike a bar code, the tag does not necessarily need to be within line of sight of the reader, and may be embedded in the tracked object.

It is defined as an ADC (Automated Data Collection) technology that

- uses radio frequency waves to transfer data between a reader and a movable item to identify, categorize, and track the tagged object.
- Is fast and does not require physical sight or contact between reader/scanner and the tagged item.

- Performs the operation using low cost components.
- Attempts to provide unique identification and backend integration that allows for wide range of applications.

RFID tags are used in many industries [1]. An RFID tag attached to an automobile during production can be used to track its progress through the assembly line. Pharmaceuticals can be tracked through warehouses. Livestock and pets may have tags injected, allowing positive identification of the animal. On off-shore oil and gas platforms, RFID tags are worn by personnel as a safety measure, allowing them to be located 24 hours a day and to be quickly found in emergencies.

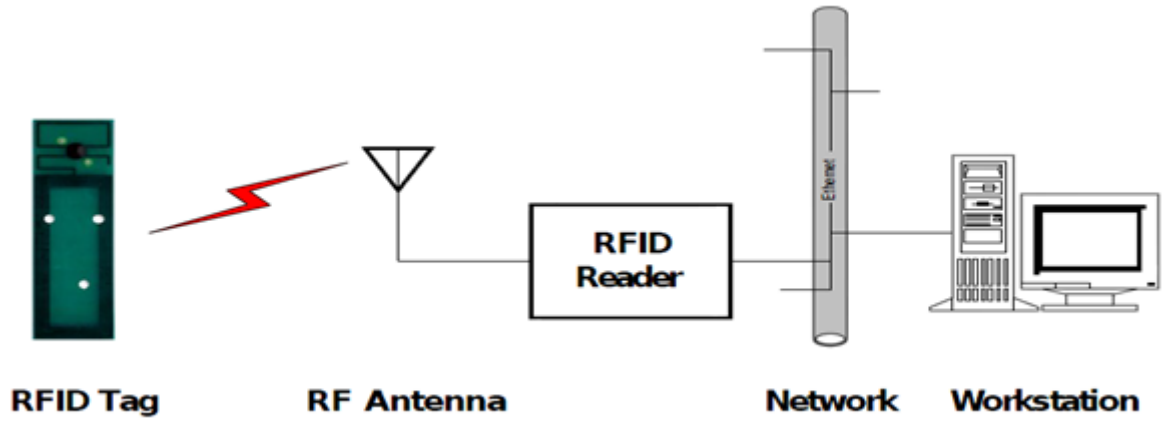


Figure 1-1: RFID System Components

1.3.1 RFID Tags

A Radio-frequency identification system uses *tags*, or *labels* attached to the objects to be identified. Two-way radio transmitter-receivers called *interrogators* or *readers* send a signal to the tag and read its response.

RFID tags contain at least two parts: an integrated circuit for storing and processing information, modulating and demodulating a radio-frequency (RF) signal, collecting DC power from the incident reader signal, and other specialized functions; and an antenna for receiving and transmitting the signal. The tag information is stored in a non-volatile memory. The RFID tag includes either a chip-wired logic or a programmed or programmable data processor for processing the transmission and sensor data, respectively.

An RFID reader transmits an encoded radio signal to interrogate the tag. The RFID tag receives the message and then responds with its identification and other information. This may be only a unique tag serial number, or may be product-related information such as a stock number, lot or batch number, production date, or other specific information.

Types of RFID Tags

There are two types of RFID tags. These are:

- **Passive RFID** tags don't have an internal source of power. There is an electrical current that is created in the antenna by the incoming radio frequency signal from the reader. This means that the antenna has to be able to collect power from the incoming signal and also transmit the outbound signal to the reader. A passive tag can respond with identification numbers or non-volatile storage data. It can be read from about 10 cm to a couple of meters and since they don't have to have a power source on the device, they can be extremely small(they can be embedded in a sticker or under the skin).
- **Active RFID tags** have their own internal source of power. This is used to power the circuits and broadcast signals to the reader. These tags are usually more reliable than passive tags. They also have a stronger signal because of their built-in power supply. This also allows them to work in places that passive tags wouldn't be able to, such as in water (which would include humans and other animals), metal, or from longer distances. They are however, larger and more expensive than passive tags. Today, active tags can transmit from hundreds of meters and their batteries can last for about 10 years. Some active tags contain different sensors that can read things like temperature, humidity, and radiation.

1.3.2 RFID Readers

An RFID Reader's function is to interrogate RFID tags. The interrogation is wireless and line of sight between the reader and tags is not necessary. A reader contains an RF module, which acts as both a transmitter and receiver of radio frequency signals. A microprocessor forms the control unit, which employs an operating system and memory to filter and store the data. The data is now ready to be sent to the network. RFID systems can be classified by the type of tag and reader.

1.3.3 RFID Security

Radio Frequency Identification (RFID) readers can read hundreds of tags per second and they do not require the line of sight, as for example a bar code scanner, thus allowing for fast automation of the reading process. RFID tags respond to any reader request within range. Consequently, a person carrying a tagged item effectively broadcasts a fixed identifier to nearby readers. So, anyone with a reader can read the information in the tag, potentially violating the owner's privacy [1].

Since RFID tags can be attached to clothing, possessions, or even implanted within people, the possibility of reading personally-linked information without consent has raised privacy concerns. Moreover, RFID applications have very limited resources, for example, tag memory is restricted to several hundred bits, and approximately 250–5000 logic gates out of the total tag space can be devoted for security-related tasks. The block ciphers used for security are lightweight because they aim to reduce the hardware resources needed.

Widespread RFID deployment creates privacy risks for everyone. In this case, a reader would periodically broadcast a query and all the reader in the vicinity will respond to that Tag. The reader will then forward these responses to the database [29].

For instance, an unauthorized user can use the reader or buy another to communicate directly with the tag. So, RFID data can easily be attacked [4]. There are various attacks like sniffing, tracking, spoofing, replay and Denial of service cause security and privacy risk for RFID technology.

1.3.3.1 Data Security Issues of RFID Tag

At present, RFID is mainly facing the following three issues of security [5]:

- Information of RFID tags are intercepted
- RFID tags can be cracked
- RFID tags can be copied

The security and privacy study [21] has the following objectives [5,6]:

- Investigate the security and privacy issues arise from the proposed use of RFID Technology

- Assess the capability of available technology to resolve those issues
- Provide recommendations to help meet security and privacy requirements

Security and privacy risk assessments [1, 21] of the passive RFID System are provided, with a description of potential countermeasures to address the stated risks.

General analysis of RFID technology indicates that several mitigation strategies are available to alleviate privacy and security concerns. Security mitigation strategies include the use of encryption, implementation of anti-collision algorithms to ensure reader availability and data integrity, the use of filters and audit trails to permit detection of counterfeit tags or replay attacks, and education of tag holders about the use of physical shielding.

1.3.3.2 Security Risk Assessment of the RFID System

The following are the security attacks which can be done on the RFID System [31]:

- **Counterfeit RFID Tag Attacks:** Counterfeiting attacks [1] which seek to duplicate legitimate tags through cloning or forgery. By virtue of its inexpensive functionality, the low cost RFID Tag contains an unencrypted identifier that can be stolen for duplication efforts.

- **Replay Attacks:** An attacker can perform replay attacks with counterfeit tags, mimicking the valid arrival and departure of tags. Replay attacks on a grand level can ultimately lead to a Denial of Service (DoS) attack in which counterfeit tags are replayed to readers in excess form. Authorized readers are inundated with counterfeit tags presented at a high rate. They consequently fail and cannot read legitimate tags. DoS attacks constitute a serious threat to system availability.

- **Eavesdropping Attacks:** Electronic access to tag contents occurs via eavesdropping by attackers in possession of rogue readers. A rogue reader is a reader that is not authorized to interrogate a tag or population of tags. Unsophisticated RFID Tags indiscriminately respond to RFID interrogation at the proper frequency and cannot differentiate between a rogue and authorized reader.

- **Electronic Collisions:** They occur when multiple RFID devices (readers and tags) respond to each other simultaneously, causing their communication signals to interfere with one another. Reader and/or Tag Collisions results in failed transmissions, lost data and faulty data integrity. Readers are also prevented from interrogating tags, a loss of system availability.

- **Introduction of Rogue Components:** RFID Readers and Middleware Access Points are units placed in strategic physical locations. To meet functionality needs, they are placed in close proximity to areas in which they are able to contact tags via radio frequency emissions. An intruder could gain access to a physical location and add a rogue reader.

1.3.3.3 Mitigation Strategies for Security Risks

Countermeasures to counterfeit tags [4] require an assurance of the confidentiality and integrity of tag data. Measures to prevent the disclosure or modification of tag contents include encryption and access controls. Unauthorized individuals, and rogue readers operating as their extensions, should be prevented from physically and electronically reading tags.

- **Encryption:** The management of symmetric keys, and the maintenance of a public key infrastructure to support asymmetric keys, is accompanied with high overhead costs. In addition, keys embedded on tags are prone to physical attacks, the success of which can lead to cloning.

- **One-Way Hash Locks:** A hash is a one-way function that converts a variable-length block of data into a fixed -length value called a “hash code.”[2] It cannot be reversed. A hash function known only to two parties provides two principles; it authenticates the sender and it provides integrity assurance for sent data.

- **Physical Shielding Sleeve (The Faraday Cage):** A Faraday Cage [2] is a metal mesh or foil container that is impenetrable by radio signals of certain frequencies. It can be used to shield a tag from unwanted eavesdropping, but requires owner compliance for use. A physical shield around the tag can serve as a potential threat to availability and integrity if it is not removed to allow legitimate readers to perform their scans.

- **The Selective Blocker Tag:** Many RFID readers implement anti-collision algorithms. These functions allow for a reader to talk with a single tag without interference from nearby tags that are also responding to the reader's interrogation signal [2]. Because these algorithms allow tags to be read singly, they are referred to as "singulation protocols." RFID Systems operating at the UHF range usually employ the silent tree-walking singulation protocol for anti-collision.

- **Secure Reader Protocol 1.0 Implementations:** Reader Protocol 1.0 is the EPCglobal[3] generated and industry accepted standard for defining communication between RFID Middleware and RFID Readers.

- **Anti-Collision Algorithms**

  Tag collisions can be prevented with the use of anti-collision [5] algorithms that essentially "singularize" a tag from a population of tags. RFID systems operating at 915 MHz generally implement the silent binary tree-walking algorithm as a singulation [2] technique. Although anti-collision algorithms [4, 5] are helpful commodities, their implementation nonetheless introduces significant tradeoffs that can potentially lead to less efficient and more costly systems. These tradeoffs include:
  - The speed which a tag may be read
  - The range at which a tag may be read
  - The bandwidth of the outgoing reader signal
  - The bandwidth of the incoming tag signal

Risks	**Security Objectives**	**Countermeasures**
Counterfeit Attacks	Confidentiality , Integrity	Encryption, One-way Hash Locks, Physical Shielding sleeve, Selective Blocker Tag
Replay Attacks	Availability, Integrity	One-way Hash Locks,

		Physical Shielding sleeve, Selective Blocker Tag
Eavesdropping Attacks	Confidentiality	Encryption, Physical Shielding sleeve
Electronic Collisions	Availability , Integrity	Anti-Collision Algorithms
Rogue Components	Availability , Integrity , Confidentiality, Non-Repudiation	Secure Reader Protocol

Table 1-1: RFID System Risks, Their Impacts and Countermeasures

1.4 Lightweight Cryptography

Cryptography on an RFID tag however comes with a cost: it consumes chip area, power, energy and time, which are very scarcely available on an RFID tag. This implies that standard cryptographic solutions cannot be deployed on lightweight devices like RFID tags. Thus, the rise of lightweight devices has created new challenges in cryptography.

