Supporting Privacy in RFID Systems

Thomas Hjorth

Supervisor: Christian D Jensen
Supervisor: Jan Madsen

IMM, DTU, Lyngby, Denmark
M.Sc. project, no. 04/87
December 14, 2004

Abstract

To improve on its supply chain management (SCM) one of US's largest chain of supermarkets, Wal-Mart, on June 11, 2003, announced that from January 2005 its top 100 suppliers are required to put radio frequency (RFID) tags on their cases and pallets. This goal seems to be achieved as all of the affected suppliers have announced they *will* be ready. Other companies monitor the situation closely, and due to the apparent success they are expected to follow Wal-Mart's example soon.

Basically RFID consists of two devices: A chip, called a transponder or tag, and a device which reads the contents of the chip, referred to as a reader. A tag/reader pair does not have to be in physical contact to communicate, as this is done through air using radio waves. This means that communication can be performed even if the reader cannot see the transponder i.e. no line-of-sight between them.

To even further improve on SCM and the handling of inventory inside stores, placing a tag on individual items is presently discussed. The flipside is that this will bring RFID out to the individual consumer, where it can be used to invade his privacy. Anyone with a scanner (which does not have to be stationary!) will now be able to trace him and know what is in his bags.

To prevent this "Big Brother"-like scenario, different solutions have been suggested. Some of these are based on encryption, which is the objective of this report. At present the main problem regarding encryption in RFID systems is not the strength of the algorithms, but due to constraints whether it is possible (and feasible). The constraints are that tags need to be small, and that only a limited supply of power is available to a tag. Besides these limits tags are not allowed to cost much either!

In this report several encryption algorithms are discussed based upon implementation (using Gezel and VHDL) and synthesis onto different technologies (using Synopsys). Through simulations, from knowledge of what is possible today, and what is believed to happen in the future, the possibility and feasibility of the different encryption algorithms is assessed.

The main conclusion is that encryption is possible with the technology we possess today, at least when we focus on secret key encryption. Encryption in RFID is therefore a question of the cost of it.

Contents

Chapter 1

Introduction

No matter which company you mention, one of the things it is striving to achieve is the highest effectivity at the lowest cost. If the company is manufacturing or just reselling goods, one way to achieve this goal is to always know what they got in stock and where.

The management of goods has until now been relying on bar codes, but it seems as if this is about to change. On June 11, 2003, one of America's largest chains of supermarkets, Wal-Mart, announced that from January 2005 its top 100 suppliers are required to place radio frequency identification (RFID) tags on their cases and pallets [2]. Shortly after the American Department of Defense on October 23, 2003, announced that its suppliers are to place RFID tags on their deliveries [3].

Until today RFID tags have been expensive to manufacture because nobody use them - and nobody use them because they are too expensive to buy! Now it seems as if the RFID ball is rolling, though. In July 2004 Wal-Mart revealed that 137 (and not just the required 100!) of its top suppliers will be able to comply with the January 2005 deadline. At the same time they also announced that the next 200 top suppliers are required to use RFID tags no later than January 2006 [4].

So what is it RFID can which barcodes cannot? The answer to this question consists of (at least) two parts.

Firstly, consider the scenario where a new batch of items arrives at a warehouse where they are checked in. With barcodes the truck transporting the items either has to stop at a specific scanning area, or a person with a bar code scanner has to go to the place where the items are stored. With RFID a reader can be placed at the entrance to the warehouse, making it possible to read the tags as the truck drives through.

Secondly, take a scenario of searching for a specific (perhaps lost) box among hundreds or thousands of other boxes locked up in containers. With barcodes you have to open container after container until you find what you are looking for. With RFID you simply have to walk with a handheld reader among the containers. When it picks up the signal from the right tag you know which container to open.

Consider for example the experience of the American army during the first Gulf War. A lot of containers were hastily transported to camps and stacked in container yards. They did not have the same content, so when a medical officer suddenly had an immediate need for bandages he had to go to the right one. As it was a chaotic situation (it was in the war zone after all), this task would be almost impossible with barcodes, however using RFID made it only a matter of minutes [8].

When pallets and containers are tagged the next step is to tag the individual items on or inside them. This will provide the possibility of improving the handling of items inside stores. One example is “the intelligent shelf”, which is a shelf equipped with an RFID reader. The reader registers what items are on the shelf, and when they are removed it registers this as well. Therefore the shelf is able to signal the store manager when refilling is needed.

The consumer may also profit by having tags on individual items. Examples of this are again intelligent objects such as a washing machine or an oven. The washing machine knows what is inside it, and can inform the operator whether it contains clothes which should not be washed using the chosen program. The oven will be able to learn what has been put inside it, and automatically choose the right way to cook it.

Even when the lifetime of a tagged item is at an end the tag can be of great help. At recycling stations RFID readers can scan items, and if for instance the reading implies that the item is a bottle of wine, the item is directed to the glass container.

From the above it can be seen how practical it is to have tags which can be read at all times. But exactly this property also has an unwanted side effect: If no countermeasures are taken it will be a step away from personal privacy. Anyone with a handheld scanner will be able to trace you just by tracking an item you are carrying. Furthermore they will know exactly what you have in your bag.

It does not even have to be a trace as direct as the above mentioned. Suppose someone observes you over a period of time, noting what kind of clothes you wear. When RFID is widespread enough to be almost ubiquitous this will enable him to trace you through the tags in your clothes.

Several solutions to prevent this privacy invasion have been suggested, some of which use cryptography. Due to constraints on RFID systems cryptography is not without problems though. The size of a tag is limited in order to make it fit onto small objects. This limits the number of gates in an RFID chip and thereby how many operations it can perform. It also limits the amount of energy it has at its disposal: As it cannot have a battery attached the power must be drawn from the incoming signal. Both of these limitations increase the response time of a tag. However, due to standards and to how long a customer is prepared to wait while an item is being scanned, the response time cannot be allowed to become to long. Finally an RFID tag must be inexpensive, as it requires *a lot* of tags to tag individual items. In general it is perceived that the price of one tag can be no more than 5 cents before it is feasible.

This report will look at implementations of XTEA, 3DES, and AES, and thereby be able to make assessments on the possibility and feasibility of embedding cryptographic elements into RFID tags. These implementations will be done in VHDL using GEZEL, and the syntheses and simulations are performed in Synopsys.

By doing this we find that it is possible to embed cryptographic measures in RFID with the technology we possess today, but the cost of it seems to be to high for at least a couple of years into the future.

1.1 Organization

Chapter 2 will give the reader an introduction to RFID and how it is to be implemented into retail, while Chapter 3 will introduce the reader to some basics of cryptography. In Chapter 4 we look at what suggestions have been set forth to enhance privacy in RFID tagging. The limits which this report will use as a basis for its evaluation of embedding cryptography into RFID chip is presented in Chapter 5, and the encryption algorithms are presented in Chapter 6. In Chapter 7 the implementation, synthesis, and simulation tools are presented, followed by the results of performing these operations. The report ends with Chapter 8 which contains the conclusions and suggestions for future works.

Chapter 2

RFID

This chapter introduces the reader to RFID. First the basics of how RFID works are presented, and this will be followed by some examples of where and how it is already deployed today. Finally an explanation of RFID systems as they are proposed in retail is given.

2.1 RFID Basics

Basically an RFID system consists of two devises: A chip which contains information, and an interrogator which can communicate with it. The chip is called a transponder, and the interrogator is referred to as a reader.

2.1.1 The Transponder

The name ‘transponder’ is made from the two words ‘transmitter’ and ‘responder’, which also describe its function: It responds to a request by transmitting its information.

A transponder consists of a chip connected to an antenna, and sometimes also a battery. When a battery is connected it is called an *active* transponder, and when no battery is connected it is called a *passive* transponder. In the case of passive transponders, the energy is obtained by induction on the signal send from the reader. This means that they are only active when inside a readers range (hence the name passive).

2.1.2 The Reader

The purpose of a reader is to inquire for any transponders inside its range and to communicate with these. Therefore a reader sometimes consists of two systems which work together; one system ”shouts” out the inquiry and the second system listens for the responses. At least in case of passive transponders each of these systems do not make sense without the other, and the literature on RFID therefore often just refers to them as the *reader*, a practice which is also adopted in this report.

2.2 Examples of RFID Systems

There is no such thing as *the* RFID system, as they come in many forms. To illustrate how different they can be we consider two well-know applications: An access control key card, and a road toll system.

2.2.1 Access Control Key Card

It is not uncommon for corporations to have access control to at least part of their buildings, and often this is done with key cards. You either have to put the card into a card reader, or place it on a special area next to the door. Both of these solutions can be using RFID (although if the card in the former has a magnetic stripe it could simply just be reading this). In case of RFID, this is known as *close coupling* (distance between reader and transponder ≤ 1 cm).

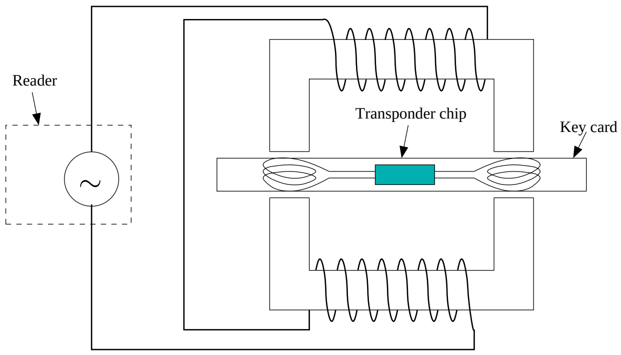

Figure 2.1: Close coupling

Figure 2.1 shows an example of what close coupling can look like, which is similar to how a transformer works. The card is inserted into the air gap of the loop and inductive coupling takes place. The reader is the primary winding which induces a current in the secondary winding, the transponder. The transponder is then activated, and to send its information to the reader it varies the load (impedance) on the windings, which can be detected and interpreted by the reader. This kind of system operates in the 1-10 MHZ frequency range [10, section 3.2.3].

2.2.2 Road Toll

People tired of having to wait in the slow lanes which leads up to the booths where you pay in cash or plastic (e.g. on their way to work everyday) will often acquire the transponder part of an electronic toll-collection system. This is placed in the front window, and they are now allowed to pass the queues and without stopping drive through a special "booth" where the reader is placed.

This RFID system operates in ultra high frequencies (868 or 915 MHz) or microwave frequencies (2.5 or 5.8 GHz), which are the frequencies used by *long-ranged systems* (distance between reader and transponder is > 1 m). The reader continuously sends out a signal thus creating an electro magnetic field. When a transponder enters the field it gets activated. For this kind of system an active transponder is used. The power received from the field is only used to "wake up" the battery, which then drives the chip. When the transponder leaves the field the battery shuts down and thereby the chip is deactivated.

The information is send to the reader by a technique called electromagnetic backscattering (see figure 2.2). When the electromagnetic waves hit a surface (in this case the antenna on the transponder) part of it is reflected. By changing the load on the antenna, the transponder controls how much is reflected. These variations can be detected by the reader and interpreted as information.

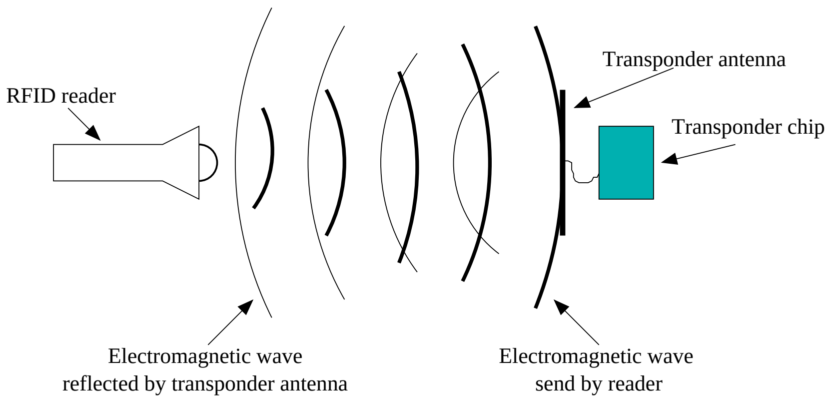

Figure 2.2: Electromagnetic backscattering

2.3 Reading Multiple Tags

Transponders can only be read one at the time, so when more than one enters a reader's scanning area a collision occurs. Different schemes for solving this situation exist, with the two most popular being a Tree Walk and a scheme build on the slotted Aloha protocol.

2.3.1 Tree Walk

To describe how a tree walk is performed, a small example is given. In this the transponders ID only consists of three bits. Three transponders with the ID's “001”, “011”, and “110” are introduced into the reader's scanning area.

