

Electronics and Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

Author: Lewis Smith

December 8, 2017

Low Power Hardware Accelerated Internet of Things Cryptography

Project Supervisor: Mark Zwolinski
Second Examiner:

A project progress report submitted for the award of MEng
Electronic Systems with Computer Systems

DRAFT

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

Abstract	1
Contents	2
1 Introduction	3
2 Background Research and Literature	5
2.1 Internet of Things	5
2.2 Cryptography	5
2.2.1 Asymmetric Key	6
2.2.2 Symmetric Key	6
2.2.3 Decisions	7
2.3 Conventional Algorithms	7
2.3.1 Standardization	8
2.3.2 Other Algorithms	8
2.3.3 Decisions	9
2.4 Lightweight Algorithms	9
2.4.1 SIMON & SPECK	9
2.4.2 Other Algorithms	10
2.4.3 Decisions	10
3 Progress	11
3.1 Hosted C	11
3.2 System Verilog	12
4 Future Plan	13
References	14
A DES	15
B AES	16
C Blowfish	17
D SIMON & SPECK	18
E Other Algorithms	19
F Table of FPGA and Software Data	20

Chapter 1

Introduction

Over the last few years there has been a shift in type of [devices](#) connected to the internet from just servers, PC's and later smartphones, to small embedded processors that can control many [devices](#). The idea of connecting such [devices](#) to internet has been dubbed 'Internet of Things' or 'IoT' and has the aim to make our lives simpler. These IoT [devices](#) can range from industrial applications, such as automated factories, to public infrastructure, like a smart electrical grid, or more homely appliances in a smart connected home. Thus due to the wide range of products that a connected IoT device can be applied to improve the efficiency and/or usefulness, it has been predicted that billions of [devices](#) will be in use by [\[insert year/reference\]](#). This also means that the complexity of the [devices](#) varies greatly with simple light switches and even kettles being given the IoT treatment[\[insert reference\]](#), but at the other end of the scale a network of connected self-driving cars is being considered[\[insert reference\]](#).

The one thing that all of these [devices](#) have in common though is that they need to be secure as they communicate sensitive and private data through an open channel on the internet between the user and the device. Hence, to keep the potential adversaries from accessing the data and possibly controlling numerous connected [devices](#), maliciously or not, an encryption algorithm can be used with only the specified users having access to the decryption key. There are many encryption algorithms that perform this function and most can be implemented in both software and dedicated hardware such as a Application Specific Integrated Circuit (ASIC) or a Field Programmable Gate Array (FPGA). As a majority of IoT [devices](#) are implemented on small embedded processors which have limited resources, the hardware option might possibly be a better solution for IoT [devices](#). However, due to the fact that most IoT devices are always on, power consumption is a very important factor when considering options for adding hardware accelerated encryption and for battery powered devices it is often more critical than the actual encryption.

The goals of this project are to explore various encryption algorithms and compare their performance based on data throughput, accuracy, security and power consumption when implemented in software and hardware. To evaluate these parameters the same algorithms can be coded in C or C++ for the software versions and a Hardware Development Language (HDL) such as System Verilog can be first simulated in ModelSim[\[insert reference\]](#), before programming a FPGA for the hardware version.

These comparisons can then be used to match the algorithms to the appropriate IoT device as they all have different requirements for relative security level and power consumption, as for example a light switch does not necessarily need to be protected from the same level of attack as a set of digital locks or private data storage. In order for the hardware to work with IoT devices it will also need a communication protocol like I^2C or SPI to work with embedded processors, and possibly Ethernet or WiFi to act as the gateway to internet for the device. Some of these protocols are available on FPGA development boards but can be implemented in System Verilog code.

The scope of this project is therefore to develop encryption algorithms in System Verilog for a FPGA and compare their performance to the equivalent software before applying the hardware to actual IoT devices. Also if more time is available then a layout, based on the System Verilog synthesis, could be developed for ASIC fabrication in the future which could increase efficiency of the hardware due to the fact that only the required circuitry would be present on the chip.