Lightweight cryptography has been proposed to answer the demand for cryptography that uses minimal resources. Not only the cryptographic functions that are implemented on the chip should be minimized in terms of area, power or energy[8] while at the same time preserving an acceptable performance. Since cryptography also comes with a communication overhead, which in case of RFID tags can consume much energy and cause additional delay, this is an additional constraint that needs to be taken into account. It is clear that lightweight cryptography tries to minimize the impact of security and privacy protection[8] on the performance and cost of devices.

Lightweight cryptography is a branch of the modern cryptography, which covers cryptographic algorithms intended for use in devices with low or extremely resource constrained environment. Lightweight cryptography does not determine strict criteria for classifying a cryptographic algorithm as lightweight, but the common features of lightweight algorithms are extremely low requirements to essential resources of target devices.

It is a relatively new field aimed to develop more efficient cryptographic implementations in response to typical constraints in the hardware used in Internet of Things (IoT). The hardware used in IoT will likely be constrained in computational power, battery, as well as memory. Lightweight cryptography is tailored for such constrained devices, with the goal of balancing the tradeoffs between low resource requirements, performance, and cryptographic strength. Techniques used to meet this challenge include the adaptation of block ciphers, hash functions, and public key cryptography for lightweight cryptography.

Lightweight Cryptography is

- Cryptography tailored to (extremely) constrained devices
- Not intended to replace traditional cryptography
- Not intended for all extremely strong adversaries
- Not weak Cryptography

The expansion of smart technologies crucially raises data security problems. However, now it is impossible to suggest a cryptographic primitive that can be implemented in all types of target devices. We can tell that AES [20] is a really strong algorithm with good performance. It is absolutely advisable to use AES in high-end devices, in a large variety of embedded systems or in some low-end devices (with several constraints). But it is impossible to use common cryptographic algorithms in specific devices with extremely constrained resources. The examples of such devices include:

- RFIDs
- low-end smart cards (including wireless)
- wireless sensors
- Indicators, measuring devices, custom controllers etc.

Security & privacy issues for applications that can be termed as *lightweight security*, due to the associated constraints on metrics such as available power, energy, computing ability, area, execution time, and memory requirements. As such applications are becoming ubiquitous, definitely providing an immense value to the society, they are also affecting a greater portion of the public & leading to a plethora of economical & security and privacy related concerns. The goal is to create a platform where these concerns can be addressed and proposed solutions are discussed and evaluated. The solutions should be economically applicable in constrained

environments such as wireless embedded systems. Providing implementation results & demonstrating the applicability of the proposed solutions are among the essentials. Metrics to evaluate different aspects of lightweight security solutions and combined metrics for overall evaluations thereof for a given application scenario are useful for implementers and engineers. Compactness and efficiency are the properties which are commonly sought.

1.4.1 Design Strategies

Lightweight Cryptography is a relatively young scientific sub-field that is located at the intersection of *electrical engineering, cryptography and computer science* and focuses on new designs, adaption or efficient implementations of cryptographic primitives and protocols. Due to the harsh cost constraints and a very strong attacker model, there is an increasing need for lightweight security solutions that are tailored to the ubiquitous computing paradigm.

Every designer of lightweight cryptography has to cope with the trade-off between security, costs, and performance [9]. For block ciphers the key length provides a security-cost trade-off, while the amount of rounds provides a security-performance trade-off and the hardware architecture a cost-performance trade-off (see Figure).

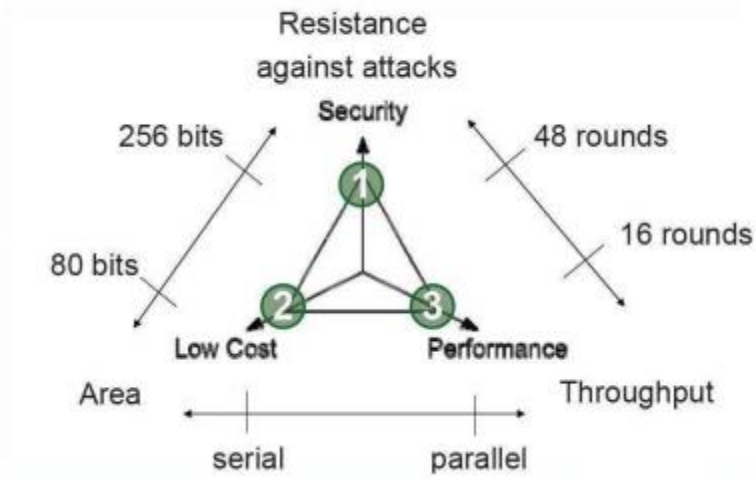


Figure 1-2: Design trade-offs between security/cost/performance for lightweight cryptography

Usually, any two of the three design goals – security and low costs, security and performance, or low costs and performance – can be easily optimized, whereas it is very difficult to optimize all three design goals at the same time.

For example, a secure and high performance hardware implementation can be achieved by a pipelined architecture which also incorporates many countermeasures against side-channel attacks. The resulting design would have a high area requirement, which correlates with high costs. On the other hand it is possible to design a secure and low-cost hardware implementation with the drawback of limited performance.

Generally speaking, there are three approaches for providing cryptographic primitives for extremely lightweight applications such as RFID tags:

- Optimized low-cost hardware implementations for standardized and trusted algorithms.
- Slightly modify a well investigated and trusted cipher.
- Design new ciphers with the goal of having low hardware implementation costs.

The problem with the first approach is that most modern block ciphers were primarily designed with good software implementation properties in mind, and not necessarily with hardware-friendly properties. This is the right approach for today's block ciphers, because on the one hand the vast majority of algorithms run in software on PCs or embedded devices, and on the other hand silicon area has become so inexpensive that very high performance hardware implementations (achieved through large chip area) are not a problem.

The second approach is to have a well investigated cipher, the design of which was driven by

low hardware costs. A very well known cipher to this respect is the Data Encryption Standard, DES. DES was designed in the first half of the 1970s and the targeted implementation platform was hardware.

Hence, virtually all components of DES were heavily driven by low hardware complexity: exclusive bit-wise OR (XOR), bit permutation and small S-boxes. We will follow the second approach by slightly modifying DES in order to gain DESL [18]. The obvious drawback of DES is that its key length is not adequate for many of today's applications, but by applying key-whitening techniques the security level can be increased.

Though the implementation results of DESL are encouraging, they also show optimization potentials. In order to further decrease the hardware area requirements, the third approach ie design of *the ultra-lightweight cipher* is more preferred.

1.4.2 Metrics

To assess the efficiency of the implementation, the following metrics are used.

- **Area:** Area requirements are usually measured in **µ**m^2, but this value depends on the fabrication technology and the standard cell library. In order to compare the area requirements independently it is common to state the area as gate equivalents [GE]. One GE is equivalent to the area which is required by the two-input NAND gate with the lowest driving strength of the appropriate technology. The area in GE is derived by dividing the area in **µ**m^2 by the area of a two-input NAND gate.

- **Cycles:** Number of clock cycles to compute and read out the result.

- **Time:** The required amount of time for a certain operation can be calculated by dividing the amount of cycles by the operating frequency such that t = (cycles / freq). In most cases the time is given in milli seconds [ms].
- **Throughput:** The rate at which new output is produced with respect to time. The number of output bits is divided by the time, i.e. by the needed cycles and multiplied by the operating frequency. It is expressed in bits per second [bps].

- **Power:** The power consumption is estimated on the gate level by Synopsys Power Compiler. It is provided in micro Watt [µW]. Note that power estimations on the transistor level are more accurate, but this would also require further design steps in the design flow.

- **Energy:** The energy consumption denotes the power consumption over a certain time period. It can be calculated by multiplying the power consumption with the required time

of the operation. The energy consumption is provided in micro Joule [μJ] or micro Joule per bit [μJ/bit].

- **Efficiency:** Throughput to Area ratio is used as a measure of hardware efficiency. It is expressed in gate equivalents per bits per second [GE/bps].

1.4.3 Design Considerations

RFID applications have very limited resources, for example, tag memory is restricted to several hundred bits, and approximately 250–5000 logic gates out of the total tag space can be devoted for security-related tasks. The block ciphers used for security are lightweight because they aim to reduce the hardware resources needed

1.4.3.1 Software vs Hardware Implementation

Cryptography algorithms demand high processing capabilities. A software implementation [26] of these algorithms requires more clock cycles and multiplies instructions to execute the partial operations and thus increase the total consumed energy. In addition, Software-based encryption provides privacy for data residing on the computer systems disk by using the system CPU to perform the encryption/decryption and related cryptographic operations. The main scope is the designing of the appropriate hardware primitives that make these operations more efficient in terms of energy and area resources consumed. Hardware-based encryption [26] moves the encryption/decryption function inside the hard disk drive. Isolating the encryption functions and keys in the disk drive subsystem, where they are not accessible by the operating system, is advantageous because it protects these security components from root kits and malware. In addition, utilizing dedicated hardware in the disk drive to perform the encryption and decryption offloads results in system performance that is closer to that of an unencrypted computer.

Important disadvantages that are common to most software-based encryption include performance, which is generally noticeably worse than on hardware encryption products.

Configuration complexity and the amount of time needed to initially set up the software are also disadvantages.

1.4.3.2 Characteristics of the ciphers

A quite huge range of ciphers were implemented, starting with DESL providing a lower security level with 56-bit key, and continuing with PRESENT[15,25] , which provides medium level of security with 80-bit key and 64-bit block size. The HIGHT ciphers provide a better level of security with 128-bit key and 64-bit block size. Last, DESXL was implemented that uses 184-bit key and 64-bit block size. DESL, DESXL, HIGHT are Feistel ciphers. CURUPIRA-1[25], and PRESENT [15] are Substitution-Permutation network (SP-network) ciphers. Some algorithms are based on the concept of pseudorandom numbers[7] .

Feistel ciphers modify only half of the block in each round, while the SP-network ciphers modify the complete block in each round. Also, some substitution boxes (S-boxes), for example in CURUPIRA-1 ciphers, are implemented in either combinational logic or with memory elements, while other S-boxes (DESL, DESXL and PRESENT ciphers) may be implemented only using memory elements as their combinational equations are not given by the ciphers' constructors. The major problem with the S-boxes operation is that it cannot be easily encoded in a linear equation.

For the above reasons a general architecture option for all ciphers' implementations was used and no algorithm specific hardware designs were used for each cipher. It is impossible to apply the same design optimizations to all ciphers in order to examine the optimization methods efficiently and accurately.