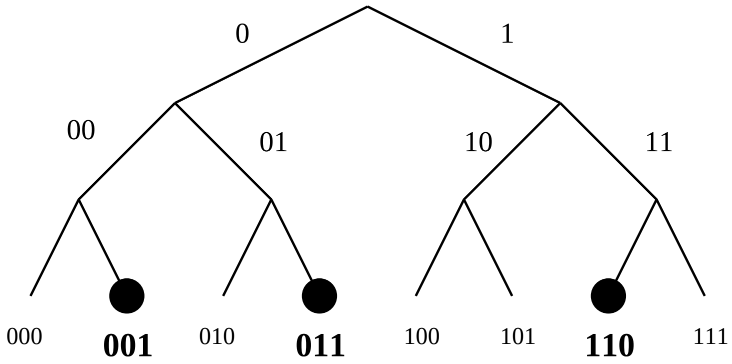

Figure 2.3: The Tree Walk Illustrated

The reader first asks if any transponders have a ‘0’ as the first bit. The “110” transponder does not and goes into a ‘sleep’ state, while the two other answers.

The reader then asks if any transponders have a ‘0’ as the second bit. Again this is confirmed by the “001” transponder, but the “011” transponder goes into the ‘sleep’ state.

Then the reader asks for transponders with a ‘0’ as the third bit. Nobody answers and “001” goes into the ‘sleep’ state.

As nobody answers, the reader backs up one step and asks all transponders which confirmed their presence at the second bit to wake up. This reactivates “001”. The reader now asks for transponders with a ‘1’ as the third bit. “001” answers and is now fully identified.

By continuing this ‘back up one step’ and ‘forward one step’ a number of times all three transponders are identified.

2.3.2 Slotted Aloha Protocol

The Aloha protocol is a simple protocol originally developed for use in radio communication systems, but can be applied in every system where uncoordinated information is send over the same channel. The original protocol has two rules:

1. Whenever you have something to send, send it.
2. If there is a collision when transmitting i.e. another entity is trying to send at the same time, try to resend later. This also applies in case of transmission failure.

Slotted Aloha is a more advanced, but still simple, protocol, where the receiving entity sends out a signal (called a beacon) at equally spaced intervals, thus dividing time into 'slots'. The beacon announces the start of a new slot and thereby the time to start sending the next packet for any entity having one ready.

The version of slotted Aloha applied in RFID collects a number of consecutive slots into groups. At the beginning of each group the reader announces that only transponders with ID's starting with a specified substring are to answer now. Each tag thus activated picks a random number and waits for that many slots before transmitting.

2.4 Regulations and Standards

RFID operates at different frequencies. The choice of frequency depends on the application, but it is not a free choice as radio frequencies are regulated. Regulations are of course needed in order to avoid interference between the different radio systems.

In order to provide interoperability worldwide a variety of specific frequencies for RFID have been decided upon. These are known as *ISM frequencies* (Industrial-Scientific-Medical). Ten such frequencies are defined of which the lowest is 6.78 MHz and the highest is 24.125 GHz. Beside these frequencies everything below 135 kHz is accepted (in North and South America, and Japan the limit is 400 kHz).

When the ISM frequencies were decided upon the world was divided into three regions:

- Region 1: Europe and Africa
- Region 2: North and South America

- Region 3: Far East and Australasia

Not all of the ISM frequencies are applicable worldwide. One example of this is frequencies around 900 MHz: In Region 1 the ISM frequency is a little less (around 860 MHz), while it in Region 2 is a little more (around 910 MHz) [42].

As long as an RFID system is within one of the above mentioned frequencies and adheres to other regulating rules (such as maximum field strength) no special permission needs to be obtained before employing it. There are exceptions though, as some countries have regulations predating the ISM frequencies. However, these become fewer as more governments implement the regulations. The goal is that all countries become as uniformly regulated as possible by 2010. [12]

When RFID was in its early stages much interest fell on three frequencies. These were 135 kHz, 13.56 MHz, and 2.45 GHz, which have all since become ISM frequencies [12]. The reason for this is that they where “free” in most countries and represented a selection of low, intermediate and high frequencies, allowing for RFID systems with different purposes. Some examples are:

135 kHz Animal identification. This can be the ear tags used on cows, which has an RFID transponder incorporated. The ID of the tag can be read up to 1 m away by an RFID reader.

13.56 MHz Contactless smart cards. In Section 2.2.1 close coupling smart cards which can be read at a distance of no more than 1 cm were presented. Smart cards with a longer range exist as well, and are used more frequently. These are proximity coupling cards (range: 7 - 15 cm), and vicinity coupling cards (range: 1 m), which operate at this frequency.

2.45 GHz Absolute positioning reference system for subway trains. This improves safety (e.g., by preventing collisions and by informing trains on local speed limits [18]).

2.5 RFID in retail

Before looking at how RFID is thought to be implemented in retail, it is worth noting that RFID is already employed inside stores. This is the electronic article surveillance (EAS) which provide a very simple form of identification, namely one saying “Here I am”.

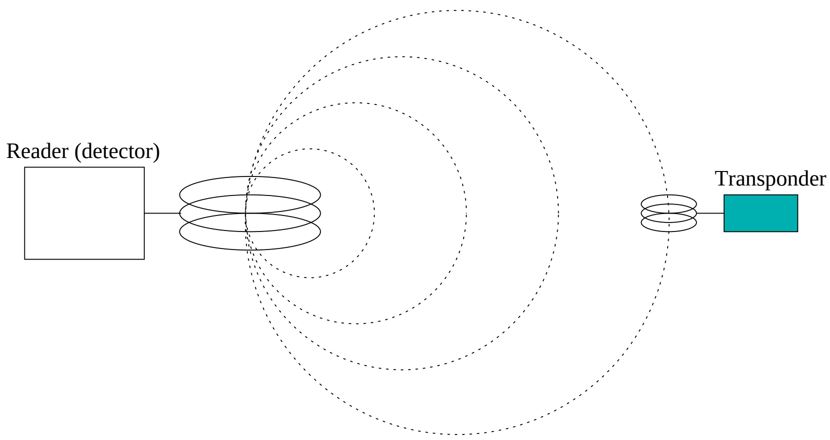

Figure 2.4: Basics in electronic article surveillance

The principle in EAS can be seen in Figure 2.4. As with the access control key card described in Section 2.2.1 the tag (which the transponder in this application is called) is inductively coupled with the detector (reader). When the tag enters the electromagnetic field created by the detector it is powered up and starts sending a signal which the detector picks up.

It is the same principle which is being implemented on pallets and cases in Wal-Mart's warehouses. These tags are more advanced though and therefore able to send out a long identifying number instead of just one bit. Thus tags will function as identifiers in the same way barcodes do today.

The plans are to take the tagging even further than just pallet and cases, namely to tag individual items. Some examples showing how this will improve a lot of procedures are: An inventory check can be performed much quicker and easier (see [13]), "intelligent shelves" will help the store manager to keep the store properly supplied (see Chapter 1), and bad products recalled by manufactures are easily identified.

In order to utilize these advantages it is required that the different vendors use the same system to identify items. Therefore, in 1999, the Auto-ID Center was founded. The Auto-ID center was a partnership between companies in the retail industry, chip manufactures, consulting agencies, and 5 universities situated all over the world. The center's purpose was to research the RFID technology, and to develop a system called Electronic Product Code (EPC). EPC is a barcode-like system, and both the format of the code and the infrastructure to handle it was the goal of the development.

In 2003 the development of EPC was so advanced that the Auto-ID Center was split into two: The Auto-ID Labs and EPC Global. The labs purpose is to continue the research of the RFID technology, while EPC Global is working together with standardization organizations and the industry to bring the academic results out into the real world. EPC Global is also entrusted to maintain the EPC system.

2.5.1 The EPC Network

The EPC is meant as a replacement of the Universal Product Code (UPC) which is used in bar codes. Where UPC describes the object (e.g. a bottle of milk) the EPC assigns individual numbers to each object. It is therefore possible to distinguish "bottle of milk #24" from "bottle of milk #3746".

In order to cover different situations there are many formats of EPC, most of them are derived from existing product codes and consist of either 64 or 96 bits [44]. The format intended to be used in retail is comprised of 96 bits, and is independent of any specifications which exist today. The format is shown in Figure 2.5.

Header	EPC manager	Object class	Serial number
8 bit	28 bit	24 bit	36 bit
Version (00110101)	*Code of manufacturer*	*Article classification*	

Figure 2.5: The general EPC format specified for retail

The header is 8 bits which are "00110101" to identify it as the general 96 bit code. Unlike UPC the EPC does not identify the object directly. Instead a network to decipher the code is applied (see Figure 2.6). When the reader has read the EPC it is send to the computer the reader is connected to. In stores this would be the computer managing the database. This computer runs a middleware program called Savant which supervises the rest of the procedure (the numbers refer to the numbers in Figure 2.6):

1. Savant sends the EPC manager part of the EPC to an Object Name Service (ONS) server via the internet.

2. The ONS server contains addresses to all the servers which contain information on items. Therefore, using the manager part of the EPC, the address is found by the ONS server and returned to Savant.

3. Using Physical Markup Language (PML), a language invented by EPC Global for this purpose, Savant sends the Object class and Serial number parts of EPC to the server with the information. The server is called a PML server.

4. The PML server identifies the information and returns the relevant information to Savant.

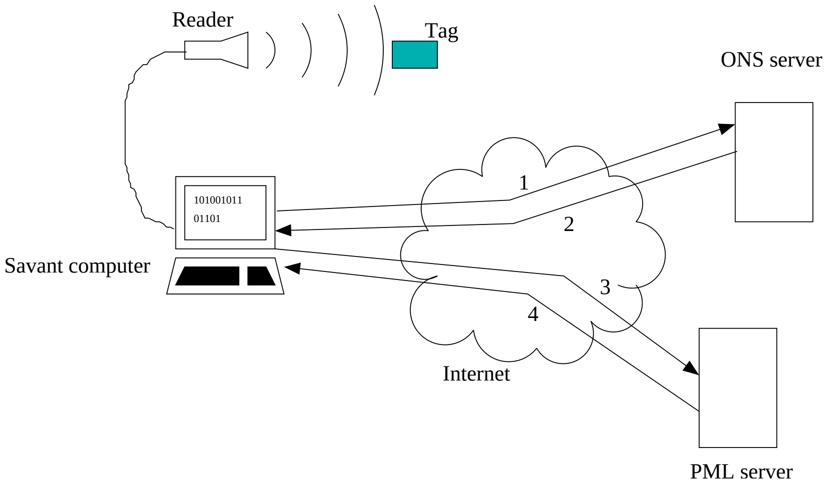

Figure 2.6: The EPC Network

From the above it can be seen that EPC is just a pointer to a server (database) containing the information, giving the ONS server the same function as a DNS server has on the internet. PML does not specify what information can be stored about an object, and the information can therefore change dynamically as an object is moved from place to place, having different owners with different desires.

2.5.2 Classes of Tags

EPC Global has specified six classes of tags which can be found in Figure 2.7 [37]. Presently only specifications for Class 0 and Class 1 tags have been ratified and released. These are called Generation 1 specifications, also referred to as Version 1.

It has been realized that the Generation 1 tags are in lack of many of the features which they were originally indented to have (e.g., Class 0 and Class 1 tags are not compatible with each other, and backwards compatibility with higher classes seems to be at a dead end [38]). Therefore the plan is that EPC Global will ratify specifications for Generation 2 Class 1 tags by late 2004, making up for these shortcomings.

The plans ran into some difficulties, though. Before ratification can take place thorough testing of the specifications needs to be conducted on prototypes, but Intermec (a company specialized in barcode products and data

Class 5	Class 4 capabilities plus the ability to communicate with passive tags
Class 4	Class 3 capabilities plus active communication
Class 3	Class 2 capabilities plus a power source
Class 2	Read, write
Class 1	Read, write once (also known as WORM, write once read many)
Class 0	Read only

Figure 2.7: The classes of RFID tags defined by EPC Global

collection systems) claims that the Generation 2 specifications infringe on some of their intellectual properties (IP). Before this issue was solved the plans was put into a dormant state. On November 3 it was announced that Intermec would suspend its IP claims for 60 days in order to allow for the testing, and exactly one month later it was announced that the testing was completed. The tests validate the Generation 2 specification "feasible" [40, 55].

The Generation 1 Class 0 specification defines the working frequency for communication between reader and tag to 900 MHz, while both 13.56 MHz and 860-930 MHz have been defined for Class 1. Only the members of EPC Global know exactly what is in the Generation 2 specification yet, but in order to ensure a more worldwide interoperability it is expected that at least 900 MHz will still be specified as a working frequency [41, 42]. A good indication of why this might be true is that Wal-Mart is a member of EPC Global Board of Governors, and earlier Wal-Mart has announced that they are only interested in RFID tags working at this frequency [43].

If the 900 MHz frequency is the only one allowed by the specifications it will be a setback for the RFID chip manufactures already having 13.56 MHz chips on the market (e.g. Texas Instruments, Holtek, and Microchip). At least that is what the writer of this report believes, and since many of the affected manufactures participate actively in the EPC Global work the 13.56 MHz frequency should not be written off yet.

Chapter 3

Security and Privacy

In this chapter the basic elements in secure communication is presented. These are encryption and hashing. Furthermore we describe how different systems involving communication and interaction with others have different degrees of privacy. This is done by introducing the *nymity slider*. The chapter ends by discussing where RFID in retail is placed on the slider, and why special attention to incorporate privacy into it is required.