Chapter 2

Background Research and Literature

2.1 Internet of Things

2.2 Cryptography

The primary objective of cryptography is to convert, or encrypt, a readable message known as plaintext into an unreadable form, ciphertext, so that adversaries cannot read the contents, but over the years the scope of cryptography has widened[insert reference]. Throughout history encryption has been used allow people and groups to exchange secret messages, especially in times of war. Since the early transposition and substitution ciphers, where each character in a message are rearranged and replaced by others a certain number further down the alphabet respectively, encryption has evolved to include techniques for identity authentication, integrity checks and much more[insert reference]. Cryptography is therefore the study of encryption and these techniques, its counterpart: the study of breaking the encryption to find the original message, is known as cryptanalysis. Eventually, for most encryption techniques a weakness is found, and subsequently exploited, so more complex techniques are conceived and with the invention of computers the complexity of the algorithms has increased greatly. However, the computer power is also available for cryptanalysis so the cycle of continuous improvement of the algorithms hasn't stopped.

In cryptography there are two main concepts that the algorithms are based on: symmetric and asymmetric keys[insert reference]. In asymmetric key cryptography a unique key is used to encrypt data and different, but a related key, is used to decrypt it. The relationship between the two keys is often defined by maths problem that is very difficult to solve which is the basis of the encryption[insert reference]. On the other hand, symmetric key cryptography uses the same key for both encryption and decryption, hence symmetric, with the security usually provided by a combination of simple logic operations[insert reference].

2.2.1 Asymmetric Key

Asymmetric key cryptography can be referred to as public key due to the fact that one of the related keys can be publicly available without compromising the security of the encrypted data. This is because the keys are usually generated based on mathematical problems that have no solution or the solution is impossible for a computer to solve efficiently, such that solving it takes longer than an exhaustive key search[insert reference]. There are many problems that fit this criteria but the most popular in use today are the integer factorization and elliptic curve problems used by the RSA[insert reference] and the family of Elliptic-curve cryptography (ECC) techniques[insert reference] respectively. Public key cryptography can be used in two different modes as if data is encrypted with the intended recipients public key only they can decrypt it with their private key, thus encryption. However, if a private key is used for encryption then using the public key to decrypt it ensures the senders identity, authentication[insert reference].

2.2.2 Symmetric Key

Similar to the symmetric/private comparison symmetric key cryptography is also known as private key as in order to keep the encrypted data secure the key used must be kept secret. There are two main types of private key algorithms that operate on the plaintext differently: block ciphers which uses a fixed number of bits, block; or stream ciphers which encrypts data bit by bit[insert reference].

Modern block ciphers are based on Claude Shannons work on product ciphers in [insert reference], in which he suggested that iterating a cipher for multiple rounds, with subkeys, improves the security. Hence, the cipher to be iterated didn't need to be complex operations and simple logic operations such as XOR, substitution or permutation of the plaintext could be used[insert reference]. The base cipher that is iterated is known as the round function and it takes as an input a block of plaintext and a subkey, which is generated from the main key by a separate key expansion function, and outputs a block of ciphertext. The round functions are mostly designed using either a Feistel network (F network) or a Substitution Permutation network (SP network)[insert reference].

The Fesitel network was named after physicist Horst Feistel who was a integral part of the team at IBM that developed the early block cipher Lucifer, which of course used a Feistel network[insert reference]. The F network works by splitting the input plaintext into two equal words, known as the left (MSB) and right (LSB) words. The round function is then applied to the right word before the result is XORed with the left word and then the words are swapped over and iterated as in ??, with the ciphertext being equal to (R_{n+1}, L_{n+1}) where n is the number of rounds iterated. The advantage of using a F network is that decryption is just applying the same algorithm but with the sub keys in reverse as in [insert equation], with the ciphertext (R_{n+1}, L_{n+1}) as the input and the plaintext (L_0, R_0) returned.