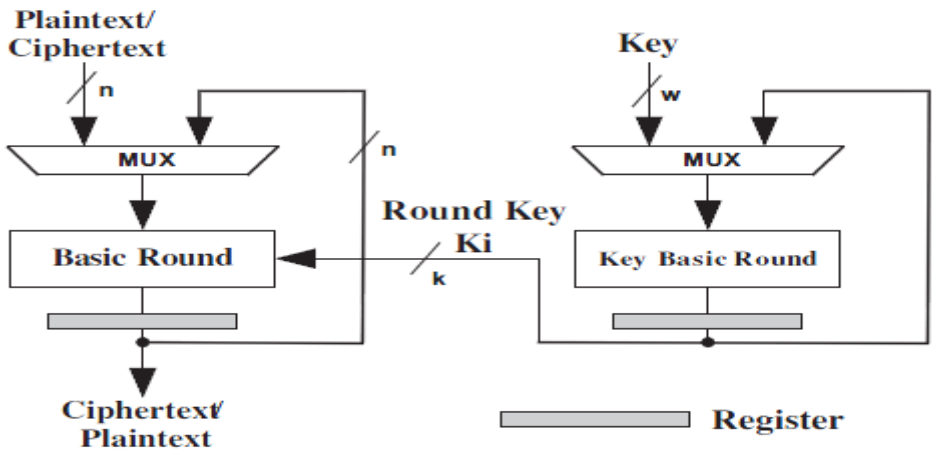


Figure 1-3: Basic Round Unit

Above figure shows the general architecture of majority of the algorithms. In this figure, only one round of each cipher is implemented. The output of the basic round unit is buffered and used as input to the next round. During initialization the multiplexer chooses the plaintext and then chooses the output of the basic round unit. In this architecture the key scheduler also consists of one basic round. The produced round subkey is used both for the data encryption/decryption and as input to the next key round. All S-boxes used by the ciphers have been implemented by Look-Up-Tables (LUTs) using ROM blocks.

The iterative looping architecture was used for all ciphers. As the payload data transferred in RFID applications is too small and the bit rate is also low. So, the full loop unrolling architecture with successive rounds and pipeline stages between each round is not a practical solution.

CHAPTER 2
BASE WORK

This work describes a lightweight security mechanism for securing transactions conducted over the RFID platform. RFID is gaining popularity and it is widely accepted. However, security over the platform is more critical due to the open nature of wireless networks. Furthermore, security is more difficult to implement on these platform because of the resource limitation of these devices. Therefore, security mechanisms for protecting traditional computer communications need to be revisited so as to ensure that transactions involving these devices can be secured and implemented in an effective manner.

This research is part of designing security infrastructure for securing the data which needs to be protected from unauthorised access. A lightweight mechanism was designed to meet the security needs in face of the resource constraints. The proposed mechanism is proven to be practical in real deployment environment.

Due to the pervasiveness of high-speed networks and multimedia communications and storage, the demand for high speed cryptosystems is ever increasing. The Cryptographic techniques and algorithms which are considered as highly secure, for example AES, takes time to encrypt the data and thus are not suitable for high speed networks. Highly Secure Systems that have embedded security mechanism to ensure its security operating requirements, which may sacrifice its performance due to limited system resources. Therefore, security for data transmission over wireless channel results in throughput loss. Trade-off between security and throughput is always a major concern in wireless networks.

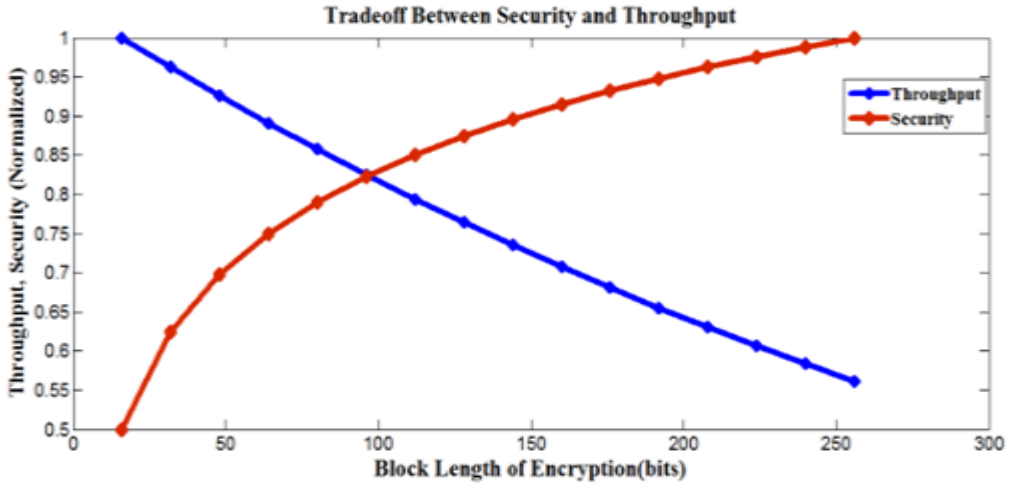


Figure 2-1 : Security-Throughput Tradeoff as a function of Block Length

No existing encryption algorithms are both fast enough for high-speed operation and sufficiently secure to withstand powerful cryptanalysis. In most of the cryptographic algorithms, our primary focus is to protect the Security key which is used in the encryption process. If the key is compromised, the data can be easily deciphered by the unauthorised person. So the primary aims at protecting the keys from unauthorized intrusion.

Feng Bao and Robert H. Deng[10] has proposed, a generic construction of high-speed encryption schemes. The solution is based on the fact that there exist secure but relatively slow block ciphers, e. g. AES[13, 20], and super-fast but relatively weaker stream ciphers. Secure block cipher with a super-fast stream cipher is combined in such a way that the resulting encryption scheme possesses both the speed of the stream cipher and the security of the block cipher. Secure block cipher is used to generate the keys which later are used to encrypt the text using Stream Cipher. We will show the performance analysis as well as our experiment on a Xilinx simulator. However, AES is inadequate for resource constrained devices since it is too heavy and slow for data. Furthermore, AES provides very low compression rate.

2.1 High speed requirements

A growing need of ultra – high speed data transfers has motivated continuous improvements in the physical layer transmission speed. Communication applications are on the rise for a

variety of services, such as multimedia electronic mail, video conferencing, and high-definition televisions. Various types of high-speed networks exist. For example is the Asynchronous Transfer Mode (ATM) technology which provides data rates from tens of Mb/s to Gb/s. As new network services become more capable and user friendly, high-speed networks will continue to attract more traffic and more sensitive information. This type of speed is necessary for streaming high definition video, playing online games and sending and receiving large amounts of data.

However, as researchers develop software and protocols to operate over such networks , they often fail to account for security .The processing power required to encrypt or sign data can significantly decrease transfer rates, and thus security is often sacrifice for throughput. High speed networks need to provide sufficient security in number of specific application so as to prevent any data compromise. So, secure cryptographic algorithms should be implemented in these networks to provide sufficient level of Security. There should be efficient encryption algorithms so that speed does not become a performance bottleneck in bandwidth-hungry applications.

2.2 Trade-off between Speed and Security

A **stream cipher** is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digital stream (keystream). In a stream cipher each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the ciphertext stream. Block ciphers operate on large blocks of digits with a fixed, unvarying transformation.

Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity. However, stream ciphers can be susceptible to serious security problems if used incorrectly (see stream cipher attacks); in particular, the same starting state (seed) must never be used twice i.e. in Stream Cipher the security of the algorithm rests on the security of the key. If key is compromised, the algorithm is no longer considered as safe.

An example is the stream cipher TWOPRIMES, which is proved to possess high linear complexity, large cycle length, good resistance to LSFR-synthesis attacks and many other

desirable attributes of good stream ciphers; however, it was broken in with only a complexity of 2^{32}.

One way of providing security in High Speed networks is by using Stream Ciphers which are fast enough to meet the high speed requirement but they do not provide high security. We use Block Cipher which are highly secure (for example AES Cipher), but they are not fast enough to meet high speed requirements.

2.3 Summary of Research Contributions and Outline

In Stream Cipher, the security of the cipher rests on the key stream used in Encryption. If key is compromised, the cipher is no longer secure. So, in this work, we aim at designing super-fast and secure symmetric key encryption schemes. However, we do not propose new encryption algorithms. What we do is to combine existing strong algorithms (relatively slow) with fast algorithms to form new encryption schemes, which are fast enough to satisfy the high-speed requirement for various applications.

Symmetric key ciphers can be roughly classified into stream ciphers and block ciphers. Block ciphers usually have stronger security than stream ciphers. Here we refer to the traditional stream cipher based on the LFSR. A block cipher is typically an iterated function on a fixed block size. The security mainly comes from the iteration.

For a block cipher, it is impossible to express the output and the input in an explicit algebraic formula. This is because the iteration function makes the formula expand into an overwhelmingly large size. Block ciphers are usually designed to resist various kinds of attacks, including linear attack and differential attack [11] and so on. Most of the attacks to block cipher are aimed at deriving the key. So a secure block cipher should be able to keep the key forever secret, even under attacks where attackers can arbitrarily choose plaintext/ciphertext pairs as they want. Examples of such block ciphers are DES, AES, IDEA and SERPENT [12].

A stream cipher is usually a pseudo-random generator. The pseudo-random sequence (key stream) generated by the pseudo-random generator is XORed with the plaintext to obtain the

ciphertext. The pseudo-random generator may or may not depend on the plaintext and cipher text. eg Algorithms based on multi-recursive generator [17].

This proposed approach combines a secure block cipher with a fast stream cipher such that the encryption scheme can have both the security of the block cipher and the speed of the stream cipher. The secret key of our encryption schemes is protected by the block cipher while the large plaintext is encrypted by the stream cipher with segment keys generated from the block cipher.

2.4 Different Variants of Fast Stream Cipher

Let **BE**(*K, m*) denote a block cipher encryption algorithm (such as AES) on message *m* using key *K* and **SE**(*k, M*) denote a stream cipher encryption algorithm on message *M* using key *k*. Here *m* has fixed size, i.e., the block size of **BE** while *M* has arbitrary size. We divided a plaintext into segments with equal size (padding may be applied to the last segment):

$$Plaintext = pseg_1, pseg_2, ..., pseg_t$$

The segment size is the number of bits of seg_i . Let *K* be the secret key of the scheme, which is a key of **BE**. The encryption is performed as follows. First we randomly choose a number *r* of the block size of BE, then generate the segment keys as

$$k1 = BE(K, r), \qquad k2 = BE(K, k1), \ldots, kt = BE(K, kt\text{-}1).$$

The corresponding ciphertext is given by

$$Ciphertext = r, cseg_1, cseg_2, ..., cseg_t$$

where $cseg_i = SE(k_i, pseg_i)$

Note that *r* precedes the ciphertext so that decryption can be carried out at the receiver. We require that *r* never be reused. In our schemes, *r* is set to be 128-bit; therefore, the probability that two randomly selected *r*'s happen to be the same is as small as correctly guessing the key *K*. (The *r* can also be generated from the plaintext, say, let *r* = **BE**(the first 128 bits of the plaintext, *K*). There are actually many ways to generate *r*.

2.4.1 Fast Encryption Scheme Variant I

Let BE be the secure Block Cipher and SE be the fast Stream Ciphers. Let the Plaintext be $P_1, P_2, ..., P_{t+1}$ and the corresponding cipher text will be $C_1, C_2, ..., C_{t+1}$.

The first block P_1 is not in the original plaintext. It may be a random head appended to the original plaintext before the encryption. It may also be a number used as a counter to indicate the i^{th} encryption. It is fine as long as P_1 satisfies

a) P_1 is different for different plaintext or b) P_1 is not required to be confidential.

We have

1. $C_1 = P_1$

2. $C_2, C_3, ..., C_n = SE(BE(k, P_1), P_2 P_3, ..., P_n)$

3. $C_{(in+1)} C_{(in+2)}, ..., C_{((i+1)n+1)} = SE(BE(K, P_{(in)}), P_{(in+1)} P_{(in+2)}, ..., P_{((i+1)n+1)})$ for $i = 1, 2, \ldots, t$.

The decryption is easy just reverse of encryption.