3.1 Setting the Scene

Basically communication between two people consist of person A sending a message to person B. In the cryptographic world these two people are traditionally called Alice and Bob.

When the message is on the way, there is a risk of a third person learning the contents of it. Or perhaps worse yet, the third person might be able to snatch the message and alter it before it reaches its destination. This third (potentially malicious) person is given the name Oscar [1] .

3.2 Secret Key Encryption

To prevent Oscar from learning the contents of the message, called the plaintext P, it is encrypted. Alice and Bob decide on a secret key k_s and an encryption algorithm which uses the key to mix up the message. The algorithm will of course have to be reversible. When Alice wants to send a message to Bob she encrypts the plaintext by using the algorithm under the

[1] Often you would see the third person given different names depending on how much is possible for him or her: Eve for an eavesdropper, Phyllis for a person in physical contact with the system, and so on. This report will use the name Oscar to cover for all of them.

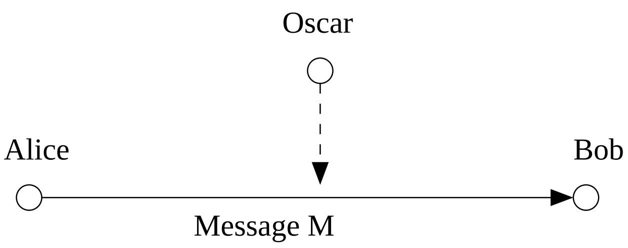

Figure 3.1: Setting the scene

influence of k_s, written as $C = E(k_s, P)$. C is called the ciphertext. When Bob receives the encrypted message he decrypts it by running it through the reversed algorithm again using k_s to influence the result. Decryption is written as $P = D(k_s, C)$.

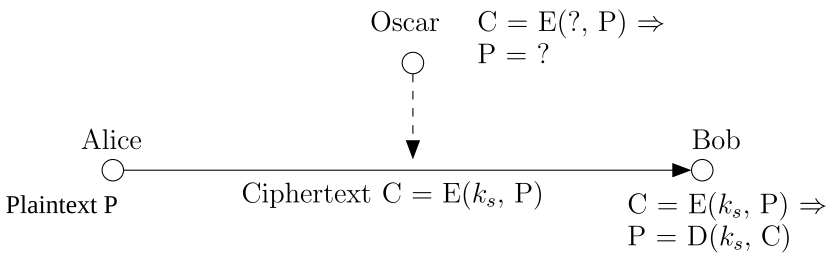

Figure 3.2: Secret key encryption

For secret key encryption two basic principles exists: Stream ciphers and block ciphers. Block ciphers has been more analyzed than stream ciphers, and they seem to be more applicable [7]. Therefore the rest of the description involving secret key encryption will focus on block ciphers, although a short description of stream ciphers will be given first.

3.2.1 Stream Ciphers

In stream ciphers a plaintext is treated as a stream of data, encrypting smaller quantities (bit or byte) of the message as soon as they are available. A typical stream cipher uses a keystream, which is a continuous stream of bits or bytes derived from the secret key. This is xor'ed with the plaintext to obtain the ciphertext. As the same keystream can be generated by a receiver of C who knows k_s, decryption is trivial.

3.2.2 Block Ciphers

In block ciphers a message is divided into blocks of data. A block is comprised of several bytes, and encryption cannot take place until all bytes in a block is ready. Each block is encrypted into a block of the same size.

In Figure 3.2 we see how Oscar is able to learn of C but not P. Even though he cannot know the real message, this might still leak some information to him. One example of this is traffic analysis: Oscar is able to listen to the communication in a system, he knows where a message originates from and where it is destined to, but does not know who are placed at the end of the lines when. Every time Oscar sees the same ciphertext he can with high probability conclude that it is the same two persons involved. This is especially true if the system carries many static messages.

To avoid the kind of traffic analysis just described two things can be done: Change the *mode of operation* or use a *nonce*.

There are four modes of operation for block ciphers (see also Figure 3.3):

Electronic Codebook (ECB) The EBC is the simplest mode, where each block is encrypted individually of each other. In this mode it is possible for Oscar to perform the traffic analysis described above, since a plaintext is always encrypted to the same ciphertext.

Cipher Block Chaining (CBC) In CBC mode the ciphertext from the previous encryption is xor'ed with the plaintext for the present before encryption. A specific plaintext will no longer automatically be encrypted to a specific ciphertext as this depends on the order of plaintexts. But now Alice and Bob always need to agree on what the last ciphertext was in order to communicate. If a message is lost during transmission Alice and Bob will come out of synchronization and the rest of the decrypted messages will not make sense. In this case they will have to agree to start over from some common point. However, if a message is just corrupted during transmission this will only have an impact on decryption of the message itself and the one following it.

Cipher Feedback (CFB), and Output Feedback (OFB) The two last modes, CFB and OFB, make it possible to transform a block cipher into a stream cipher. From Figure 3.3 it can be seen that the plaintexts are xor'ed with a keystream in order to produce the ciphertexts. In CFB the keystream depends on previous ciphertexts, but in OFB the keystream only depends on earlier parts of itself. The advantage of OFB is that if a ciphertext is corrupted during transmission it only influences the plaintext it is an encryption of - the rest of the messages will be decrypted properly. On the other hand OFB makes messages more vulnerable to controlled modifications [7].

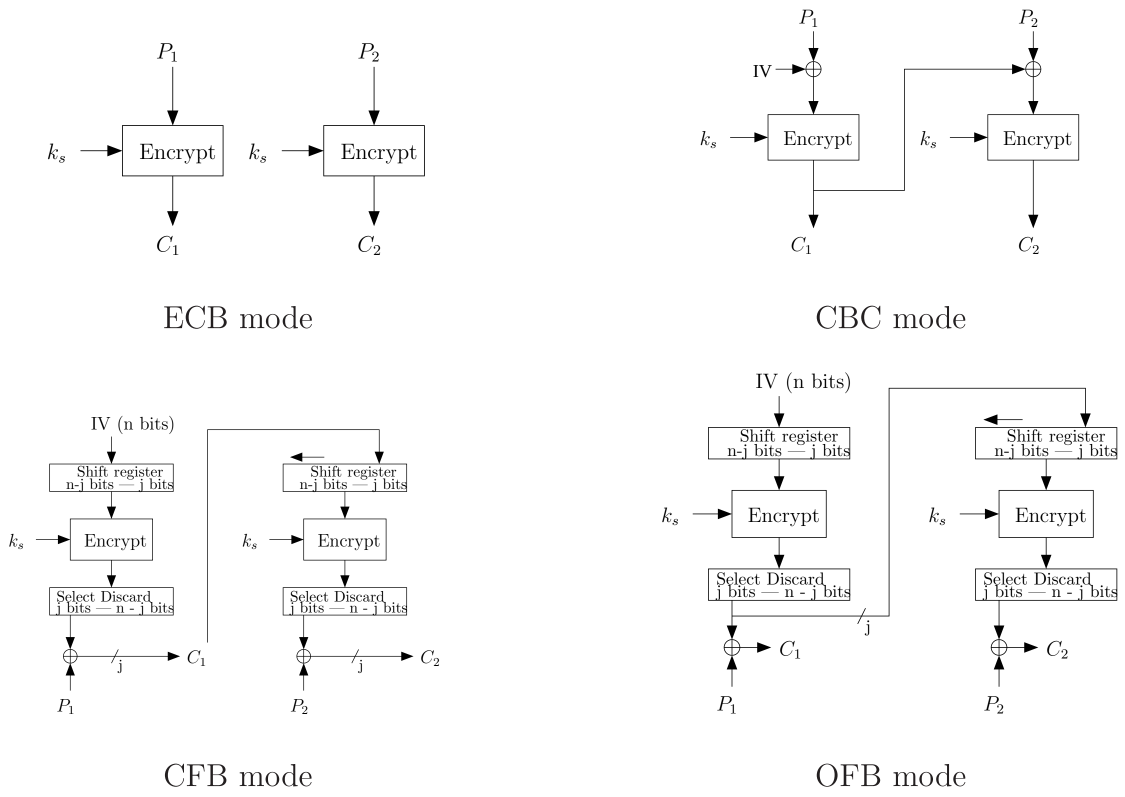

Figure 3.3: The modes of block ciphers

Another way to limit Oscar's traffic analysis is to stick to the basic ECB mode but include a nonce in the encryption scheme. Before encrypting a plaintext it is xor'ed with the nonce, which is a random number. After encryption the ciphertext is transmitted along with the used nonce, which makes it possible for the receiver to decode. Oscar now has to wait until he observes two transmissions with the same ciphertext *and* the same nonce before he is able to make the analysis - something which is unlikely to happen very often if the method for choosing nonces is implemented to distribute them evenly.

3.3 Public Key Encryption

When using secret key encryption it is important to keep the keys secret. When a key is compromised (i.e. Oscar learns what it is) all messages send using this key cannot be assumed secret anymore and it will therefore be necessary to change the keys. This is somewhat trivial to do when only two parties are involved, but it is quite another matter to distribute a new key to larger groups.

To avoid the difficulties in changing keys a public key algorithm can be used. In these algorithms every part holds two keys: A public key k_u, and a private key k_r. The algorithms in public key encryption use one of the key for encryption and the other for decryption. When Alice wants to send a message to Bob she acquires his public key $k_{u,bob}$ and uses it for encryption. When Bob receives the encrypted message he uses his private key $k_{r,bob}$ to decrypt it (see Figure 3.4).

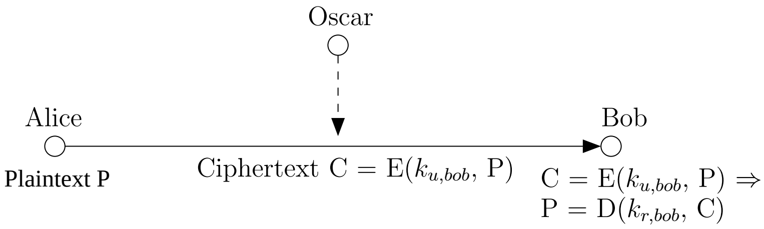

Figure 3.4: Public key encryption

3.4 Hashing

In order to make sure the message Bob receives is the one send by Alice and not an altered message from Oscar, a message authentication is required. This can be performed by a hashing of the message.

A hash function h takes a text x and produces the hashed value $y = h(x)$. The security in hashing is that h is believed to be a one-way function: Given x you can easily find y, but given y you cannot with reasonable feasibility deduce x. To provide an authentication for a plaintext P the hashed value $H = h(P)$ is calculated by the transmitter and send encrypted along with the encrypted text. The receiver decrypts the received authentication and calculates the hashed value of the received decrypted text P', $H' = h(P')$. To check the authentication he verifies that $H' = H$.

3.5 The Nymity Slider

There are many reasons and situations where privacy is required. There are people which to some degree are just "privacy freaks" per definition. For others there can be deeper reasons. Perhaps they hold a position where what they do in private does not influence their job, but others might still misuse it (e.g., in the press).

Privacy in your own home is a matter of course for most people, and it is generally agreed to what it involves: What you do at home nobody but you knows, and only you decide who this is disclosed to. Privacy in public is a different thing: You know people can see you, what you do, and who you are with. So privacy has to be ensured by other means, and most of these involve some form of anonymity through pseudonyms or by "disappearing in the crowd".

3.5.1 The States

In his Ph.D. thesis Ian Goldberg introduces the nymity slider which describes different levels of "nymity" [11]. Whenever you interact with other people you give them some form of information which may or may not (directly or indirectly) identify you. This information is what the nymity slider classifies. At the high end of the slider is no anonymity at all, a state called *verinymity*, and at the low end is total anonymity, called *unlinkable anonymity*. In between are two states with different degrees of anonymity, *persistent pseudonymity* and *linkable anonymity* (see figure 3.5).

Information which uniquely identifies you belongs to the verinymity state. This could be your social security number or credit card number. For information falling into the lower end of this state it depends on the situation whether it uniquely identifies you, or just narrows down the field of potential candidates heavily. One example of this is your name. If for instance you are looking for a person in Denmark you have more than 5 million candidates, but if you are also told that the person's name is Thomas Hjorth you are down to approximately 10 candidates. If instead the field consisted of people registered at Technical University of Denmark, Thomas Hjorth will give you only one result.