$$L_{i+1} = R_i \tag{2.1}$$

$$R_{i+1} = L_i \oplus F(R_i, K_i) \tag{2.2}$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

$$R_i = R_{i+1}$$

$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

The other structure used in block ciphers is Substitution Permutation network and unlike F networks an SP network operates on the whole plaintext block using substitution, in the form of a S-box, to provide Shannon's confusion and permutation, a P-box, for diffusion[insert reference]. Individually, these operations aren't particularly strong as a S-box and a P-box can be thought of as simple substitution and transposition ciphers respectively. Although if combined in a SP network over multiple rounds the security can be very strong due to Shannon's confusion and diffusion properties being satisfied. The S-boxes usually take in a certain number of bits and outputs the same number of bits but of a different value. P-boxes are then used to spread the bits around such that the output of the S-boxes are used by as many S-boxes in the next round. After the S-boxes and P-boxes and before the next round occurs the block is XORed with the round key so the round equation is [insert equation]. Decryption is achieved using inverted S-boxes and P-boxes and the round keys in reverse order which means that different hardware or operations are needed.

$$B_{i+1} = F(B_i) \oplus K_i$$

$$B_i = F'(B_{i+1}) \oplus K_i$$

[Insert Stream Ciphers]

[Insert Modes of Operation]

2.2.3 Decisions

Due to the fact that asymmetric key algorithms are hard to solve they require complex hardware or software to implement which is undesirable for this project. Also, with the exception of ECC the key sizes needed for the security can be very large so with the limited IO pins available on FPGAs they could prove difficult to program. On the other hand, many private key algorithms are designed to be efficient in hardware especially Feistel networks as an inverted round function isn't required. While a stream cipher can be useful to encrypt serial data that will most likely be the source, although the modes of operation available for block ciphers provide more flexible functionality including stream cipher modes. Therefore, the algorithm chosen for this project will most likely be a block cipher with a Feistel network.

2.3 Conventional Algorithms

There are many block ciphers that are considered very secure and therefore popular, they include: DES, AES, Blowfish[insert reference]. DES operates on a block of 64 bits for 16 rounds using a key length of 64 bits but it has an effective key length of 56 bits as 8 bits were used for parity[insert reference]. AES, an upgrade to DES, is far

more secure as uses a 128 bit block and has the flexibility of using three different key lengths: 128, 192 and 256. The number of rounds that AES iterates depends on the key length with 10 rounds used for a 128 bit key, 12 for 192, and 14 for the largest key[insert reference]. Blowfish, like DES, operates on a 64 bit block and iterates for 16 rounds, but it can use a variable key length in the range 32 to 448 bits[insert reference].

2.3.1 Standardization

DES, which stands for Data Encryption Standard, is one the earliest block ciphers used in the computer age and it was developed by IBM in the 1970s based on their earlier cipher Lucifer[insert reference]. As with Lucifer it was designed around a Feistel network but the round function used also has a SP network structure to it[insert reference], but the S-boxes aren't a one-to-one function but rather output 4 bits from a 6 bit input. It has the name Data Encryption Standard as it was accepted as the standard encryption algorithm by the US National Bureau of Standards (NBS), now the National Institute of Standards and Technology (NIST), in 1977 after it was altered by the National Security Agency (NSA), which caused some controversy[insert reference].

[Insert Attacks of DES]

DES was used for about two decades but in the 1990s several successful attacks proved its weakness so in 1997 NIST started a selection process to find its replacement. Due the controversy of the NSAs involvement and comments from the cryptography community the selection process was as transparent as possible[insert reference]. Many algorithms were submitted as candidates for the standard but the finalists were: MARS, RC6, Rijndael, Serpent, and Twofish[insert reference]. It took three years to decide on the algorithm to be set as the standard which was announced as Rijndael in 2000 and the standard was et in 2001, with the 128, 192, 256 bit keys being used in the standard[insert reference]. Unlike DES, AES uses a SP network as it is efficient, in time, in both hardware and software. The round function treats the block as a 4x4 byte matrix and performs multiple steps on the data: sub bytes; shift rows; mix columns and add round key[insert reference]. The sub bytes uses S-boxes with byte inputs and outputs another byte based on a lookup table. The shift row step then shifts each of the rows in the matrix by the same number of bytes as the row number. The mix column step is the most complicated as it involves matrix multiplication in the finite field $GF(2^8)$, but is commonly implemented in lookup tables. The add round key step is the simplest as it is just XORing the block with the key generated by the key schedule.