Description of SE

Let the plaintext be

$$b_1 b_2 ... b_m$$

where each b_i is a 32-bit string. Let F be a function defined as

$$F(k, x) = ((((x + k_1) \oplus k_2) \times k_3) \oplus k_4) >>>$$

Notations

K	the 128-bit key and $k = k_1 k_2 k_3 k_4$ for 32-bit k_i
X	32-bit string
$\oplus$	the bit-wise exclusive-OR
+	mod 2^{32} addition
$\times$	mod 2^{32} multiplication
>>>	to reverse the 32 bits into opposite ranking

The encryption of $b_1 b_2 ... b_m$ is

$$d_i = b_i \oplus F(k, F(k, F(k, d_{(i-1)}) \oplus b_{(i-1)}) \oplus d_{(i-2)})$$

where $d_1 d_2 ... d_m$ *is* the cipher text. The $d_0 d_{-1} d_{-2}$ can be set $k_2 k_3 k_4$.

The Block cipher used in the fast encryption variant is **AES** [13, 33] which is highly secure and the key size in AES is 128,192 and 256 bits. In addition, the AES was designed primarily for hardware and is relatively slow when implemented in software.

AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys – 128, 192, 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES-256 respectively. As well as these differences AES differs from DES in that it is not a feistel structure. AES was designed to have the following characteristics:

- Resistance against all known attacks eg, Differential Cryptanalysis[11] , Linear Cryptanalysis attacks
- Speed and code compactness on a wide range of platforms.
- Design Simplicity.
- It is implementable on both software and hardware.

2.4.2 Fast Encryption Scheme Variant II

In this Fast Encryption Variant, the block cipher is used to generate segment keys and the stream cipher is used to generate key streams from segment keys. The block cipher used is AES.

Design of Stream Cipher:

This stream cipher is used to extend a 128-bit key into a key stream of segment size.

Notations

&	Bit-wise AND,
⊕	Bit-wise XOR,
>>>	Right Rotation

T	Table containing 32 elements, each element is with 32 bits.
F	Feedback function,
G	Output function.
K	The 128-bit secret key. It consists of four 32-bit words: k_1, k_2, k_3, k_4
$C_i (0 \le i \le 31)$	Each one is a 32-bit constant. They are generated from the constant e in the following way: for i = 0 to 31 $C_i = (e \times 2^{32(i+1)} \ \& 0xFFFFFFFF$;
$r_i \ (0 \le i \le 63)$	Each one is between 3 and 14. They are generated from the constant π in the following way: for i = 0 to 63 $r_i = ((\prod \times 2^{8(i+1)}) \ \& 0xFF) mod 12 + 3$;
F	The input of function F is the table T and two rotation constants r_1 and r_2. The output of function F is denoted as *feedback*. F operates in the following way: $tem = (((((T[0] \oplus T[22]) >>> r_1) + T[10]) \oplus T[27]) >>> r_2) + T[15])$ $feedback = tem \oplus T[tem \ \& 31]$
G	The input of function G is the table T. The output of function of G is denoted as output. G operates in the following way: $tem = (((T[29] \oplus T[23]) >>> r_1) + T[20]) \oplus T[13]) >>> r_2) + T[2])$ $output = tem \oplus T[tem \ \& 31]$

The operation of this stream cipher consists of two stages: an initial setup stage and an output stage.

Initial Setup

1. Initialize the table T

for i= 0 to 31

$$T_i = C_i + k_{(i mod 4)}$$

2. Run the Main algorithm (given below) for 64 cycles and prepare for the output.

3. **The Main Algorithm**

For the *i*th cycle, the cipher operates in the following way:

1. Run the *F* function with $\boldsymbol{r}_{(2i \bmod 32)}$ *and* $\boldsymbol{r}_{(2(i+1) \bmod 32)}$, obtain the value of *feedback* .

2. for $j = 0$ to 30

$$T[j] = T[j+1] ;$$

$$T[31] = feedback ;$$

3. Run the function *G* and generate the *output.*

CHAPTER 3
PROPOSED WORK

3.1 Problem Statement

The plethora of available security primitives are too excessive in terms of cost to be implemented on a cost constrained RFID chip. Low cost labels are also not self-powered and only consist of limited logic functionality, unlike smart card processors. For instance, private key cryptosystem such as AES are not suitable since a commercial implementation of AES typically requires 20,000 – 30,000 gates. This is far more than the number of gates on an entire low cost label. However the SHA-1 specified by the US Department of commerce is a possible candidate for an encryption rule but hardware implementations of SHA-1 are currently too costly to meet the cost budget of low cost RFID labels. Cryptographic systems and protocols need to fit into a label footprint without dramatically increasing the cost of a label.

The problem with the existing scheme is that we were using AES scheme which is highly secure block cipher techniques but due to high computational complexity, the technique is not suitable for light weight devices like RFID tags, smart cards etc. In fact, it requires an extra module for implementing the Galois field multiplication which, additionally, has to be invoked several times.

Instead of using AES which is not suitable for RFID, we propose an algorithm ultralight weight Hummingbird Cryptographic Algorithm. Different from existing (ultra-lightweight) cryptographic primitives which are either block ciphers or stream ciphers, Hummingbird is an elegant combination of the above two cipher structures with a 16-bit block size, 256-bit key size, and 80-bit internal state. The size of the key and the internal state of Hummingbird provides a security level which is adequate for many RFID applications. Existing lightweight Cryptographic algorithms are PRESENT[15], KATAN [23, 25] etc.

The fast and efficient stream Ciphers are given by eSTREAM[24] project which was a multi-year effort to promote the design of efficient and compact stream ciphers suitable for widespread adoption. The eSTREAM portfolio ciphers fall into two profiles. Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements. Profile 2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption.

So in our scheme, Hummingbird is used to generate keys which are used in stream ciphers given in eSTREAM[24] project, for encrypting the data.

3.2 Hummingbird Cryptographic Algorithm

Hummingbird Cryptographic Algorithm is a new ultra-lightweight cryptographic algorithm which is mainly designed for resource-constrained devices like smart cards, RFID tags, and wireless sensor nodes.

The radiofrequency identification (RFID) technology provides an extensible, flexible and secure measure against product counterfeiting. However, due to the limited cost and power constraints of RFID tags, only dedicated cryptographic engines or low-power consumption microcontrollers can be integrated into tags to implement various security mechanisms. Hummingbird provides designed security with small block size and it is resistant to the most common attacks such as linear and differential cryptanalysis[11] , birthday attack As the key size of Hummingbird cryptographic algorithm is very large i.e. 256 bits , so it can be used to provide desired security to lightweight devices like RFID tags , smart cards etc.

The key issue of designing lightweight cryptographic algorithm is to deal with the trade-off among *security, cost and performance*. Hummingbird has a *Hybird structure* of both stream cipher and blocks cipher and is developed with both lightweight software and lightweight hardware for constrained devices in mind. The hybrid model can provide designed security with small block size and is therefore expected to meet the stringent response time and power consumption requirement for the large variety of embedded applications.

Notations

PT_i	The **i**th Plaintext block , i = 1,2,3,……,n
CT_i	The **i**th Ciphertext block , i = 1,2,3,…….,n
K	The 256-bit secret key
$E_k(...)$	The Encryption function of Hummingbird with 256-bit secret key K
K_i	The 64-bit subkey used in **i**th block cipher, i = 1,2,3,4, such that $\boldsymbol{K = K_1 \,||\, K_2 \,||\, K_3 \,||\, K_4}$
$E_{k_i}(...)$	A block cipher encryption algorithm with 16-bit input,64-bit key K_i , and 16-bit output , ie $\boldsymbol{E_{k_i} : \{0,1\}^{16} \times \{0,1\}^{16} \rightarrow \{0,1\}^{16}, i = 1,2,3,4}$
RSi	The **i**th 16-bit internal state register
LFSR	16–stage Linear Feedback Shift Register with the characteristic polynomial f(x) such that , $\mathbf{f(x) = x^{16} + x^{15} + x^{12} + x^{10} + x^7 + x^3 + 1}$
+	Modular 2^{16} addition operator
$\oplus$	Exclusive –Or (XOR) operator
m<<< l	Left circular shift operator which rotates all bits of m to left by l-bits
$k_j^{\,i}$	The jth 16-bit key used in the block cipher, j = 1,2,3,4 such that as $\boldsymbol{K_i = K_1^i \,||\, K_2^i \,||\, K_3^i \,||\, K_4^i}$
S_i	The **i**th 4-bit to 4-bit S-Box used in the block cipher
$NONCE_i$	The **i**th nonce which is a 16-bit random number
IV	64-bit initial vector, such that $\boldsymbol{IV = NONCE_1 \,||\, NONCE_2 \,||\, NONCE_3 \,||\, NONCE_4}$

3.2.1 Encryption and Decryption

The design of Hummingbird is based on an elegant combination of block cipher and stream cipher with 16 -bit block size, 256-bit key size, and 80-bit internal state. The size of the key and the internal state of Hummingbird provides a security level which is adequate for many embedded Applications.

The overall structure of the Hummingbird encryption algorithm (see Figure 3-1(a)) consists of four 16-bit block ciphers $\boldsymbol{E_{k_1}, E_{k_2}, E_{k_3}, E_{k_4}}$, four 16-bit internal state registers $RS1, RS2, RS3, RS4$, and a 16-stage LFSR. The 256-bit secret key *K* is divided into four 64-bit subkeys $\boldsymbol{K_1, K_2, K_3, K_4}$ which are used in the four block ciphers, respectively.

3.2.1.1 Initialisation Process

Figure 3-1 illustrates the initialization process of the Hummingbird cryptographic algorithm. The initialization consist of four 16-bit block ciphers $\boldsymbol{E_{k_i}}$ (i = 1, 2, 3, 4), four 16-bit internal state registers $\boldsymbol{RS_i}$ (i = 1, 2, 3, 4), and a 16-stage Linear Shift Feedback Register (LFSR). Moreover, the 256-bit secret key K is divided into four 64-bit subkeys $\boldsymbol{K_1, K_2, K_3, K_4}$ which are used in the four block ciphers, respectively.

When using Hummingbird in practice, four 16-bit random nonces $\boldsymbol{NONCE_i}$ are first chosen to initialize the four internal state registers $\boldsymbol{RSi}$ (i = 1, 2, 3, 4), respectively, followed by four consecutive encryptions on the message $RS1 - RS3$ by Hummingbird running in initialization mode. The final 16-bit Ciphertext TV is used to initialize the LFSR. Moreover, the 13th bit of LFSR is always set to prevent zero register. The LFSR is also stepped once before it is used to update the internal state register $\boldsymbol{RS_4}$.

3.2.1.2 Encryption Process

A 16-bit plaintext block $\boldsymbol{PT_i}$ is encrypted by first executing modulo $\mathbf{2^{16}}$ addition of PT_i and the content of the first internal state register RSi. The result of the addition is then encrypted by the first block cipher $\boldsymbol{E_{k_1}}$. This procedure is repeated in a similar manner for another three

times and the output of E_{k_4} is the corresponding ciphertext CT_i. Furthermore, the states of the four internal state registers will also be updated in an unpredictable way based on their current states, the outputs of the first three block ciphers, and the state of the LFSR. The decryption process (see Figure 3-1) follows the similar pattern as the encryption. The exact encryption/decryption procedure and the internal state updating of Hummingbird are illustrated in the following algorithms. When using Hummingbird in practice, four 16-bit random nonce $NONCE_i$ are first chosen to initialize the four internal state registers RSi (i = 1, 2, 3, 4), respectively, followed by four consecutive encryptions on the message by Hummingbird running in the initialization mode. The final 16-bit ciphertext is used to initialize the LFSR. Moreover, the **13th** bit of the LFSR is always set to prevent a zero register. The LFSR is also stepped once before it is used to update the internal state register.