At the low end of the nymity slider we find unlinkable anonymity containing information which cannot be linked to a person. An example of this is payment in cash. When you pay in cash in a shop, the shop assistant taking the money is not able to see how or where you got them, and he cannot deduce who you are. When the shop assistant counts the money in the cash register at the end of the day, he is not able to see what money was used to

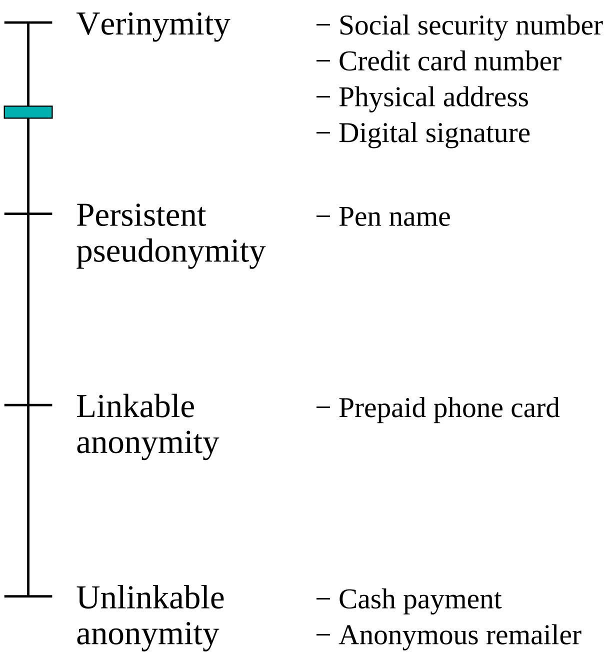

Figure 3.5: The nymity slider

buy which items, nor who the person using them was.

In between the two extreme states we find two other. One of these is linkable anonymity, to which prepaid phone card belong. When you pay for the card you might use a credit card linking the sale to you, but there is no guarantee that you are the person who will use it. Therefore, phone companies might be able to link the incidents where the phone card is used, but they cannot tell who is using it, not even whether it is the same person.

Less anonymous but still not in the verinymity state we find information which can link separate events to the same person, but not who that person is. To this state belongs a pen name (nom de plume). When we see two different books having the same author we believe that it is the same writer who has written both books. It does not have to be the writer's real name which is written on the cover though, so it is not a verinym.

3.5.2 Start Low, Then Move Up

In his thesis Ian Goldberg points out that it is easy to move up on the nymity slider, but close to impossible to move down.

In order to move up the only thing needed is a nudge in the form of some extra information. An example of this is a pen name: If the real name behind a pen name is disclosed it links all books authored by this pen name to this person.

Moving down the slider on the other hand is (close to) impossible. Once you have let the cat out of the bag, it is hard to put it back in; you just cannot make information disappear. Think of the internet for instance. If a web site publishes something and later removes it, almost inevitably there will be a couple of search engines which has it cached. If the published is interesting enough there are probably also a person or two who has copied and saved it on their own hard disk.

Ian Goldberg concludes that when you design a system you should try to make the information in it fall into the lowest possible class on the nymity slider. If (later on) you want to move the system up on the slider it should not be hard to do so by incorporating some extra information into it.

3.6 RFID Tagging on the Nymity Slider

With RFID tagging identically looking items will have different RFID tags identifying them uniquely. The tags will identify what they are, and exactly which numbers the particular items have. Apparently this places the tagging in the linkable anonymity state of the nymity slider, as for instance each time a pair of pants are observed you can log the place and time but not *who* is wearing them.

However, this argumentation is wrong. Actually RFID must be considered to fall into the state of persistent pseudonymity. If you are able to read the tag of a pair of pants, you are also able to read the one in the person's shirt, shoes, socks, mobile phone, wallet... After having observed a person for at short period of time you will be able to make a profile of his clothes and accessories. When you later on observe a number of these items in the same place the conclusion must be that the person is present.

Taking this example to the shops we see how extra information can suddenly be added automatically to the person's profile. Often people are only able to pass the cash register one at a time, which makes it a perfect place to scan for their tags. Given that a person stands in a very limited area close to the register when paying for the things he is buying, the shop is able to single him out and produce a "clothes profile" of him. This can then be linked with his "shopping profile". Given that a person has the same wallet for quite some time, the clothes profile actually only have to consist of the wallet's tag to be of use.

Of course this example is not limited to shops. People reading your tags without your knowledge can happen everywhere (e.g., train stations, your work, restaurants, and parks).

Some will argue that this line of thoughts is paranoid, who would do these kind of things and for what reason? In the shopping example above people might ask what harm is really done?

The answer is that your control of who knows what about you, and thereby your privacy, is greatly diminished. Today you do not *have* to have a shop loyalty card and you can therefore decide not to give the shop a possibility to make a shopping profile of you. With RFID you cannot opt out if a shop decides to make this profile, and your only choice is to accept it or choose another place to do your shopping.

The privacy problem with RFID does not have to involve databases and the building of profiles, though. A less extensive example is a bag snatcher (or, even worse, a mugger) walking round in a public park. With his portable scanner he is able to find out what people have in their pockets and bags, thereby enabling him to pick the best victims.

Whether you worry about being an easy picked 'choice target' of a mugger, or do not want shops to make an extensive shopping profile of you, it is clear that something has to be done to prevent random scans of your person. The next chapter will present different solutions, which involve blocking, obstruction, encryption and killing (of the tags, that is!).

Chapter 4

Privacy in RFID

In this chapter several privacy enhancing technologies (PET) for RFID are examined. These involve physical solutions such as blocking and obstructing, and logical solutions such as hashing and encryption. First a very physical solution is discussed, but opposed to all the others no recovery from it is possible.

4.1 Disabling the Tag

In the RFID specifications from Auto-ID Center (see section 2.5) a 'destroy' command is included. It is not specified how this is carried out, just that "No recovery from the DESTROYED state is possible." and "In this state, the [tag] will no longer [answer] in any way." [15].

There are two methods to do this: Either you set a flag inside the RFID chip telling it not to respond anymore, or you simply destroy it, for instance blowing it by applying too much power. In the first case you can never be sure that the chip is not later re-activated without your knowledge. For this reason the latter of the two alternatives is probably the most acceptable for the common consumer. Popularly speaking the tag is "killed at the counter". This renders it useless and effectively prevented others from reading it.

This is also the reason why killing is discouraged in the long run: If no one (not even yourself) can read the tag, you do not get the advantages outside stores which are described in Chapter 1.

Another problem with killing arises when an item is returned to the store. The item does not necessesarily have an error (e.g. it is a duplicate birthday present), so it can be sold again. However the tag has been killed and therefore the shop cannot scan it as usual. This complicates both the procedure for adding it back to the shop's inventory list, and the procedure when it is sold again.

4.2 Physical Solutions

As the tags which we have today (Class 0 and Class 1) does not give us the possibility to protect our privacy while a tag is active, it is needed to find other methods with do so. These are all physical object influencing on the (reading of) tags from the outside.

4.2.1 Shielding the Tag

From the field of electromagnetism it is known that radio waves can be shielded off from the world by a box made of a conducting material. The box is known as a Faraday Cage [16].

A Faraday Cage works both ways: Radio waves from the outside cannot get in and vice versa. This means that placing a tag inside a container made of a conducting material prevents it from being read, as passive tags do not receive power and signals from active tags cannot escape.

We now have a way to prevent people from scanning your bags in order to learn of its content: Put items you carry around into metal boxes, or into bags with foil lining or a metal mesh inside.

Even though this PET solution does work it can only be part of a final solution. It does prevent people from scanning your bags at random, but you cannot wrap people up in metal foil. Therefore the making of a clothes profile as mentioned in section 3.6 is not thwarted.

4.2.2 Jamming

Another form of shielding is the jamming of radio frequency signals. This is done by having a device which broadcasts radio signals, such that readers are blocked (or at least interrupted).

This solution is even less preferred than the disabling mentioned in Section 4.1. Partly because it is seen as a primitive solution, but mainly because it is (probably) illegal to use in most of the places where it is needed; if it hinders RFID readers in performing, then it also risks obstructing other nearby systems which use radio frequency.

Moreover, customers will have to disable jamming at the checkout so that new items can be bought. This will allow the store to scan their old items as well.

4.2.3 The Blocker Tag

The completely opposite of preventing information from reaching the reader is to apply to much (and wrong) information. Such a solution is the Blocker Tag developed by the laboratories of RSA Security [17].

The Blocker Tag is a device which can simulate all tags. If a reader inquires for e.g. "shirt #13829" it will get the answer that it is present - even if it is not. This is not the same as jamming as described in Section 4.2.2, as the Blocker Tag does not send out random noise, and it only answers when a reader asks.

A problem with the Blocker Tag is if it indiscriminately just answers "present" no matter which tag is asked for. You might have placed it on your person to prevent reading of your clothes, but at the counter of a store it could also interfere which the reading of your groceries.

To prevent this [17] introduces "zones", where the Blocker Tag is only active if a reader inquires inside a zone it has been set to protect. To understand how this works, take the example from Section 2.3.1, which describes a tree walk to identify the tag "001", "011", and "110": The Blocker Tag can be set to only answer if the reader asks for tags with a '1' as the first bit. This divides the tag number space into two, protecting the "110" tag, but leaving "001" and "011" open for scanning. The two zones can reasonably be denoted 'public' and 'private'.

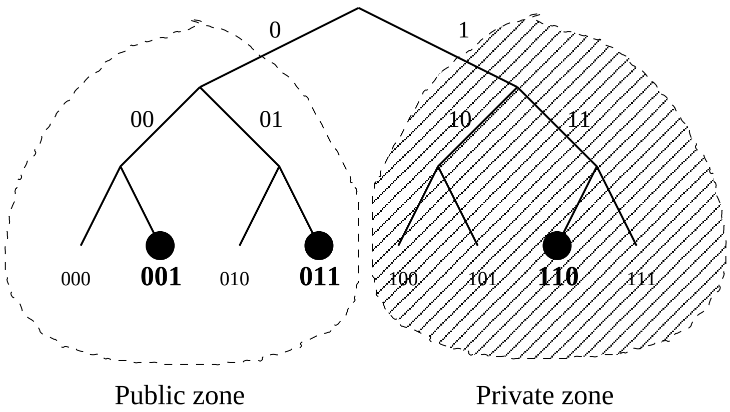

Figure 4.1: Using the first bit to divide into public and private zones illustrated on the Tree Walk example from Section 2.3.1

The concept of zones is especially useful if using rewriteable tags. Inside a store all the goods for sale have a '0' as the first bit, implying that they are in the public zone. When an item is scanned at the counter the first bit is flipped to a '1', thus transferring it into the private zone.

This example with only two zones is very simple, but we need to define more zones. Partly because the world is not black an white (i.e., there is a need for different levels of private/public zones), and partly to prevent certain "attacks" which can be carried out to invade people's privacy [17].

This section has only described how the Blocker Tag works in an RFID system using tree walk for anti-collision, but it can however be used in system which employ the Aloha-like method (described in Section 2.3.2) as well.

4.3 Logical Solutions

So far we have only discussed what can be done if the functionality of the tags is as they are today, that is if we only have Class 0 and Class 1 tags. When tags with higher level classes emerge it is possible to make them more "smart". This can be done because it will be possible to rewrite data to them, thus enabling hashing and encryption which require that you are able to change keys or hash values (e.g. when the ownership changes).

4.3.1 Hash Lock, Version 1

In 2003 Stephen A Weis et al. proposed a scheme which utilizes hashing to "lock" a tag. When the owner does not want a tag to be read, it is given a hash value y which it stores. While it is in the locked state a tag only answer with a meta-ID to queries. The owner then has a database with pairs of EPC and the matching meta-ID. To unlock a tag the (secret) value x is send to it, and by using the hash function h it confirms that $y = h(x)$ [45].

4.3.2 Hash Lock, Version 2

Stephen A Weis et al. mention themselves that the scheme above does not protect against tracking of individuals: The tag always answers with the same meta-ID, so this can be used instead of the real ID (i.e., the EPC).

To make up for this another scheme of hash locking is proposed. In this the tag has a random number generator in the chip. When locked the tag only answers with a pair consisting of a random number r, and a hashed value of r xor'ed with the EPC, (r, y = hash($r\oplus$ EPC)). The value r is changed between every reading, so it is no longer possible to track individuals from the answer of the tag. To unlock a tag a command including its EPC is issued.

The greatest downside in this scheme is that the reader (or the computer it is connected to) has to perform a brute-force search to retrieve the EPC:

Upon receiving the (r, y)-pair it has to fetch all EPC's in its database and to each of them xor it with r followed by the hashing. Only when it finds a match to y has it identified the right EPC.

4.3.3 Hash Lock, Version 3

The second version of the hash lock solution still has a traceability flaw, identified by Miyako Ohkubo et al.: If the tags secret is ever revealed (i.e., the EPC), it will be possible to identify earlier answers from the tag. This is a flaw in the *forward security*, which means that what you do now can be traced later on [46].

Miyako Ohkubo et al. suggest to use an "extended EPC" and a hash chain. Instead of the EPC, a tag stores a secret value s_i. When inquired by a reader it uses a hash function G to reply with the value $a_i = G(s_i)$. The tag also uses a hash function H to calculate a new secret value s_{i+1} (see Figure 4.2).