[Insert Attacks and FPGA implementations of AES]

2.3.2 Other Algorithms

The US standardized algorithms quickly became very popular and can be considered the unofficial global standard[insert reference]. Although, there are many other algorithms that are considered secure and are commonly used. One of these algorithms,

TripleDES, is actually based around DES which increases the security by encrypting the plaintext three times with separate keys making the effective key length of 168 bits. Other algorithms might be used because there is still some distrust of the NSA's involvement in the algorithms and they are more open source. This is the case of the Blowfish algorithm as it is unpatented and can therefore be used in any product without legal consequence. Blowfish was designed by Bruce Schneier in the early 1990s as he, and many others, noticed the insecurity of DES particularly with the 56 bit key length making a brute force attack more plausible[insert reference]. The design of the algorithm is based around a Feistel network with, similar to DES, the round function using S-boxes. The sub keys and S-box lookup tables are generated using the hexadecimal digits of pi which are provided by the designers[insert reference].

[Insert Attacks and FPGA implementations of Blowfish]

2.3.3 Decisions

2.4 Lightweight Algorithms

After deciding that the conventional algorithms might not be suitable for the low power devices targeted by this project, some more lightweight algorithms were found including: PRESENT, PRINCE and the SIMON and SPECK algorithms[insert reference]. However, as IoT is an emerging technology and is the main reason for lightweight cryptography there isn't a standard set by NIST or any other organisation[insert reference].

2.4.1 SIMON & SPECK

The SIMON and SPECK family of algorithms are the lightweight techniques proposed by the NSA that were designed to perform well in both software and hardware while still being secure; and to be flexible in terms of block and key size[insert reference], listed in Table 2.1. The different algorithms are similar but SIMON was optimised for hardware implementations and SPECK for software. As with AES the number of rounds iterated depends on the key size but as the block size varies as well it also has an effect as shown in Table 2.1. The structure of both algorithms is a Feistel network and thus it works on words of n bits, where $2n$ is the block size, and with a key of mn bits. This lends it self to the naming format of SIMON or SPECK $2n/mn$, which means, for example, SIMON48/72 has a word size of 24 and uses 3 words for the key[insert reference].

Block Size	Key Size	n	m	SIMON Rounds	SPECK Rounds
32	64	16	4	32	22
48	72	24	3	36	22
48	96	24	4	36	23
64	96	32	3	42	26
64	128	32	4	44	27
96	96	48	2	52	28
96	144	48	3	54	29
128	128	64	2	68	32
128	192	64	3	69	33
128	256	64	4	72	34

Table 2.1: A table of the modes of operation for the SIMON & SPECK Algorithms. Adapted from [\[insert reference\]](#).

As SIMON was designed primarily for hardware it only makes use of XOR (\oplus), AND ($\&$) and circular rotate operations ($R^j[x]$) on the n bit wide words. For the rotate operation the word x is rotated by j bits to the left or right if j is negative. The encryption and decryption functions take the Feistel network described in [\[insert equations\]](#) and the round function is [\[insert equation\]](#).

$$F(x) = (R^1[x] \& R^8[x]) \oplus R^2[x]$$

SPECK on the other hand, being optimised for software implementations uses XOR (\oplus), modulo 2^n addition ($+$) and circular rotate operations ($R^j[x]$), with the circular rotate being equivalent to what is used in SIMON. The encryption and decryption functions take a slightly different form to the basic Feistel network equations [\[insert equations\]](#) and are shown in [\[insert equation\]](#) where $\alpha = 7$ and $\beta = 2$ if $n = 16$, but $\alpha = 8$ and $\beta = 3$ otherwise.