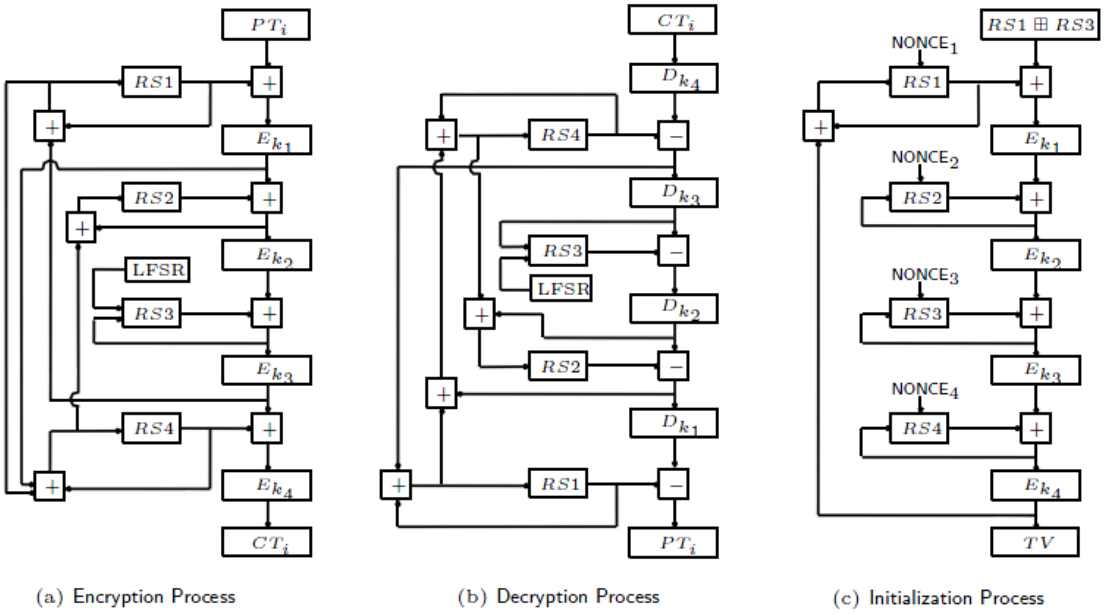


Figure 3-1: Hummingbird Cryptographic Algorithm

3.2.1.3 Block Cipher Encryption

Four identical 16-bit block ciphers are employed in a consecutive manner in the Hummingbird encryption scheme. The 16-bit block cipher is a typical substitution permutation (SP) network with 16-bit block size and 64-bit key as shown in figure. It consists

of four regular rounds and a final round that only includes the key mixing and the S-box substitution steps. The 64-bit subkey K_i is split into four 16-bit round keys $K_1^i, K_2^i, K_3^i, K_4^i$ which are used in the four regular rounds, respectively. Moreover, the final round utilizes two keys K_5^i, K_6^i directly derived from the four round keys. Like any other SP network, one regular round comprises of three stages: a key mixing step, a substitution layer, and a permutation layer. For the key mixing, a simple exclusive-OR operation is used in this 16-bit block cipher for efficient implementation in both software and hardware. The permutation layer in this 16-bit block cipher is given by the linear transform

$L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$ defined as follows:

$$L(m) = m \oplus (m << 6) \oplus (m << 10);$$

where $m = (m_0, m_1, ..., m_{15})$ is a 16-bit data block.

Four S-Boxes Given in Hexadecimal Notation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

Linear Transform $L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$

$L(m) = m \oplus (m \lll 6) \oplus (m \lll 10)$

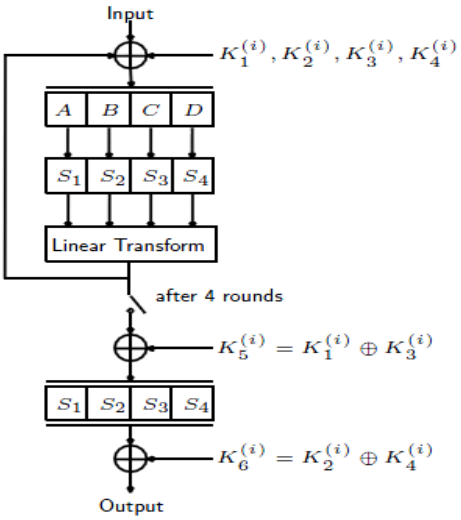


Figure 3-2: The Structure of the Block Cipher in the Hummingbird

The various algorithms used in Hummingbird Encryption process are as follows.

Algorithm 1 Hummingbird Initialization

Input : Four 16-bit random nonce $NONCE_i$ (i = 1,2,3,4)

Output : Initialised four rotors RSi_0 (i = 1,2,3,4) and LFSR

1: $RS1_0 = NONCE_1$ [Nonce Initialization]

2: $RS2_0 = NONCE_2$

s

3: $RS3_0 = NONCE_3$

4: $RS4_0 = NONCE_4$

5: for t = 0 to 3 do

6: $V12_t = E_{k_1}((RS1_t + RS3_t) + RS1_t)$

7: $V23_t = E_{k_2}(V12_t + RS2_t)$

8: $V34_t = E_{k_3}(V23_t + RS3_t)$

9: $TV_t = E_{k_4}(V34_t + RS4_t)$

10: $RS1_{t+1} = RS1_t + TV_t$

11: $RS2_{t+1} = RS2_t + V12_t$

12: $RS3_{t+1} = RS3_t + V23_t$

13: $RS4_{t+1} = RS4_t + V34_t$

14: end for

15: $LFSR = TV_t \,||\, 0x1000$ [LFSR Initialization]

16: Return RSi_t (i = 1,2,3,4) and LFSR

Algorithm 2 Hummingbird Encryption

Input : A 16-bit plaintext PT_i and four rotors RSi_t (i = 1,2,3,4)

Output : A 16-bit Ciphertext CT_i

1: $V12_t = E_{k_1}(PT_i + RS1_t)$ [Block Encryption]

2: $V23_t = E_{k_2}(V12_t + RS2_t)$

3: $V34_t = E_{k_3}(V23_t + RS3_t)$

4: $CT_i = E_{k_4}(V34_t + RS4_t)$

5: $LFSR_{t+1} = LFSR_t$ [Internal State Updating]

6: $RS1_{t+1} = RS1_t + V34_t$

7: $RS3_{t+1} = RS3_t + V23_t + LFSR_{t+1}$

8: $RS4_{t+1} = RS4_t + V12_t + RS1_{t+1}$

9: $RS2_{t+1} = RS2_t + V12_t + RS4_{t+1}$

10: Return CT_i

Algorithm 3 A 16-bit Block Cipher Encryption $E_{k_i}(.)$

Input: A 16-bit data block $m = (m_0, m_1, ..., m_{15})$ and a 64-bit subkey k_i , such that

Subkey $K_i = K_1^i \| K_2^i \| K_3^i \| K_4^i$

Output: A 16-bit data block $m' = (m_0', m_1', ..., m_{15}')$

1: For j = 1 to 4 do

2: $m \leftarrow m \oplus K_j^i$ [key mixing step]

3: $A = m_0 \| m_1 \| m_2 \| m_3, \quad B = m_4 \| m_5 \| m_6 \| m_7,$

$C = m_8 \| m_9 \| m_{10} \| m_{11}, \quad D = m_{12} \| m_{13} \| m_{14} \| m_{15}$

4: $m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$ [substitution layer]

5: $m \leftarrow m \oplus (m << 6) \oplus (m << 10)$ [permutation layer]

6: End for

7: $m \leftarrow m \oplus K_1^i \oplus K_3^i$

8: $A = m_0 \| m_1 \| m_2 \| m_3, \quad B = m_4 \| m_5 \| m_6 \| m_7,$

$C = m_8 \| m_9 \| m_{10} \| m_{11}, \quad D = m_{12} \| m_{13} \| m_{14} \| m_{15}$

9: $m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$

10: $m \leftarrow m \oplus K_2^i \oplus K_4^i$

11: return $m' = (m_0', m_1', ..., m_{15}')$

3.2.3 Security Analysis of Hummingbird Cryptographic algorithm

Hummingbird cryptographic algorithm by showing that it is resistant to the most common attacks[14] to block ciphers and stream ciphers including birthday attack, differential and linear cryptanalysis, etc. Note that Hummingbird has a hybrid mode of block cipher and stream cipher (This is the reason that the analysis in cannot be employed directly here.),

which can be considered as a finite state machine with the internal state (*RS*1, *RS*2, *RS*3, *RS*4, LFSR). However, the value of LFSR does not depend on those of *RS*1, *RS*2, *RS*3, and *RS*4. The purpose of using the LFSR is to guarantee the period of the internal state is at least $\mathbf{2^{16}}$.

- **Birthday Attack** on the Initialization. For a fixed key, one may want to find two identical internal states (RS1, RS2, RS3, RS4, LFSR) initialized by two different IV s using the birthday attack. However, if we fix the key in the initialization procedure of the Hummingbird encryption scheme, the mapping is one-to-one such as

  $$(RS1_t, RS2_t, RS3_t, RS4_t) \rightarrow (RS1_{t+1}, RS2_{t+1}, RS3_{t+1}, RS4_{t+1}).$$

  Hence the birthday attack does not work in this case.
- **Differential Cryptanalysis and Linear Cryptanalysis**: Hummingbird cryptography is resistant to differential and linear cryptanalysis [11].
- **Slide and Related-Key Attack:** Both slide attacks and related-key attacks [21] need to exploit the weakness of key scheduling. However, there is no key scheduling in Hummingbird. In particular, the subkeys used in four small block ciphers are independent. In addition, the four rotors affect the output of each small block cipher in a nonlinear way. Hence, both slide attacks and related-key attacks cannot be applied to the Hummingbird.
- **Interpolation and Higher Order Differential Attack:** Interpolation and higher order differential attacks can be applied to block ciphers with the low algebraic degree. As we discussed before for algebraic attack, the algebraic degree of the Hummingbird encryption is high. Hence it is difficult to apply interpolation and higher order differential attacks to the Hummingbird.
- **Complementation Properties:** The DES has the following well-known complementation[28] property, namely that if C is the ciphertext of the plaintext P under key K, then C' is the ciphertext of P' under key K, where x is the bitwise complement of x. However, Hummingbird does not have this weakness due to the presence of the carry propagation resulting from four rotors.
- **Structural Attack:** The internal state transition in Humming-bird encryption scheme is much more complicated. Hence, those attacks cannot be simply applied to the Hummingbird encryption scheme.

In this paper, we describe efficient hardware implementations of a stand-alone Hummingbird component in field-programmable gate array (FPGA) devices. We implement an encryption only core on the low-cost Xilinx FPGA series Spartan-2E family on XC2S300e device with speed grade of (-6) on ft256 package and compare our results with the already existing scheme on the same series. Our experimental results highlight that in the context of low-cost FPGA implementation Hummingbird has favourable efficiency and low area and power requirements which can satisfy the resource constraints of Lightweight devices like RFID tags, Smart cards etc.

3.3 New Lightweight Hummingbird Variants

More over it is necessary to maintain confidentiality as much as it is necessary to process the sensitive personal information stored. Thus there is an ever increasing demand for integrating cryptographic functions into embedded applications and improvising them.

This research is part of designing security infrastructure for securing the data which needs to be protected from unauthorised intrusion. A lightweight Hummingbird variant is designed to meet the security needs for the resource constrained environment. The proposed mechanism is proven to be practical in real deployment environment.