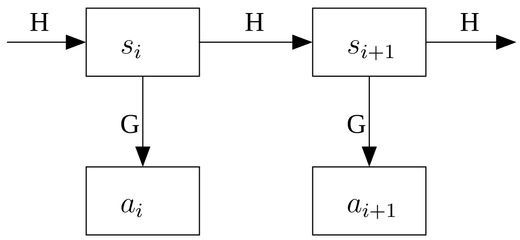

Figure 4.2: Hash lock, version 3

Upon receiving a_i the reader performs a brute-force search just like in version 2 of the hash lock. This time the database contains (EPC, s_1)-pairs, and the procedure is to perform the calculation $a'_i = G(H^i(s_1))$, until it finds a value of s_1 for which $a'_i = a_1$.

Miyako Ohkubo et al. continue by explaining how this can be adapted into the EPC network, but that is beyond the scope of this report. What is of interest to us is that this scheme repairs the flaw in forward security: If at some point the secret value is revealed, it is NOT possible from this information alone to learn of any previous transmissions from the tag. This is because learning the present secret s_i is not enough to learn the previous secret s_{i-1}.

Due to what seems to be scalability problems in the second and third version (the brute-force search), it seems that these are not suitable for supermarkets with millions of items. However, Miyako Ohkubo et al. explain how the search can be distributed by having an expanded EPC which gives a better scalability. This means that the supermarkets *can* actually use version 2 and 3, but as prevention of forward security and individual tracking is not the paramount of importance to supermarkets version 1 might still be preferred by these (because the system requires less hardware/software, thus making it cheaper).

The second and third version is well-suited for the consumer, who will have a more limited amount of objects, resulting in a smaller amount of tag secrets in his personal database. Seemingly the third version uses a more extensive brute-force search, but because of the distribution mentioned earlier this might not be such a great problem.

It is not needed to select only one of the versions for a specific tag, as the preferred version can be activated at any stage in the life of a tag.

4.3.4 Temporary Change of ID

This solution gives the owner of a tag the possibility to temporarily change a tags ID (i.e., the EPC) [47]. When the tag is in its public mode, the ID stored in the chip's ROM can be read by everyone. When the owner wants to disguise the ID he loads a new temporary value into the chip's RAM.

While a value is stored in the RAM, the tag will only use this in its replies. In order to receive the real ID the RAM has to be reset. This applies even to the owner.

By itself, this solution does not do much to prevent tracing of the owner. In order to ensure this at least a procedure to change the temporary ID on a regular basis has to be established. Furthermore, to prevent a malicious person to change the temporary ID at any time, a procedure to do this securely has to be established.

4.3.5 Zero-Knowledge Authentication

Stephan J Engberg et al. suggest a zero-knowledge authentication protocol [48]. As always with zero-knowledge protocols the two parties communicating share a secret, SSDK. Communication between a reader and a tag starts with the reader sending the request along with the zero-knowledge authentication message (ZAM). The ZAM contains two nonces (DT and RSK), and hash values of combinations of these and SSDK:

$$\text{ZAM} = [\text{DT};(\text{RSK} \oplus \text{Hash}(\text{DT} \oplus \text{SSDK})); \text{Hash}(\text{RSK} \oplus \text{SSDK})]$$

Stephan et al. suggest that DT is a date timestamp (or similar) to prevent replays; only ZAM's with stamps indicating a time later than the one in the previous ZAM are accepted. The two other parts of the ZAM are needed for the authentication.

A tag will only respond if the ZAM passes the check. Any response from the tag will contain the following ZAM acknowledgement:

$$\text{Hash}(\text{RSK} \oplus \text{DT} \oplus \text{SSDK})$$

The difference between this solution and the hash locks is that the tag is always in a full operational mode. All you have to do in order to make the tag behave as you want it to is to provide a correct ZAM.

4.3.6 Universal Re-encryption Mixnet

Phillippe Golle et al. suggest a privacy solution called universal re-encryption mixnet (URM), which is based on mixnets[49]. A mixnet is a network based on public cryptography. Initially more than one message (all encrypted with the network's public key) is posted to the network. The network picks up all the messages, decrypts them, and delivers them to their destination. The trick is that the messages are not delivered in the same order they are picked up. Only the network knows how the messages are mixed, so it is not possible for the receiver (or outsiders listening on the wires) to determine who send what message, and who they send it to.

The suggestion Phillippe et al. come with is a system where the network does not decrypt a message, but instead re-encrypts it. A re-encryption of a ciphertext C means that it is transformed into another ciphertext C', but both C and C' decrypts into the same plaintext. Traditionally this will require that the network knows the public key which C is encrypted with, but this is not needed in a URM. Phillippe et al. give an example using ElGamal and two ciphertext: The first ciphertext is the encrypted message (i.e., the ID of the tag), while the second is the identity element of the encryption. When posting a message the two ciphertexts are posted together, and due to algebraic properties of ElGamal it is possible for the network to do the re-encryption without knowledge of the public key used for encryption.

In connection with tags, postings to a network can be all tags in a readers scanning area. The reader receive the tags encrypted IDs, re-encrypts them, and broadcasts the results back. Every tag will therefore receive every re-encrypted ciphertext, but by evaluating each of them (using the secret key) the individual tag can determine which one applies to it.

When readers become ubiquitous they will ensure that the encrypted ID of a tag changes rapidly, thus thwarting a trace on a tag's ID. The downside (besides that the ubiquitousity of reader has to be a reality first) is that this solution requires that a tag has to perform a verification of all receive re-encrypted ciphertext until it finds its own.

There is a probability that the tag will move outside a readers range before it receive the re-encrypted ID. However this is only a problem if it happens all the time, as a tag does not need to update the encrypted ID all the time, just often enough.

4.3.7 Protection of RFID in Banknotes

Another scheme using public key encryption and re-encryption is proposed by Ari Juels and Ravikanth Pappu [50]. The scheme is aimed at tags embedded into banknotes as it has been suggested done, and the RFID-wise security is based on the need for optical reading.

Inside the tag in the banknote two values are kept: A nonce and an encryption of the note's serial number concatenated with its value. The encrypted value is influenced by the nonce and encrypted with a public key from a public, trusted third party. Both of the values are re-writeable, but this can only be done using an access key. You also need the access key for reading the nonce, whereas the encrypted part can be read by everyone.

On the banknote its serial number is printed along with information to construct the access key. This information can only be read optically. The verification of a note is performed in the following manner:

1. Calculate the access key from the optical information.

2. Get the nonce from the tag, and read the serial number and denomination optically.

3. Calculate the excepted value of the encrypted part of the tag.

4. Get the encrypted part from the tag, and compare it to the calculated value.

In order to prevent RFID tracking of a note (by reading the encrypted part in the tag), it is expected that whoever performs a verification also changes the nonce and the encrypted part accordingly.

The security in this scheme is not as high as the constructors had expected, though. Gildas Avoine has uncovered several flaws involving the ability to recover the access key without having to read the information optically, and ciphertext tracking due to infrequent change of data in tag. Before these issues are solved the scheme is not recommendable [51].

4.3.8 Standard Encryption

So far we have only concentrated on logical privacy enhancing solutions which involve hashing and public key encryption. Already existing solutions involving secret key encryption can be used as well. The most basic of them all is simply to encrypt all communication between reader and tag.

Information exchanged between reader and tag is expected to be quite static by itself (due to the limited commands and answers a tag knows) so something has to be done to prevent replay attacks. In Chapter 3 we saw how stream ciphers and other modes than ECB for block ciphers will encode the same data differently depending on the order it comes. However, this will not prevent replay attacks as the whole sequence of the exchange can be recorded.

Therefore some sort of randomized information needs to be included in the transmission. As always this can be done in the form of a nonce. In order to prevent replays it is needed to make reuse of the nonce impossible, and we therefore use the suggestion from Section 4.3.5 to make the nonce a timestamp or something similar. Section 3.2.2 showed that this will make it possible to use a block cipher in ECB mode also.

Most papers mentioning standard encryption also mentions that tags are limited (e.g., size and power supply). These limits are explored in Chapter 5 so for now it suffice to say that all papers agree that secret key encryption is not an option today, mostly because it takes to much space and costs to much.

For some algorithms decryption is somewhat more advanced than encryption, one such example being the advanced encryption standard (AES) (see Section 7.5.1). In order to reduce the space needed, only implementing the encryption or the decryption part of an algorithm can therefore pay off. Martin Feldhofer gives an example of such a scheme [14]. Even though the scheme only seems to be meant as a proof-of-concept, it is still worth considering.

4.4 Summary

In order to ensure privacy when RFID tags are present in everything several solutions have been proposed. Some of these involve some kind of physical means to block unauthorized communication with the tags. As this means that the consumer needs to have some kind of blocking device with him all the time these solutions will fails as soon as he forgets this.

Instead privacy build into the tag is preferred, as this can be active all the time. Some solutions use hashing to ensure only authorized people can communicate with the tag. Others are more advanced and use encryption to ensure the privacy.

However, hashing and encryption is not without problems. Most of the problems come from the fact that RFID is subject to a number of practical and regulated limits, which are explored in the next Chapter 5. This limits the kind of algorithms and schemes which can be implemented.

Chapter 5

RFID Resource Limitations

This chapter will look at what resource limits exist for RFID systems. These fall into four groups: Area, power, time, and cost. But first the technology used to realize a chip is discussed.

5.1 Technology

The production of *semiconductors* (which the chips in RFID tags are) starts with a disc made of silicon called a *wafer*. A wafer is 150-300 mm in diameter so more than one chip can be made of each wafer. The chips are build layer-by-layer by etching away parts of the wafer and *dope* them, i.e. apply other materials with different conducting characteristics. By repeating this procedure a number of times the chips are made. When the chips are ready they are cut from the wafer.

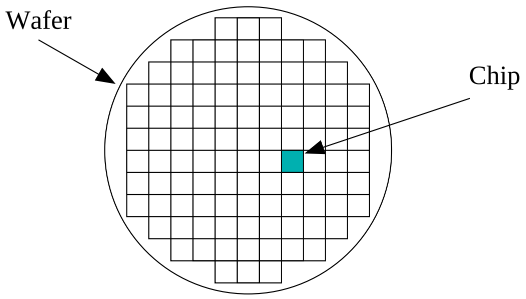

Figure 5.1: Illustration of the way chips are placed on a wafer (not to scale)

The technology for producing chips is constantly improving. The methods for etching a wafer are getting more refined which makes it possible to dope smaller areas with higher precision. This means that the chips et smaller, more advanced, or a combination of both. The methods for cutting the ready-made chip from the wafer is also getting more refined, as the cuts get thinner. This makes it possible to place the chips closer to each other, resulting in better utilization of wafers as more chips can be placed on them, giving less waste.

When describing chip technology there is one important term which needs to be mentioned: *Feature size*, or *minimum geometry*. This is defined as "The smallest line width or spacing between lines or features on a semiconductor die." [21]. The term is closely related to how refined the methods for etching is, and is used as a standard measure when discussing semiconductor technology. In 1993 the ".50 technology" was the leading technology, which means that the smallest possible feature size was $0.50\mu m$. Since then several improvements have evolved and we now have a .09 technology. In fact the .09 technology was achieved already in 2002, but it is not until this year that large scale production of CPU's using it has been done [1] . Today some research goes into the .65 technology, which Intel has already demonstrated in a prototype static RAM (SRAM) [52]. Figure 5.2 shows this evolution along with what the industry is projecting for the nearest future [22].

Year	1995	1998	2000	2002	2003	2004	2006	2007	2009
Feture size	.35	.25	.18	.13	.10	.09	.07	.065	.05

Figure 5.2: Evolution of feature size

5.2 Area

When RFID is used to tag individual items, the size of the chip in the tag will be limited as it has to fit onto (or into) small items. No standards are given for the size of the chip, only for the composite size of chip and antenna i.e. the whole tag. Most texts which do treat this aspect anyway mention that the practical limit on the size of a chip is 1 mm^2 [19, 14, 20].

From Section 5.1, we know that knowledge of the size of a chip alone does not determine how advanced it can be. We also have to know what technology has been used to produce it. Keeping the size at 1 mm^2 but changing the technology from .35 to .18 (reducing the minimum distance on

[1] Apparently it is Intel which has been in the lead with the latest development in feature size the past couple of years, thus making CPU manufacturers the first to take these improvements into use [53]

the chip to half) will in theory make the original circuitry occupy only 1/4 of the available space, leaving plenty of room for more advanced features.

In order to compare sizes of different circuitries independently of the technology another measure is introduced. Basically all circuits are made of NAND gates which are combined to make other/more advanced functions. The number of NAND gates in a circuit is called the *gate equivalent* (GE, or just gates"), and can be used to compare circuits implemented in different technologies. Mapping from one technology to another gives improvements (fewer gates) and compromises (more gates) in different parts, and also results in a different wiring. This means that the mapping is not exactly 1:1 but it is still close enough to qualify as a base for comparison.

How many gates can be squeezed into 1 mm^2, then? This of course depends on the technology used, but also who you ask. Take the .35 technology for instance: Martin Feldhofer says approximately 20.000 gates, but Stephen A Weis only estimates it to be 10.000 [14, 54]. The reason for the discrepancy can be that Stephen are giving "typical" numbers, while Martin presents the edge of the technology.