$$L_{i+1} = (R^{-\alpha}[L_i] + R_i) \oplus K_i$$

$$R_{i+1} = R^\beta[R_i] \oplus (R^{-\alpha}[L_i] + R_i) \oplus K_i = R^\beta[R_i] \oplus L_{i+1}$$

[\[Insert Attacks and FPGA implementations of SIMON & SPECK\]](#)

2.4.2 Other Algorithms

2.4.3 Decisions

Chapter 3

Progress

Based on the research explored in chapter 2 I chose the SIMON algorithm from the NSA, mainly because of its flexibility in security levels, with different key lengths, that could be applied to the different devices explored in section 2.1. This means that multiple algorithms don't need to be developed and I could concentrate on making my code as efficient as possible. As the aim of this project is to compare how an algorithm performs in hardware and software and due to the fact that there isn't a standard library or program for SIMON, a software version was required for benchmarking. Starting in software also increased by familiarity of the algorithm and provide a good starting point for System Verilog development.

For all versions of the algorithm (table 2.1) efficiency, in terms of power consumption and resource use, is very important but time efficiency is not an initial priority. All versions of this algorithm were developed not only with the description but also the pseudocode provided in [insert reference]. That document also has a set of test vectors for all modes that define the ciphertext that the algorithm should produce from the given key and plaintext[insert reference]. Due to the Feistel network structure of this algorithm encryption can be done in parallel to the key expansion but for decryption requires the keys to be pre expanded. For this reason, and because some modes of operation only require encryption, two variants of the algorithm were developed: one for just encryption and one full functionality.

3.1 Hosted C

As there are ten versions of this algorithm that all differ slightly with the block key sizes they could all be developed individually. However, as ten versions of the similar functions would be required I felt it was better to have one version that is flexible. However, checking which mode the program is before most operations would be very inefficient and as the program would rarely changing mode during runtime the decisions could be made at compile time using preprocessor macros as in [insert listing][insert reference]. The macros were used to define the variable type used for the words based on the values for n with the `uint n .t` type. Also the value of m was used with macro if statements in the key expansion functions where extra code is required if $m = 4$.

[Insert Listings, test vectors and results]

3.2 System Verilog

After the Hosted C version was working with the test vectors the System Verilog development began. Similar to the function used in the software developed in section 3.1 modules were created and tested with individual testbenches before being combined in the top level control modules. Most of the modules, including the rotate operations, were done in combination logic blocks for initial simplicity to ensure correct simulations. While this might work in simulations, it could be unreliable and inefficient when implemented in an FPGA. Also parameters were used so the different modes could use the same code but the only testing done so far is with the SIMON32/64 version.

[Insert Listings, test vectors, simulations and synthesis]

Chapter 4

Future Plan

To continue this project the System Verilog code needs to be improved with more sequential operations to improve reliability and efficiency. Serializing the algorithm at the byte level could also be explored which could have some interesting results similar to the bit serial version presented in [\[insert reference\]](#). Work will also have to be done to implement the System Verilog top module onto an FPGA and interface it with communication module, that could be provided by the FPGA development board used or from an OpenCores.org project [\[insert reference\]](#). When the FPGA is functioning correctly tests will need to be developed to determine the throughput in hardware and software, preferably operating at similar clock frequencies to ensure the results will be as comparable as possible. The hosted C program, section 3.1, should also be adapted to be programmed onto an 8 or 16 bit microcontroller and then the same parameters will be tested and compared.

Bibliography

Appendix A

DES

Appendix B

AES

Appendix C

Blowfish

Appendix D

SIMON & SPECK

Appendix E

Other Algorithms

Appendix F

Table of FPGA and Software Data