In most of the cryptographic algorithms, our primary focus is to protect the Security key which is used in the encryption process. If the key is compromised, the data can be easily deciphered by the adversary. So the main aim is protecting the keys from unauthorized intrusion.

In this paper, a generic construction of high-speed encryption schemes is proposed and analysed. The solution is based on the fact that there exist strong and secure block ciphers, e. g. Hummingbird, and fast but relatively weaker stream ciphers. We then combine a secure block cipher with a fast stream cipher such that the resulting encryption scheme is much stronger than the existing ones. Secure block cipher is used to generate the keys which later are used to encrypt the text using Stream Cipher. We have proposed a new ultra- light weighted scheme for encryption in light weighted devices like RFID, smart cards that can secure sensitive personal information and biological data. This new lightweight Hummingbird variant combines the already existing techniques in such a way that the keys used in

encryption method are generated from the Hummingbird and the encryption is done by the stream cipher techniques. We show the performance analysis as well as our experiment on a Xilinx simulator.

We proposed the two new variants[14] of Hummingbird which uses the keys generated from the output of Hummingbird Scheme. The First Variant is used to secure the data by encrypting the Plaintext using stream cipher and the Second Variant is used to generate the stream of Sub keys using stream cipher which are further used in other stream ciphers for encrypting the data.

We have done the hardware simulation and software implementation to prove the results that the proposed variants are better that that of the existing one in terms of speed, throughput, area and power which is required in resource constrained devices like RFID tags.

3.3.1 First Variant of Lightweight Hummingbird

In this design, a stream cipher is used with Hummingbird cryptography in such a way that the keys used in stream cipher are generated from the ciphertext resulting from Hummingbird cryptography.

3.3.1.1 Security of the Scheme

Stream ciphers are fast but they are not secure. In this scheme, fast stream cipher is used to encrypt the data using key generated by the Hummingbird scheme. This way keys are protected using Hummingbird Scheme and thus it is difficult to achieve the key. On the assumption of the security of Hummingbird, K will never be derived by any attacker no matter what kind of attacks is used. In Hummingbird, the length of the key used is 256-bits so it is highly secure[22].

- **Meet in the middle Attack :** This is a kind of attack of brute force , but the length of key used in Hummingbird cryptography is 256-bits (total number of possible combinations formed is 2^{256}) so it is not possible for adversary to find actual key used.
- **Chosen Ciphertext Attack (the attack to the segment keys) :** As all the stream ciphers that have ciphertext feedback are weak to the chosen ciphertext. For example, if stream cipher was defined by

$$d_i = b_i \oplus F(k, F(k, F(k, d_{i-1}) \oplus d_{i-2}) \oplus d_{i-3})$$

Then the cipher would be weak to chosen ciphertext attack. By choosing $d_{i-1} = d'_{i-1}$ $d_{i-2} = d'_{i-2}$, $d_{i-3} = d'_{i-3}$ different at only one bit, the attacker can ask for the decryption of d_i, d'_i and apply the differential attack. But our stream cipher is defined by

$$d_i = b_i \oplus F(k, F(k, F(k, d_{i-1}) \oplus b_{i-1}) \oplus d_{i-2})$$

where the formula has both ciphertext and plaintext feedback. In such case, if the attacker choose both plaintext and ciphertext, the decrypted plaintext has very small chance to meet the plaintext chosen by the attacker.

3.3.2 Second Variant of Lightweight Hummingbird

In this Scheme, Hummingbird is used to generate the segment keys and the stream cipher is used to generate key streams from the segment keys.

3.3.2.1 Security of the Scheme

In this scheme, the secret *K* of the encryption scheme is protected by Hummingbird cryptography. Therefore, to attack the key is as hard as attack Hummingbird. Second, each segment key generated by Hummingbird is used to encrypt message of a very limited length by the stream cipher.

To resist known plaintext/ciphertext attack: Since in this stream cipher the key stream is generated independent of input/output, the known plaintext/ciphertext attack is just like to know the key stream of the same length. In the stream cipher, both the linear operation and non-linear operation are well combined. The relationships among the output words are extremely complicated. Although the *G* function is relative simple comparing to the traditional shift register-based ciphers, finding the relationship between the output blocks becomes more difficult due to the introduction of *F*. At this stage, we believe that the stream cipher cannot be broken within reasonable time for given key stream of short length.

CHAPTER 4
DESIGN

4.1 Hardware Simulation

Our Algorithmic implementation utilises only basic FPGA builing blocks such as LUTs and flip-flops and does not include any IP core or hardware macros, making it portable across different FPGA platforms without design modifications. We implemented the algorithms on the FPGA platform to confirm its portability using Xilinx tool on Spartan 2E device family. As the effects of FPGA design decisions on performance and area are often specific to individual architectures, it is necessary to further refine the FPGA target before proceeding in the analysis. Like most Xilinx FPGAs, the Spartan-IIE user-programmable gate array[27] is composed of five major configurable elements:

- IOBs provide the interface between the package pins and the internal logic
- CLBs provide the functional elements for constructing most logic
- Dedicated block RAM memories of 4096 bits each
- Clock DLLs for clock-distribution delay compensation and clock domain control
- Versatile multi-level interconnect structure

The Spartan™-IIE 1.8V Field-Programmable Gate Array family gives users high performance, abundant logic resources, and a rich feature set, all at an exceptionally low price. The five-member family offers densities ranging from 50,000 to 300,000 system gates.

The Spartan-IIE[27] family is a superior alternative to mask-programmed ASICs. The FPGA avoids the initial cost, lengthy development cycles, and inherent risk of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary (impossible with ASICs).

Features

- Second generation ASIC replacement technology
  - Densities as high as 6,912 logic cells with up to 300,000 system gates
  - Streamlined features based on Virtex-E architecture
  - Unlimited in-system reprogrammability
  - Very low cost
- System level features
  - SelectRAM + hierarchical memory:
  - 16 bits/LUT distributed RAM
  - Configurable 4K-bit true dual-port block RAM
  - Fast interfaces to external RAM
  - Fully 3.3V PCI compliant to 64 bits at 66 MHz and CardBus compliant
  - Low-power segmented routing architecture
  - Full readback ability for verification/observability
  - Dedicated carry logic for high-speed arithmetic
  - Efficient multiplier support
  - Cascade chain for wide-input functions
  - Abundant registers/latches with enable, set, reset
  - Four dedicated DLLs for advanced clock control
- Versatile I/O and packaging
  - Low cost packages available in all densities
  - Family footprint compatibility in common packages
  - 19 high-performance interface standards, including LVDS and LVPECL
  - Up to 120 differential I/O pairs that can be input, output, or bidirectional
  - Zero hold time simplifies system timing •
- Fully supported by powerful Xilinx ISE development system
  - Fully automatic mapping, placement, and routing
  - Integrated with design entry and verification tools

Device	Logic Cells	Typical System Gate Range (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O	Maximum Differential I/O Pairs	Distributed RAM Bits	Block RAM Bits
XC2S50E	1,728	23,000 - 50,000	16 x 24	384	182	84	24,576	32K
XC2S100E	2,700	37,000 - 100,000	20 x 30	600	202	86	38,400	40K
XC2S150E	3,888	52,000 - 150,000	24 x 36	864	263	114	55,296	48K
XC2S200E	5,292	71,000 - 200,000	28 x 42	1,176	289	120	75,264	56K
XC2S300E	6,912	93,000 - 300,000	32 x 48	1,536	329	120	98,304	64K

Table 4-1 : Spartan-IIE FPGA Family Members

4.2 FPGA Specifications

- The FPGA used in this project has the following specifications:
- Vendor: Xilinx
- Family: Spartan 2E
- Family: XC2S300E
- Package: FT256
- Speed grade: -2Q
- Synthesis Tool: VHDL
- Simulator: Xilinx ISE 6.1i

4.3 Behavioral Simulation

After HDL designing, the code is simulated and its design and functionality is verified using simulation software, e.g. Xilinx ISE or ModelSim simulator. The code is simulated and verified using Mentor Graphics ModelSim 10.3 PE Student Edition. The output is tested for the various inputs. Actual output values are consistent with the expected values.

4.4 Synthesis of Design

Post the behavioral simulation the design is synthesized. During simulation following takes place:

4.4.1 HDL Compilation

The Xilinx ISE tool compiles all the sub-modules of the main module. Design compiler includes tools that synthesis the HDL designs into optimized technology-dependent, gate level designs. If any problem takes place then the syntax of the code must be checked. You need compile the design and verify the correctness of the model before you synthesize it for hardware realization. Once the code is successfully compiled, only then the HDL synthesis will be done.

4.4.2 HDL synthesis

It is the process that generates a gate-level netlist for an IC design that has been defined using a Hardware Description Language (HDL). Synthesis includes reading the HDL source code and optimizing the design from that description. Hardware components like Multiplexers, Adders, Subtractors, Counters, Registers, Latches, Comparators, XORs, Tri-State buffers, Decoders are synthesized from the HDL code. XST[29] generates synthesis report. This report contains the results from the synthesis run, including area and timing estimation.

4.4.3 RTL Schematic

This is a schematic representation of the pre-optimized design shown at the Register Transfer Level (RTL). This representation is in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, and is generated after the HDL synthesis phase of the synthesis process.

4.5 Parameters to compare the performance of the algorithms

- **Area:** This metric represents the area normalized to that of one two-input NAND gate. This ratio is expressed in Gate Equivalent (GE).

- **Cycles per block:** This is the number of clock cycles to compute the plaintext/ciphertext and read out the ciphertext/plaintext.

- **Throughput:** The rate at which new output is produced with respect to time. The number of ciphertext/plaintext bits is multiplied by the operating frequency and divided by the needed cycles. It is expressed in bits-per-second (bps).

- **Power:** It consists of two major components: the static power and the dynamic power. The static power is proportional to the area and the fabrication process (due to leakage current) and is denoted as Pleak. The dynamic power is proportional to the switching activity and is denoted as Pdyn. Both components depend also on the supply voltage.
- **Throughput-to-area ratio:** Measures the hardware resource cost associated with the implementation resulting throughput.

4.6 Software Implementation

The cryptographic algorithms AES, Hummingbird and the various variants of Hummingbird are also implemented in C using core i5 processor with 2.50 Ghz, in order to check their performance and results. We first examined the algorithms after software implementation. After comparing algorithms, we identified a set of efficient algorithms suitable for resource constrained systems. We also compared the performance of these algorithms for different input values. Finally, we simulated our implementation on an Xilinx 6.1i. Our implementation is more faster than the previous results without instruction set architecture extensions or hardware accelerations.

- Platform Used:
    - Windows 7 Home Premium 64-bit
- Compiler:
    - Dev C compiler

CHAPTER 5
RESULTS AND OBSERVATIONS

A summary of our implementation results is presented in Table 5-1, where the area requirements (in slices), throughput and power are given. All experimental results were extracted using ISE Design Suite from Xilinx on a Spartan-IIE family using XC2S300e device on ft256 package with speed grade -6Q. The 32-bit sub keys **ki** (i = 1, 2, 3, 4) are read from an external register based on the value of a round counter through the interfaces **KEY1 (31:0) to KEY4 (31:0).**

The design uses an iterative looping approach with block and key size of 128 bits in case of AES, and 256-bits in case of Hummingbird and lookup table implementation of S-box. This gives low complexity architecture.

An FPGA design flow is used throughout and performance results are presented together with comparison with the previously known best designs. The designs presented all support a 256-bit key in Hummingbird and 128 bit key in AES. Xilinx ISE version 6.1i was used for the design flow and the results quoted after Synthesis. The new designs were coded in VHDL.