An RFID chip might be up to 1 mm^2 in size, but of course not all of this space can be used to implements privacy features. Martin Feldhofer estimates that 5000 gates can be used for this purpose, which translates into 0.25 mm^2 if we use his estimate of 20.000 gates/mm^2.

5.3 Power

RFID, in retail, utilizes passive tags, which means that the power for the chip has to be drawn from the reader's signal. The amount of power which can be drawn is limited by the system used. In general, the higher a frequency used, the less power can be gained, which is also the reason why high speed RFID systems as those mentioned in Chapter 2 use active tags.

There are three ways to increase the power supplied to the tag: Lower transmission frequency, higher transmission strength, and a larger antenna.

If you choose to use a lower frequency, more power can be gained from the signal, but the maximal range between reader and tag decreases. Going too low will decrease the range to centimeters, which is to short for retail purposes. This is probably why EPC Global decided that RFID in retail is going to operate at 13.56 MHz and 915 MHz (see Section 2.5); at these frequencies the maximal distance between reader and tag is at least 1 meter, and enough power is available for the chip.

Increasing the transmission strength is not really an option, as this is regulated by governments. The regulations give the maximal field strength allowed, a limit which is quickly reached.

Also the size of the antenna is restricted. This is done through standards but also origins from the physics of the items it is to be implemented into as small items are to be tagged.

At present it is assumed that $\sim 20\mu$A will be available to a tag, but only 10μA can be used for privacy enhancing features [14].

5.4 Timing

There are two sources giving constraints when considering the timing in RFID systems: Standards, and the patience of people using the systems.

Whether you are using an RFID system with a working frequency of f_c 915 MHz or 13.56 MHz this is also automatically the limit to how fast the clock in the tag can run. The standards for the Class 0 and Class 1 tags also defines the slowest possible clock as they define the timing with which a tag is to reply. For a Class 1 tag operating at $f_c = 13.56$ MHz, the smallest unit in a response is $f_c/32 = 423.75$ MHz [2] , which is then the lower limit for the clock.

Today, when items are scanned at the cash register, this is done by making the barcode pass a scanner. you hear a small *bip* which indicates that the item has been scanned. The scanning itself takes milliseconds, but the whole action of grapping the item, make it pass the scanner, and put it down again afterwards typically takes one or two seconds. If 10 - 20 item have to be scanned this can easily take half a minute. All the time the costumer can *see* something is happening and accepts the time that it takes.

It is a bit different when the scanning takes place using RFID. The items do not have to pass a scanner individually, but can be piled together. This means that a scanning only needs to involve placing the trolley with all the items in a designated scanning area. While the scanning is progressing the costumer cannot *see* anything happen. If this goes on for too long people tend to become impatient and irritated, which in the long run is not good for business. This imposes a limit on the time it can take to scan several tags in the same area.

The standards limit on a tag's clock rate should only be seen as an upper limit on internal speed. Individual modules inside a tag can be supplied with a clock which has been scaled down to run slower. As the power to a tag is limited, dividing the clock can be a method to use less. In order to still be able to reach the timing constraint during communication, the tag does not transmit before enough data is ready.

[2] The number 32 is given in the standard, but it is not stated why it has been chosen.

The amount of scaling possible is in theory unlimited, but in practice the tag still have to work with a "reasonable speed", which might be why Martin Feldhofer gives the limit as being (around) 100 kHz. At this frequency a privacy enhancing feature will have to take less than 1000 clock cycles [14].

The timing limit on reading multiple tags is probably not going to be a problem. Based on a data rate (the speed of the tag-to-reader transmission) of only 10 kHz there already exists a solution which can read 100 tags in 4.5s [23].

5.5 Cost

Placing RFID tags on everything is not something which comes for free. For items costing lots of dollars the price of tagging them might be considered negligible. For cheaper items like a Mars bar it becomes more significant. The bar costs US$0.60 and if the price of a tag is just US$ 0.40 this will increase the total price by 66.6%. The price US$ 0.40 is not randomly chosen, as this is what the price for a passive tag starts at today [25].

Within the RFID industry there exists a "magical" goal of 5 cents. When the price of a tag reaches this limit it is generally agreed that price will no longer be a barrier for RFID in retail.

A number of methods can be combined to reduce the price of a tag: Reducing the size of the chip, develop new methods for producing the chips and the antenna, and improvement on the method for attaching the antenna to the chip.

Decreasing the size of a chip means that less silicon is needed, which reduces the price. One way of doing this is to change to a "smaller" technology, e.g. from .25 to .18. The problem with doing so is that the .25 technology is older, the methods for producing chips using it is more widespread and evolved, and therefore cheaper than the .18 technology. So the immediate result is that the chip will become more expensive. As time passes this will change though, as the .25 technology will slowly be abandoned, while .18 becomes more widely used.

Another problem with reducing the size of chips is that it gets more difficult to handle them during production of tags. Today the chips are handled by robots picking them up and placing them. When they get too small the robots can no longer grab them and new methods have to be invented. Again this will at first increase costs (compared to those today) but it could turn out to become a cheaper solution in the long run.

A chip does not have to be made of silicon, that is just the way it is done today. Several companies do research with other materials, including synthetic polymers and special crystals, but so far the price of this is way too high. Also the antenna can be made of other materials. Today they are made from conductive materials, but research in printing them using conductive ink stamped with layers of metal is being conducted [26, 27].

No articles on the subject believe that a 5 cents tag will be achievable within the next couple of years. The opinion most of them offer is that with the technology of today a 5 cents tag is not achievable, but who knows, this might changes within the next few years [28, 29]. A few of them offers a guess to what a tag will cost, naming the price to be 16-20 cents 4-5 years into the future [30, 31]. However, some say that it will be more, and some believe it will be less [39, 56].

5.6 Summary

In this chapter we have defined some limitations for what a privacy enhancing solution to RFID tags has to adhere to:

- Size: 0.25 mm^2
- Energy: 10μA
- Time: 1000 clock cycles at $100kHz$

Besides these limits, a limitation to what the whole tag must cost was found to be 5 cents. Even though no consensus is reach there is a general agreement that this will happen earliest four years from now.

Chapter 6

Design and Algorithms

In this chapter we look at the algorithms we intend to implement and give the reason for choosing them. A description of the framework the implementations adhere to is also be presented.

6.1 Choice of Algorithms

The first choice to be made when deciding which encryption algorithms to investigate is whether it will be secret key or public key algorithms. The choice is actually not that difficult for RFID: Given the constraints mentioned in Chapter 5 and that public key algorithms are quite extensive in computational power, the algorithms with the highest probability of success are secret key algorithms.

From Chapter 3 we know that secret key algorithms fall into the two groups of stream ciphers and block ciphers. As the length of the information which is transmitted to and from a tag is predetermined, a block cipher can easily be chosen. From Chapter 3 we also know that more research has gone into block ciphers, so for this reason alone the systems behind them are considered more secure.

Another argument for choosing block ciphers is that stream ciphers are easy to implement wrong. Even if the cipher is very secure, it can be used in a manner which makes it insecure. An example of this is the WEP (Wired Equivalent Privacy) protocol which is defined in the IEEE 802.11 standard for encryption of data in wireless networks. It uses the RC4 stream cipher algorithm but soon after its release it turned out that the protocol was quite insecure. In June 2004 it was replaced by the IEEE 802.11i standard which uses the AES block cipher(!) for encryption.

With reference to the above arguments we have chosen to investigate the application of block ciphers for encryption in RFID.

6.2 The Framework

It is not necessary to design a whole chip in order to evaluate a block cipher in connection with RFID. Instead you only need to design an encryption *module*, which can later be implemented with the rest of the chip (see Figure 6.1)

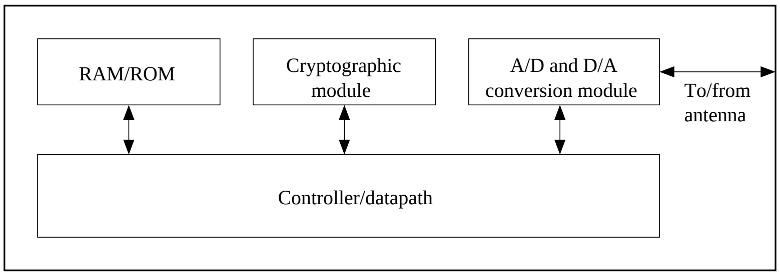

Figure 6.1: The architecture of an RFID chip with a cryptographic module

No matter which block cipher is chosen, the algorithm only needs two inputs (secret key and data) and one output (cipher text). Besides these the module needs to have an input telling it whether to encrypt or decrypt, to have a reset input and a done output for signaling with the controller, and of course to be supplied with a clock.

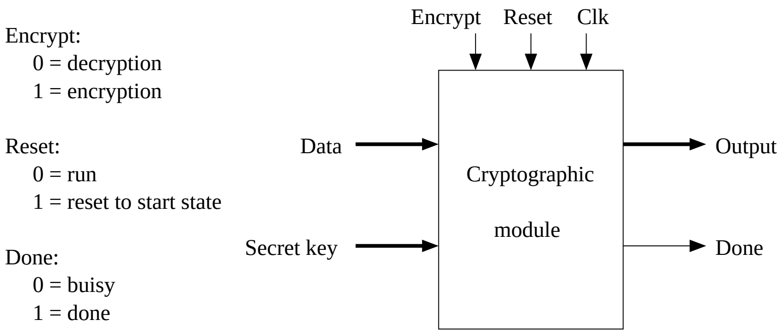

Figure 6.2: The framework for the cryptographic module

Figure 6.2 shows the framework which implementations of the chosen algorithms will have to conform to.

6.3 The Algorithms

When choosing which block ciphers to implement, two algorithms spring naturally to mind: Triple-DES (3DES) and AES. These two have been chosen because the security of these is trusted by everyone to be good, and because they are also fast in hardware, as will be clear from the descriptions in Section 6.3.2 and 6.3.3.

A third algorithm, XTEA, has also been chosen. It is an algorithm which is very simple, yet no attacks have yet succeeded in doing this. Another reason for choosing it is that it has an extensive use of addition. This will give an indication of how well such algorithms do when compared to algorithms such as 3DES (which only uses bit operations), but also of the possibilities for public key algorithms, which are based on heavy arithmetic calculations.

6.3.1 XTEA

XTEA is short for Extended Tiny Encryption Algorithm. As the name implies it is based on an earlier algorithm called TEA (Tiny Encryption Algorithm). The extension was needed to remedy TEA which had been broken. The differences between the two ciphers are not as much in the overall structure, both have the characteristics of an iterated block cipher where each cycle involves two Feistel cipher rounds, but more in the basic operations (shifts and constants) and the order of these. The XTEA algorithm can be found in appendix A.

Even though it was presented back in 1997 and looks simple, the best attack on XTEA presently is "... a related-key differential attack on 26 (...) rounds of XTEA, requiring $2^{20.5}$ chosen plaintexts and a time complexity of $2^{115.15}$ ..." [1]. As can be seen from the algorithm in appendix A the number of cycles is variable, but since 64 rounds (32 cycles) is originally recommended the cipher is of course far from being considered broken.

6.3.2 3DES

In 1976 NIST (National Institute of Standards and Technology, back then called National Bureau of Standard) adopted a slightly modified version of the encryption algorithm called Lucifer as the official data encryption standard, DES. The key length of DES is 64 bits, but in effect only 56 bits are used as the remaining 8 bits are only used for parity.

Back in 1976 DES was state-of-the-art and a brute force attack was inconceivable. But the world changes and development in the technology has been so great that in 1999 a (large) network of computers brute forced DES in less that 24 hours. It has since been shown that for $1000000 it is possible to build a dedicated hardware device which takes only 3.5 hours to go through the entire key space [6]. This development was evident already in 1997 at which point NIST announced that it would start to work on a solution. The result came four years later and was called the advanced encryption standard (AES) which is described in section 6.3.3.

Before the AES algorithm was finally decided upon a temporary solution was given by NIST in 1999. The solution is called triple-DES (3DES), and it simply consists of running three consecutive rounds of DES to encrypt a text. An important thing to notice here is that the three rounds of DES are performed in an encryption-decryption-encryption order (see figure 6.3). This is done to ensure backward compatibility with DES; if the same key is used for all three DES rounds the first two simply cancel out each other, leaving just one ordinary DES round.

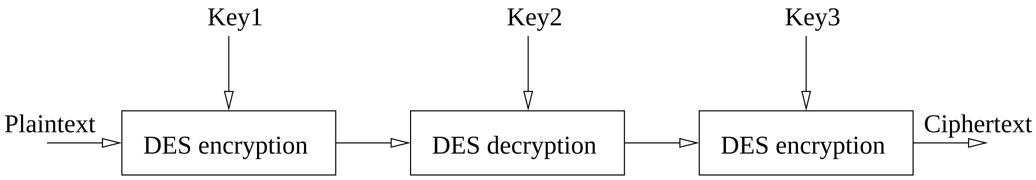

Figure 6.3: The three DES rounds of a 3DES encryption

A description of 3DES will of course mostly be a description of DES, so a (short) description of DES follows.