Mentor Graphics' ModelSim 10.3 PE student edition is also used to verify and validate the results of the algorithms.

5.1 Results of ModelSim Simulation

ModelSim executes to create the libraries, compile the source files and testbench and display the results in the Wave window. The test bench are used to generate the results of VHDL code.

A Test Bench in VHDL is code written in VHDL that provides stimulus for individual modules (also written in VHDL). Individual modules are instantiated by a single line of code showing the port connections to the module. The code for the module, itself, does not show up in the Test Bench code, just the port link. Test Benches look like regular modules but they have no port connections. They may also contain code that is not synthesizable into a real device. It allows you to provide a documented, repeatable set of stimuli that is portable across different simulators. A test bench can be as simple as a file with clock and input data or a more complicated file that includes error checking, file input and output, and conditional testing. It has Clock and reset signals also alongwith the input and output signals.

In almost any testbench, a clock signal is usually required in order to synchronise stimulus signals within the testbench. And reset is used to reset the values of the signals. Almost every test bench uses clock and reset signals.

The following figures are the test-bench simulation of the different variants of the proposed Lightweight Encryption scheme using Mentor graphics ModelSim 10.3 PE simulator. It shows how the cipher generated from the scheme for the different inputs given. The cipher is generated when the clock signal is high and on reset signal, the values of the signals are reset ie set to 0.

In ModelSim simulator, we can change the radix also ie we can view the output in different formats like ASCII , Decimal , Octal and hexadecimal.

The output of the various Encryption scheme are as shown in the following snapshots.

5.2 XPower Analyzer Results

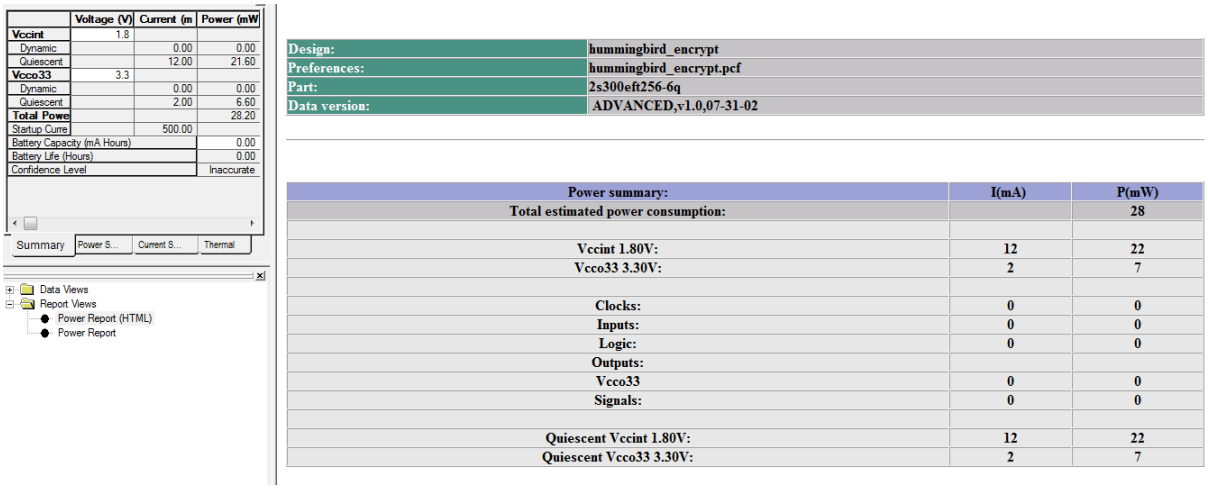
XPower Analyzer[30] is a tool dedicated to power analysis of post-implemented place and routed designs. It provides a comprehensive graphical user interface (GUI) that allows a detailed analysis of the power consumed and offers as well thermal information under operating conditions.

Different views are available to identify the power consumed either by type of blocks (Clock trees, Logic, signals, I/O's, Hard IPs such as BRAMs or DSP blocks) or over the design hierarchy.

Both of these views enable users to perform a detailed power analysis and give a very efficient way to locate the blocks or parts of the design which are the hungriest in terms of power, thereby giving an easy path to power optimization.

XPower Analyzer leverages device knowledge and design data to deliver accurate estimate of device power and individual net power utilization.

The following are the results of xpower analyzer of the proposed lightweight Encryption scheme.



	Voltage (V)	Current (m	Power (mW
Vccint	1.8		
Dynamic		0.00	0.00
Quiescent		12.00	21.60
Vcco33	3.3		
Dynamic		0.00	0.00
Quiescent		2.00	6.60
Total Powe			28.20
Startup Curre		500.00	
Battery Capacity (mA Hours)			0.00
Battery Life (Hours)			0.00
Confidence Level			Inaccurate

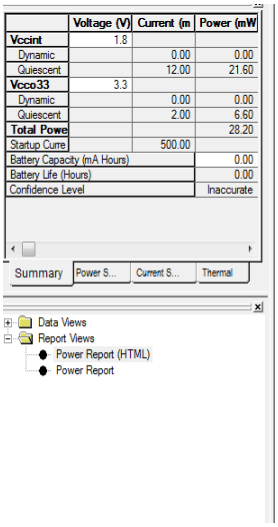
Summary | Power S... | Current S... | Thermal

Data Views
Report Views
Power Report (HTML)
Power Report

Design:	hummingbird_encrypt
Preferences:	hummingbird_encrypt.pcf
Part:	2s300eft256-6q
Data version:	ADVANCED,v1.0,07-31-02

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		28
Vccint 1.80V:	12	22
Vcco33 3.30V:	2	7
Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	12	22
Quiescent Vcco33 3.30V:	2	7

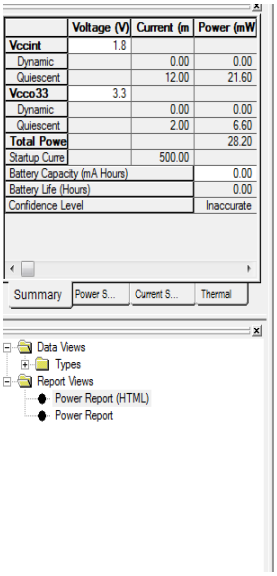
Figure 5-6: Power analysis of Hummingbird Cryptography



Design:	first_variant_encrypt
Preferences:	first_variant_encrypt.pcf
Part:	2s300eft256-6q
Data version:	ADVANCED,v1.0,07-31-02

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		7
Vccint 1.80V:	0	0
Vcco33 3.30V:	2	7
Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vcco33 3.30V:	2	7

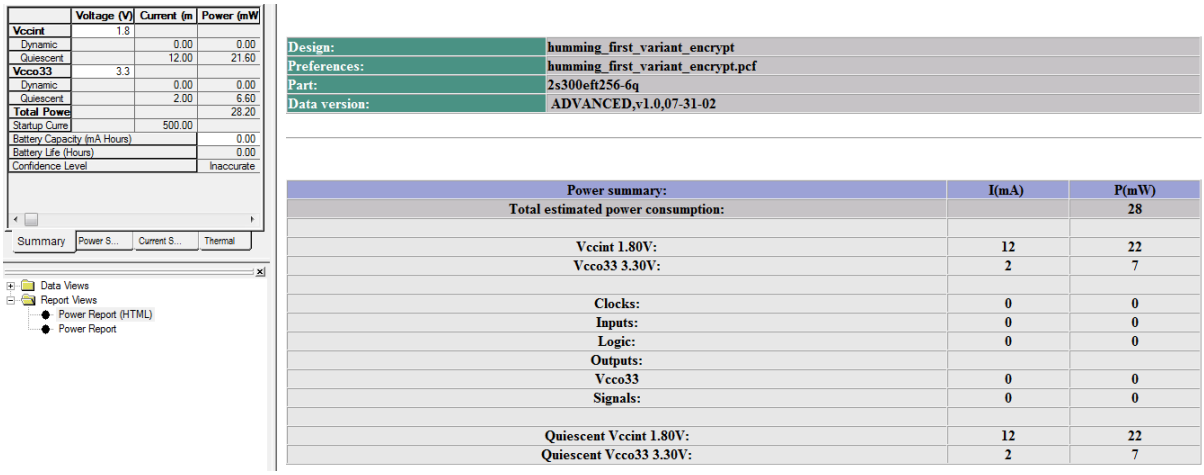
Figure 5-7 : Power analysis of First variant stream Cipher



Design:	second_variant_stream_cipher_encrypt
Preferences:	second_variant_stream_cipher_encrypt.pcf
Part:	2s300eft256-6q
Data version:	ADVANCED,v1.0,07-31-02

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		28
Vccint 1.80V:	12	22
Vcco33 3.30V:	2	7
Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	12	22
Quiescent Vcco33 3.30V:	2	7

Figure 5-8: Power analysis of Second variant stream Cipher



	Voltage (V)	Current (m	Power (mW
Vccint	1.8		
Dynamic		0.00	0.00
Quiescent		12.00	21.60
Vcco33	3.3		
Dynamic		0.00	0.00
Quiescent		2.00	6.60
Total Powe			28.20
Startup Curre		500.00	
Battery Capacity (mA Hours)			0.00
Battery Life (Hours)			0.00
Confidence Level			Inaccurate

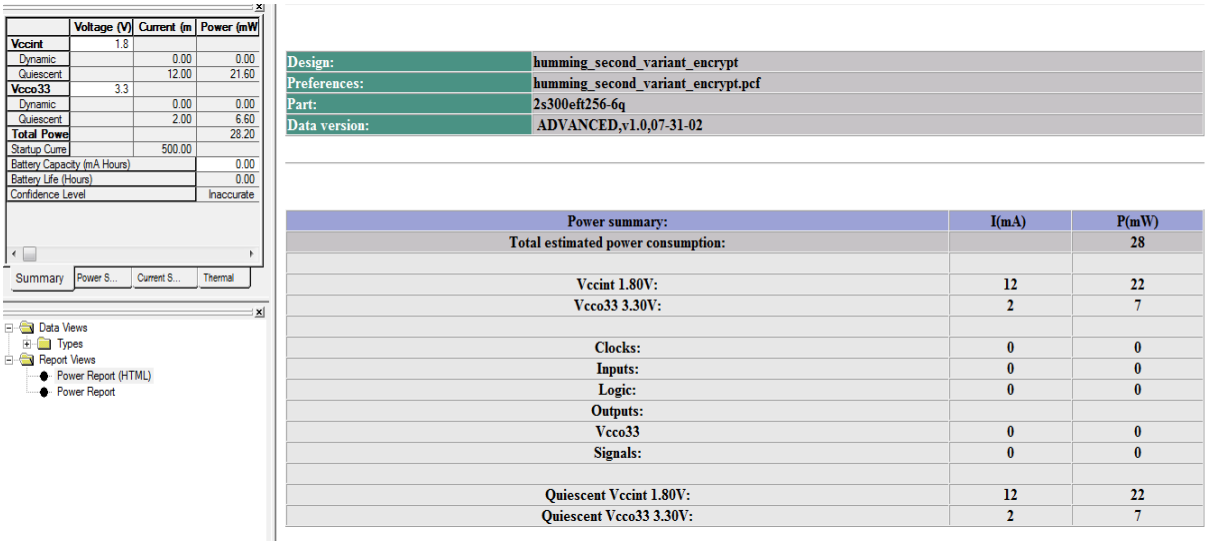
Summary | Power S... | Current S... | Thermal

Data Views
Report Views
Power Report (HTML)
Power Report

Design:	humming_first_variant_encrypt
Preferences:	humming_first_variant_encrypt.pcf
Part:	2s300eft256-6q
Data version:	ADVANCED,v1.0,07-31-02

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		28
Vccint 1.80V:	12	22
Vcco33 3.30V:	2	7
Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	12	22
Quiescent Vcco33 3.30V:	2	7

Figure 5-9: Power analysis of Hummingbird with first variant



	Voltage (V)	Current (m	Power (mW
Vccint	1.8		
Dynamic		0.00	0.00
Quiescent		12.00	21.60
Vcco33	3.3		
Dynamic		0.00	0.00
Quiescent		2.00	6.60
Total Powe			28.20
Startup Curre		500.00	
Battery Capacity (mA Hours)			0.00
Battery Life (Hours)			0.00
Confidence Level			Inaccurate

Summary | Power S... | Current S... | Thermal

Data Views
Types
Report Views
Power Report (HTML)
Power Report

Design:	humming_second_variant_encrypt
Preferences:	humming_second_variant_encrypt.pcf
Part:	2s300eft256-6q
Data version:	ADVANCED,v1.0,07-31-02

Power summary:	I(mA)	P(mW)
Total estimated power consumption:		28
Vccint 1.80V:	12	22
Vcco33 3.30V:	2	7
Clocks:	0	0
Inputs:	0	0
Logic:	0	0
Outputs:		
Vcco33	0	0
Signals:	0	0
Quiescent Vccint 1.80V:	12	22
Quiescent Vcco33 3.30V:	2	7

Figure 5-10: Power Analysis of Hummingbird with second variant

5.3 Xilinx Synthesis Results

The ISE Design Suite includes Xilinx Synthesis Technology (XST) allowing synthesis of HDL designs to create Xilinx specific netlist files. With specially optimized algorithms to leverage the advanced architectures of the Xilinx FPGA families, XST offers designers a low-

cost design solution to achieve optimal design results. Synthesis Report gives the Device utilisation summary and the timing Report.

Integrated within the ISE Project Navigator, XST provides support for mixed-language Verilog and VHDL designs. This flexibility allows designers to mix the best possible design source code for any particular project. XST helps designers solve their toughest design challenges[29]:

- **Performance**: XST incorporates next-generation physical synthesis optimizations through techniques such as register balancing, global optimization, timing-driven synthesis, and logic optimization to improve the quality of results.

- **Reduced Runtime and Design Preservation:** With its tight integration within the ISE SmartCompile Technology, XST helps maintain successful results to dramatically reduce runtimes during subsequent re-implementations.

- **Power Reduction**: Power optimizations in XST provide power-aware logic optimizations for macro processing on blocks such as multipliers, adders and BRAMs.

- **Ease-of-use**: XST provides designers with additional features to better explore their synthesis results. Integrated RTL and Technology Viewers allow designers to view their RTL netlist to better visualize how XST inferred the components of their design to help identify problems and improve their design early in the process.

	# of Slices	**Throughput (in MBps)**	**Gate Equivalent**	**Power (in mW)**
AES	1486	12.01	23,626	-
Hummingbird	162	44.8	2072	28
First Variant Stream Cipher	245	68.8	5648	7

Second Variant Stream Cipher	1395	131.04	10,056	30
Hummingbird with First Variant	174	59.23	3086	28
Hummingbird with Second Variant	2771	13.2	12,345	28

Table 5-1: Comparison between AES, Hummingbird and different variants of proposed scheme

Above is the table which shows the performance of different cryptographic primitives on Xilinx Spartan-IIE family using XC2S300e device. Table clearly shows that Hummingbird algorithm takes less Number of clock cycles, less area, less power and high throughput as compare to AES method to encrypt the bits. As Hummingbird algorithm utilises very limited resources so it can work in resource constrained environment like RFID tags.

5.4 Software Results

We implemented the Hummingbird and its various variants on 2.50 GHz Intel Core i5 processor. The average encryption speed for the AES, Hummingbird and its variants is given in the following table.

Encryption Scheme	Average Encryption speed (KB/sec)
AES	8.91
Hummingbird	9
Hummingbird First Variant	9.2
Hummingbird Second Variant	11.97

Table 5-2: Average Encryption speed of the various Encryption Schemes on software platforms

Above is the table which shows the performance of the existing schemes and the proposed variants, on the software platform.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

The proposed cipher is appropriate for RFID applications. In this we can combine strong cipher with a weak cipher such that the combined encryption scheme is a much stronger as the former ones. We also gave some concrete variant schemes under this principle.

Our design is based on the observation on the available attacks to symmetric key ciphers. When we talk about a cipher, it may be weak under some powerful attacks, i.e., the chosen plaintext/ciphertext attacks, which are targeting to drive the secret key. Though the encryption scheme ie Hummingbird itself is strong but it may weak under some powerful attacks. But, in this proposed scheme such attacks make no sense towards deriving segment keys. Therefore, many "weak" ciphers become "strong" when being used under our principle.

In this project a Lightweight Encryption scheme is designed and simulated using Xilinx ISE using VHDL as a synthesis tool. The output of the core is analyzed and verified on the test-bench, and compared with the actual values of the output obtained from Software implementation.

Although our schemes are very fast, we would like to point out that the encryption schemes constructed under our principle have security advantage when they are used to encrypt large amount of data. AES are not suitable since a commercial implementation of AES typically requires 20,000 – 30,000 gates. This is far more than the number of gates on an entire low cost label. So in this, we present a novel ultra-lightweight cryptographic algorithm, Hummingbird, which is a combination of block cipher and stream cipher. This algorithm is mainly designed to work in the extremely resource constrained environment. The hybrid structure adopted in Hummingbird can provide the designed security with small block size which is expected to meet the stringent response time and power consumption requirements in

a large variety of embedded applications. We show that Hummingbird seems to be resistant to the most common attacks to block ciphers and stream ciphers including birthday attacks, differential and linear cryptanalysis, structure attacks, algebraic attacks, cube attacks, etc.

When compared to the block cipher AES implemented on similar platforms, our experimental results sshow that after a system initialization procedure Hummingbird can achieve up to 54.74 and 4.1 times faster throughput for a size-optimized and a speed-optimized implementations, respectively.

On comparing the performance of the proposed encryption schemes implemented on software as well as the hardware platform, clearly shows that the performance of the schemes is far better on the hardware platform as the hardware encryption doesn't require system resources to perform the encryption/decryption process and therefore allows for better system performance.

6.2 Future Scope

Although the proposed scheme is efficient for resource constrained environment, but it can further be optimised. As we are using an iterative approach in designing the scheme, if the proposed scheme is designed using pipelining approach, efficiency can further be improved.

References

[1] Michael Grimaila , “RFID Security Concerns”, ISSA , The Global Voice of Information Security,feb 2007 .

[2] A. Juels, R.L. Rivest, and M. Szydlo. The Blocker Tag: Selective Blocking of RFID Tags for Consumer Privacy. 8th ACM Conference on Computer and Communications Security, pages 103-111. ACM Press, 2003.

[3] Auto-ID Center (now EPCglobal, Inc.). Auto-ID Reader Protocol 1.0 (Work in progress). http://www.epcglobalinc.org/WD-reader-protocol-200309051/4861_0.htm.

[4] Smart Border Alliance RFID Feasibility Study Final Report , “ RFID SECURITY AND PRIVACY WHITE PAPER”

[5] S. Sarma, S. Weis, and D. Engels, “RFID Systems and Security and Privacy Implications”, Cryptographic Hardware and Embedded Systems - CHES 2002, Springer, 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002.

[6] Monika Sharma1 and Dr. P. C. Agrawal, “A RESEARCH SURVEY: RFID SECURITY & PRIVACY ISSUE”, ICCSEA-2013

[7] Bo Sun , Chung-Chih Li, Kui Wu , Yang Xiao , “A lightweight secure protocol for wireless sensor networks” , ScienceDirect , Computer Communications 29 (2006) 2556–2568

[8] *Samantha Donovan, Peter Drabwell and RaeHarbird1, Nortel Networks , “*VPNs and Lightweight Clients”, Elsevier , Information Security Technical Report, Vol 6, No. 1 (2000)

[9] Sergey Panasenko and Sergey Smagin , “Lightweight Cryptography: Underlying Principles and Approaches” , *International Journal of Computer Theory and Engineering, Vol. 3, No. 4, August 2011*

[10] Feng Bao, Robert H. Deng, “Light-Weight Encryption schemes for Multimedia Data and High-Speed Networks “,IEEE 2007

[11] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems", Journal of Cryptology, Vol. 4, No. 1, pp. 3-72, 1991, Springer.

[12] E. Biham, R. Anderson and L. Knudsen, "Serpent: a proposal for the advanced Encryption standard",http://www.cl.cam.ac.uk/~rja14/serpent.html.SSA

[13]http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf

[14] Willi Meier, "Crytanalysis of Stream Ciphers", SASC – The State of the Art of Stream Ciphers, Brugge, Belgium.

[15] Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A, Robshaw MJB, Seurin Y, Vikkelsoe C. PRESENT: An Ultra-Lightweight Block Cipher, Cryptographic Hardware and Embedded Systems – CHES 2007, LNCS. Springer. 4727; 2007. p. 450–466.

[16] Engels D, Fan X, Gong G, Hu H, Smith E. "Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices", Financial Cryptography and Data Security Lecture Notes in Computer Science Volume 6054, Springer 2010

[17] Alina Olteanu, Yang Xiao, Fei Hu, Bo Sun, "A Lightweight Block Cipher Based on a Multiple Recursive Generator", IEEE 2008

[18] Axel Poschmann, Gregor Leander, Kai Schramm, and Christof Paar, "New Light-Weight Crypto Algorithms for RFID", IEEE 2007.

[19] Gregor Leander and François-Xavier Standaert (Ed.), ECRYPT Workshop on Lightweight Cryptography, November 28-29, 2011

[20] "ADVANCED ENCRYPTION STANDARD (AES)", Federal Information Processing Standards Publication 197, November 26, 2001

[21] Orr Dunkelman , "Related Key Attacks", Department of Computer Science, University of Haifa Faculty of Mathematics and Computer Science Weizmann Institute of Science June 2nd, 2011

[22] Saeed Bahrami and Majid Naderi, "Image Encryption Using a Lightweight Stream Encryption Algorithm" , Advances in Multimedia Volume 2012

[23] Peter Rombouts,"Lightweight Cryptography ", ECRYPT II Closing Event Cryptography for 2020 Tenerife January 23rd, 2013

[24] http://www.ecrypt.eu.org/stream

[25] Paris Kitsos , Nicolas Sklavos , Maria Parousi , Athanassios N. Skodras , “A comparative study of hardware architectures for lightweight block ciphers”, Computers and Electrical Engineering 38 (2012) 148–160 , Science Direct

[26] J.D. Hietala , “Hardware versus Software , A Usability Comparison of Software-Based Encryption with Seagate DriveTrust Hardware-Based Encryption” ,A SANS Whitepaper – September 2007s

[27] Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information on Xilinx, November 15,2001

[28] Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, Pierre-Alain Fouque , “Another Look at Complementation Properties”, Fast Software Encryption Lecture Notes in Computer Science Volume 6147, Springer 2010, pp 347-364.

[29] http://www.xilinx.com/tools/xst.htm

[30] http://www.xilinx.com/products/design_tools/logic_design/verification/xpower_an.htm

[31]http://www.academia.edu/2197073/Classification_of_RFID_Threats_based_on_Security_Principles