DES is a block cipher which takes a 64 bit plaintext and a 64 bit key, and produces a 64 bit ciphertext. It consists of 16 Feistel cipher rounds, an initial and final permutation, and a key schedule. One Feistel cipher round can be seen in figure 6.4 and the overall workings of the entire encryption is found in figure 6.5. The actual s-boxes, permutations, and the key schedule rotations can be found in [9].

DES only makes use of the hardware-simple bitwise functions *permutation*, *xor* and *shift*, and lookup tables (S-box), which makes it a good choice to implement in hardware.

6.3.3 AES

In November 2001 NIST announced that the AES (Advanced Encryption Standard) algorithm, based on the Rijndael algorithm, was the new encryption standard. Prior to this had gone more than four years of development and scrutiny by the cryptography society as the algorithm was one of the

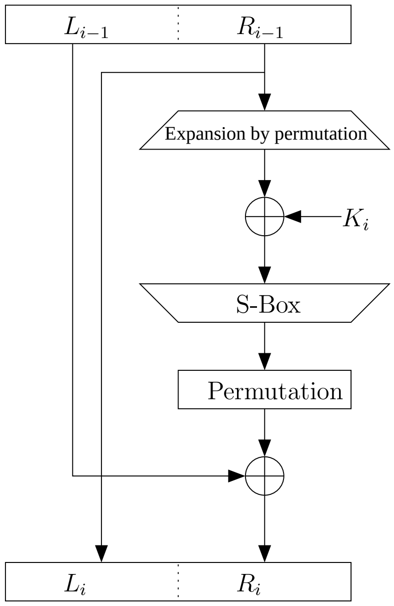

Figure 6.4: One Feistel round of DES

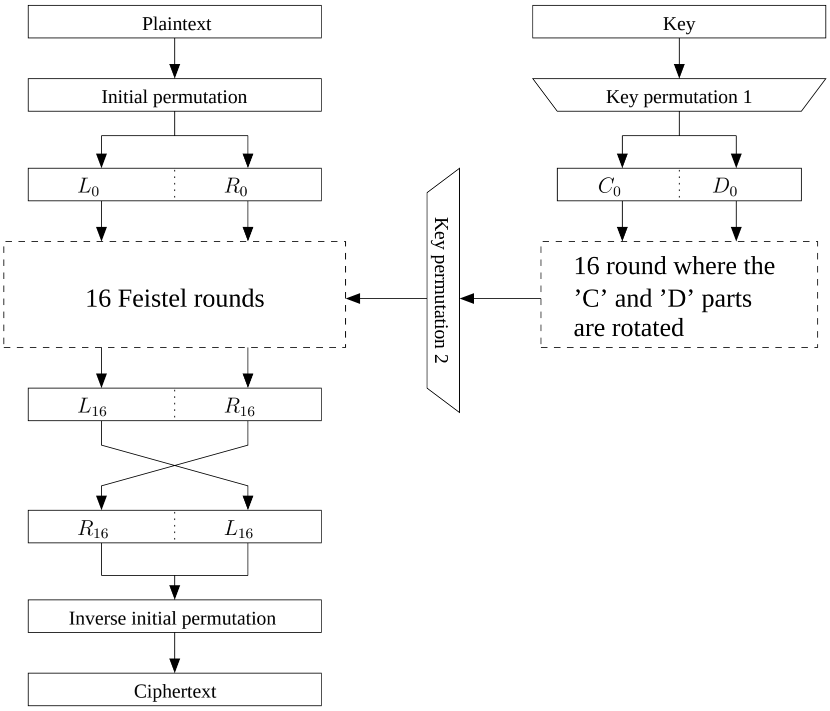

Figure 6.5: Flowchart (sketch) of the full DES algorithm

contestants in the competition (sponsored by NIST) to find the substitute for DES.

AES consists of one s-box, two other kinds of transformations (where one is an ordinary permutation), and a key schedule. The plaintext is 128 bits, while the algorithm is defined for keys of 128, 192, and 256 bits, although each individual implementation of AES only needs to support one size. Figure 6.6 shows the overall working of AES, and the details can be found in [9].

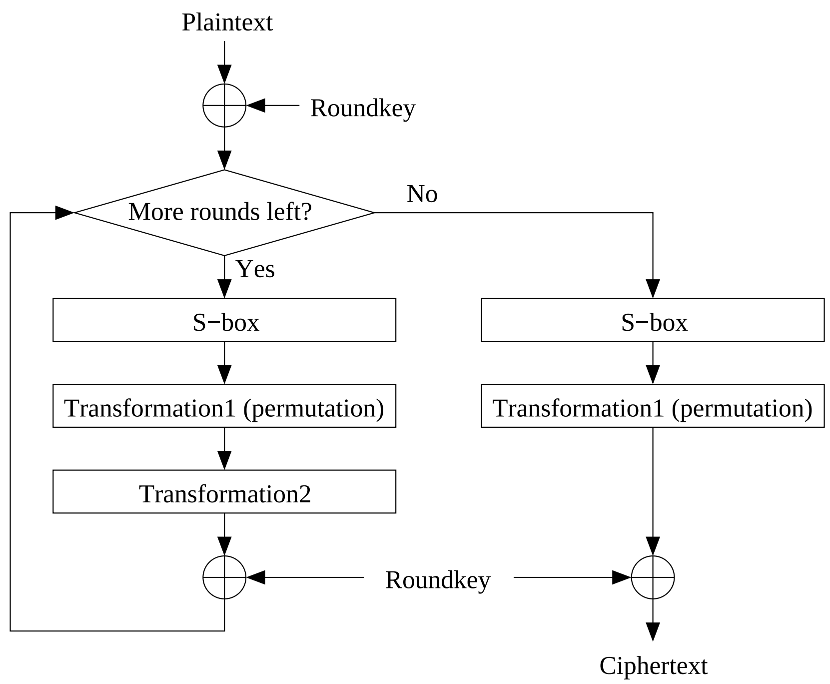

Figure 6.6: Flowchart of the AES algorithm

One demand for AES was that it would have to be simple/fast in hardware, just like DES. Even though AES is partly based on algebraic functions on polynomials, these can be implemented as lookup tables (S-boxes) and small matrix multiplications with fairly simple coefficients. So compared to the complexity of the underlying maths AES *is* actually efficient to implement in hardware.

Chapter 7

Implementation and Performance

This chapter presents the result of our simulations, although first it gives a description of which programs and setup is used. Our analyses of the three algorithms are also presented.

7.1 Synopsys

To synthesize the algorithms into a chip layout the program Synopsys is used. Synopsys can analyze VHDL files and synthesize them to a chip layout.

In order to synthesize, Synopsys needs to be given a *technology library* which describes how different parts of a network work (size, input, output, power consumption ...). Available for this project was a .25 technology library from 1997 and a .18 technology library from 2001. Both libraries are from STMicroelectronics, although it was called SGS-Thomson Microelectronics in 1997.

Synopsys does not understand the whole VHDL language, only a large subset of it. Therefore every design is checked before and after synthesis to detect if any errors have occurred. This is done by using the VHDL debugger associated with Synopsys, vhdldbx.

Some general adjustments for making Synopsys interpret VHDL correctly involve:

- Shift operations are not interpreted correctly so concatenations are used. E.g. instead of writing $x << 4$ when you want to shift x four bits to the left, you select the whole of x except the four leftmost bits and concatenate it with four zeroes.

- Some “standard” VHDL commands like *if(CLK’event AND clk=1 AND reset=1)* are not allowed as Synopsys only accepts one AND on each line. Instead such expressions are split into more lines/if-sentences.
- For at least some VHDL functions Synopsys does not understand hexadecimal arguments. Instead the decimal value has to be supplied to the S-boxes.

When the debugging is done it is time for synthesis, which basically is to use the *compile* command on you design. In connection with this it is possible to set a lot of parameters. Below is a description of some of them together with why or why not to use them:

uniquify When set to true, the compiler will make each instance of the same circuit unique. That is, if two lines of code both contain an addition, it will result in two different circuits, both performing addition. If set to false the compiler can decide to use the same circuit for both additions. As reduction of area is definitely a high concern in RFID chips (see Chapter 5), it is not in our interest to force the compiler to use individual circuits, so this parameter is set to false.

Logic-level optimizations is a collective name for a couple of parameters which decide what the compiler optimizes for. The default setting will collect the most used common subfunctions but only if it does not have a negative influence on the timing. This means that when a specific subfunction is used, the compiler tries to use the same circuit, but not if the maximum delay increases (making the compiler work somewhat opposite of the “uniquify” parameter). As timing is not expected to become a problem, we can remove the maximum delay constraint. To make the compiler check for possible reductions in boolean expressions (perhaps by using don’t care) and thereby reduce the area, the boolean parameter can be set.

map_effort indicates the effort of the compiler. The options are low, medium, and high. When choosing the high option, the compiler does its best to decide how to synthesize the design with respect to the given values of the other parameter. The drawback is that this is very time consuming. For some designs it might even overload the system, making Synopsys exit prematurely.

Having performed compilations with many different settings of parameters, the following steps have been found to yield the best result for our purpose:

1. Start by letting Synopsys analyze the VHDL files. This lets it discover the entities and types present, and also makes it check whether the VHDL code is well-formed (according to Synopsys).

2. Having done the preliminaries a simple compilation with map_effort = medium is performed.

3. Upon the first compilation another one follows. This time the boolean parameter is set, and the timing consideration is removed. This time the map_effort is set to high.

The overall effect of this procedure is that the compiler will optimize for area with little regard to timing.

The reason why the first compile is needed is simply because tests have shown this to yield the best result for the second compile. Omitting the first compile always revealed poorer results. Doubling the second compile seldom improved the result by much (sometimes even increasing the area!), a result which holds true whether or not the first compile was omitted.

At no time during synthesis of any of the algorithms did a timing violation occur, even when the timing constraint was disabled at the first compile. This has been checked for an operating frequency of 13.56 MHz, even though a tag might be limited to a working frequency of 100 kHz in practice.

It is important to note that at no time during the process is a wiring model defined. This means that the synthesis only decides which components are needed to realize a given design, thus outputting the area and power they consume. The wiring between them is of course also known, but as no model is given for this the area and power consumption of it is set to zero.

It is still possible to say something about the consumption of the wires though [34]. The exact value of area and power of course depends on the chip design used (e.g. how many layers is used for the wiring), but a general rule of thumb can be given: The power consumption is *close to* equal to that of the components, while the area is *at least* equal to that of the components, but more often than not it is a little more.

In Section 5.2 gate equivalent (GE) was mentioned. The technology libraries used do not supply information to extract this value for the synthesized designs, yet it is still possible to give an estimate of a design's GE. As mentioned earlier the area given by Synopsys covers only the components of a design. Since each component is build from NAND-gates, we can take the total area and divide it by the area of one NAND-gate to produce the estimated GE. The documentation provided together with the technology libraries states that the smallest NAND-gate in the .25 technology is $27.0\mu m^2$, and in the .18 technology it is $12.288\mu m^2$. These are the values which will be used in the calculations of the GE estimates.

7.2 GEZEL

GEZEL is a high-level programming language, which is used to model and test hardware designs. This is done by making modules, and interconnect these using wires. Each module consists of a state machine and a number of signal flow graphs. Thereby GEZEL conforms to the controller-datapath paradigm which exists in the field of hardware designing (see Figure 7.1).

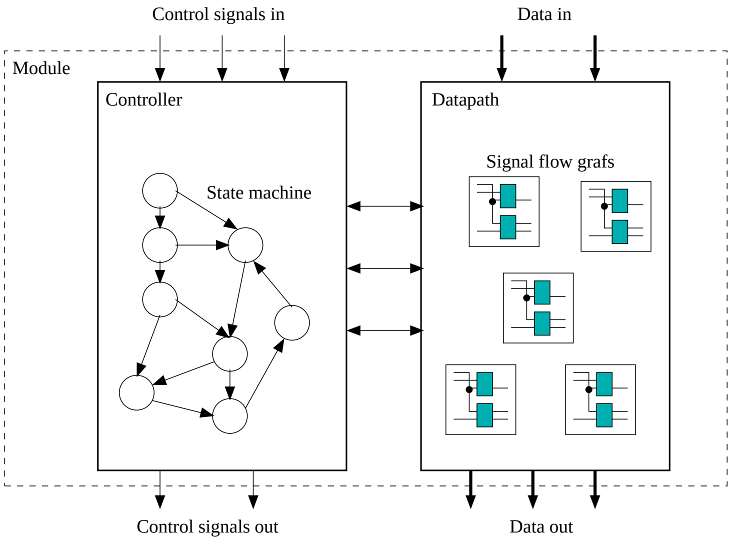

Figure 7.1: Schematic representation of a module

When the design is working satisfactorily GEZEL can translate the file(s) into VHDL. It is simpler and easier to program in GEZEL than VHDL and it is therefore preferred for implementing the algorithms described in Chapter 6.

Before using GEZEL to create VHDL files it has to be checked whether the GEZEL-to-VHDL translator is good enough. Of course we believe that the GEZEL code is translated into VHDL code which functions in precisely the same manner. However, if we compare the GEZEL-VHDL code to the manually written VHDL code in a specific design, and find that it has a much larger spacial area when synthesized it will be no good, as area in Chapter 5 was shown to be a great concern in RFID.

To test for this the XTEA algorithm is implemented both in GEZEL and in VHDL. The two designs are then synthesized and compared.

7.3 Analyzing XTEA

Before implementing XTEA the algorithm is analyzed. The line numbers given in the following refers to those in Appendix A

- As mentioned in Section 6.3.1 XTEA is secure when 32 cycles are used, and the creator of XTEA also recommends this number (line 4). We therefore decide upon 32 *is* to be the number of cycles in the implementation. Thus N can be substituted by 32 everywhere.

- As $N = 32$ everywhere, the two multiplications in lines 18 and 27 always yield the same result, respectively. These two can therefore be replaced by their constant results.

- Also as $N = 32$ everywhere, the two while-loops in lines 20 and 28 are iterated 32 times. Instead of having two different kind of conditions they can be replaced by a counter counting to 32.

- The choice of which part of the key to use in lines 21, 23, 29, and 31 will always depend on the same two bits of the sum, respectively bits 0 and 1, and bits 11 and 12 (counting from left to right, starting with zero). Instead of making the *shift* and *and* operations these bits will be used directly when choosing key part.

- Lines 22 and 30 both contain an operation which involves *sum* and *DELTA*. One is an addition of *DELTA*, the other is a subtraction. In order to save space line 30 is changed to an addition with -*DELTA*. This way the same addition-circuitry can be used for both lines, just with different inputs.

- There are four lines assigning new values to y and z. Each of these can be split into three lines, each containing only one addition/subtraction. Again this is done to make reuse of circuitry possible. Furthermore this lowers the power consumption: The same amount of operations is performed, just over a longer period of time. The total number of clock cycles to perform an encryption or decryption will become 258, making the time it takes 2.58 ms (at 100 kHz, see Section 5.4), which is not an unreasonable long time.

Making the changes suggested above and incorporating the *encrypt* signal as the condition for de- or encrypting gives the algorithm shown in Appendix B.

Area (μm^2)		Power (mW)	
By hand	From GEZEL	By hand	From GEZEL
37945	37216	0.737	0.625
42344	41582	0.789	0.681
41861	39551	0.789	0.655
41476	40706	0.770	0.645
37490	36581	0.726	0.602

Table 7.1: The result of compiling GEZEL made VHDL code and handmade VHDL code of XTEA in Synopsys, using .18 technology and a 13.56 MHz clock.

7.3.1 GEZEL vs. "Handmade" VHDL

The revised XTEA algorithm is implemented in Gezel and VHDL respectively, the GEZEL file is translated to VHDL, and both VHDL versions are synthesized using Synopsys. Instead of only trying the procedure given in Section 7.1 a lot of different approaches to synthesis are tried. The results of these are found in Table 7.1, where the first line is simply a compilation with medium effort, the last line follows the procedure, and in between different combinations involving uniquify, timing constraints, and degrees of effort are tried.

No matter whether it is based on area or power consumption then the GEZEL code produces the best results. Thus it is justified to use GEZEL to implement the rest of the algorithms.

7.4 Analyzing 3DES

DES and thereby 3DES is a quite straightforward algorithm which only contains bit-level operations and table look-up's, so it is somewhat limited how much analysis can be performed on it. The implementation done in this project therefore follows the analysis performed by J. Orlin Grubbe [35].

The only item which needs special mentioning is the key schedule during decryption. As 3DES is a Feistel cipher the decryption of a text is done in the same manner as encryption, but the order of the round keys are reversed. The round keys are produced by making only shifts on the secret key and the last round key can therefore be produced by making the total shifts on the secret key. Thus only the secret key needs to be supplied to the cryptographic module.

7.5 Analyzing AES

7.5.1 Matrix Multiplication in GF(2^8)

As mentioned in Section 6.3.3 the math behind AES does not originally rely on only bit-level operations and table look-up's, but also involve matrix multiplications in connection with finite field calculations (the math behind finite field calculations is beyond the scope of this project, but an introduction to it can be found in [36]). The matrix multiplications can be reduced to shift and xor operations though, which is shown in the following.

The matrix multiplication is one of the transformations in an AES round, and can be written as (see [9])

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 03 & 02 \\ 02 & 01 & 01 & 03 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

...where the constants are in hexadecimal and the size of the variables is eight bits. The interpretation of the multiplication is

$$\begin{aligned} A' &= (02 \bullet A) \oplus (03 \bullet B) \oplus C \oplus D \\ B' &= A \oplus (02 \bullet B) \oplus (03 \bullet C) \oplus D \\ C' &= A \oplus B \oplus (02 \bullet C) \oplus (03 \bullet D) \\ D' &= (03 \bullet A) \oplus B \oplus C \oplus (02 \bullet D) \end{aligned}$$

...where $\oplus$ is an xor operation, and $\bullet$ is a multiplication in the finite field GF(2^8) with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ (binary notation: 100011011) as generator polynomial.

A multiplication of x by 2 can be performed as a left shift of x. Multiplication of y by 3 can be performed as a left shift of y which is then added to y itself, as $3 * y = (2 + 1) * y$. Addition in GF(2^8) is performed by the xor operation, so if we denounce the bits of A as $A = a_7a_6a_5a_4a_3a_2a_1a_0$ and likewise with B, C, and D, the equation for A' can be interpreted as:

$$\begin{array}{ccccccccccl} & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & \text{modulo m(x)} \\ & & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & \\ & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & 0 & \text{modulo m(x)} \\ & & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 & \\ \oplus & & d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 & \\ \hline & & a'_7 & a'_6 & a'_5 & a'_4 & a'_3 & a'_2 & a'_1 & a'_0 & \end{array}$$

As the modulo operation is an xor with $m(x)$ it can be left out during the calculations but then performed on the result. Basically this means that the result will have a ninth bit a'_8, and if this is 1 then $m(x)$ is xor'ed onto the result. As $a'_8 = a_7 \oplus b_7$ this can become a conditional xor when implementing it. The condition changes to $b_7 \oplus c_7$ for B', and similarly for C' and D'.

The multiplication matrix for decryption looks like this:

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

Even though the constants are higher we can still make do with just shifts, xors, and conditional xors. By the same argumentation as before we get the following calculations for A':

$$\begin{array}{cccccccccccc}
 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 \\
 & & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 \\
 & & & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\
 & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & 0 & 0 & 0 \\
 & & & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & 0 \\
 & & & & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\
 & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 & 0 & 0 & 0 \\
 & & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 & 0 & 0 \\
 & & & & c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
 & d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 & 0 & 0 & 0 \\
\oplus & & & & d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\
\hline
 & a'_{10} & a'_9 & a'_8 & a'_7 & a'_6 & a'_5 & a'_4 & a'_3 & a'_2 & a'_1 & a'_0
\end{array}$$

…where $A' = (a'_{10}a'_9a'_8a'_7a'_6a'_5a'_4a'_3a'_2a'_1a'_0)$ modulo m(x). This time there are three conditional xor operations, one for each of a'_{10}, a'_9, and a'_8. Each of these conditions are more elaborate than the condition for encryption, and the total number of xor operations are a lot higher. In order to cut down on the area and power consumption in the implementation it has therefore been chosen to implement the matrix multiplication like this:

- When encrypting, the values of the four variables A', B', C', and D' are calculated during the *same* clock cycle.
- When decrypting, the four values are calculated in four *different* clock cycles.

As there is such a great difference between encryption and decryption, there will be probably be a difference in area and power consumption worth noticing between a full implementation and one only having the encryption part. In Section 4.3.8 it was explained that such a "half" implementation can still be of use.

7.5.2 The Key Schedule

AES contains a *key schedule*, which means that the secret key is expanded into subkeys which are used in the different rounds. In the key schedule a subkey in one round influences the subkey in the next. As opposed to 3DES it is therefore not just a matter of simply combining operations in order to obtain the first subkey during decryption (i.e. the last subkey during encryption). To recover the subkey the following can be done:

- The whole array of subkeys can be given to the cryptographic module. Each round the key contains 128 bits, and since our implementation needs 11 subkeys this amounts to 1408 bits.

- Only the secret key will be supplied to the cryptographic module. If a decryption is needed, the module starts by performing the key schedule to the end to obtain the right subkey. Only then can the decryption begin.

- The cryptographic module is given the right subkey to begin with. That is, if encryption is required the secret key is supplied, where if decryption is needed the appropriate subkey is supplied. It is possible to derive all subkeys from anyone of them with no extra information supplied to the module.

The first solution is of course not an option in RFID today. As the array cannot be given by the reader (because the secret key then would not be secret anymore!), all 1408 bits will have to be stored inside the chip. Today manufactures offer RFID chips which can only store 128 bits, so storing the whole array inside the RFID chip is somewhere far of into the future.

This leaves us with the two last solutions. The first involves storing of 128 bits on the chip while the second involves storing twice as many, namely 256 bits. On the other hand decryption can be started immediately in the second solution, while the first will take some clock cycles to get ready. It is therefore decided to implement both solutions and compare them.

The solution using only the secret key will be called *asymmetric* (because it needs to process the key before starting decryption, which it does not have to do before an encryption), and the solution where the right subkey is present from the beginning will be called *symmetric*. It should be noted that both solutions will take a different number of clock cycles when comparing encryption to decryption. This is due to the extra clock cycles coming from the matrix multiplication (see Section 7.5.1).

Description	Cycles	Time to finish (μs)	Area (μm^2)	Approximated GE's	Power (max) (mW)
XTEA	259	19.1	86688	3210	1.189
3DES	199	14.7	107388	3977	1.371
AES (asym)	83/343	6.12/25.3	761949	28220	
AES (sym)	83/264	6.12/19.5	743985	27555	4.618
AES (half)	83	6.12	486432	18016	

Table 7.2: Results for simulations in .25 technology at 13.56 MHz (Area and power does *not* include the wires!)

Description	Cycles	Time to finish (μs)	Area (μm^2)	Approximated GE's	Power (max) (mW)
XTEA	259	19.1	36581	2977	0.602
3DES	199	14.7	47104	3833	0.697
AES (asym)	83/343	6.12/25.3	344103	28003	1.691
AES (sym)	83/264	6.12/19.5	339431	27623	2.870
AES (half)	83	6.12	229449	18672	

Table 7.3: Results for simulations in .18 technology at 13.56 MHz (Area and power does *not* include the wires!)

7.6 Results of Implementation

In Table 7.2 and Table 7.3 the results of the simulations running at 13.56 MHz are presented.

As the workings of the implementation of XTEA in GEZEL is no different from the one implemented manually, only the results of the former is presented (cf. Section 7.3.1). The two different values for the number of cycles in the symmetric and asymmetric versions of AES are for encryption and decryption respectively. The power consumption of these only denotes the larger of the two, which both times are for encryption although decryption follows closely behind.

Given that the .25 technology is based on a voltage of 1.8 V and the .18 technology is based on 1.6 V, then the implementation with the lowest current flow at "peak time" is XTEA with 960 μA - and that number even excludes what is needed for the wires. From Section 5.3 we know that the maximal available current is 20 μA, so running either of the algorithms at 13.56 MHz is out of the question. As this conclusion was evident before

Description	.25 technology Power (max) (μW)	.25 technology Current (max) (μA)	.18 technology Power (max) (μW)	.18 technology Current (max) (μA)
XTEA	8.46	4.7	5.7	3.6
3DES	8.12	4.5	5.7	3.6
AES (asym)			28.1	17.6
AES (sym)	33.6	18.7	35.8	22.4
AES (half)	21.8	12.1	15.7	9.8

Table 7.4: Results for simulations at 100 kHz (Wires are *not* included in the figures!)

having simulated all algorithm we did not perform these simulations, which is the reason for the "wholes" in the Table 7.2 and Table 7.3.

Increasing the frequency to about 900 MHz will of course not lower the power consumption and thereby the current flow. Instead we simulate what happens at the lowest possible frequency, namely 100 kHz. The results can be found in Table 7.4.

From the tables it can be seen that when shifting between the two technologies the proportions stays almost the same (as would be expected). This means that if the area of algorithm x is close to double that of algorithm y in the .25 technology, then the area of x will also be close to twice that of y in the .18 technology. It will in the following analysis of the results therefore not be needed to specify the technology when discussing relative figures.

7.6.1 XTEA and 3DES

3DES only takes 3/4 as long time as XTEA but neither of the algorithms are in conflict with the number of cycles allowed (in Chapter 5 found to be 1000).

The simulations shows that XTEA and 3DES have a maximal power consumptions which are quite close to each other, no matter whether it is the .25 technology or the .18 technology. Taking the consumption in the wires into account (In Section 7.1 stated to be equal to that of the circuitry) both algorithms in both technologies stays below the 10μA limit given in Chapter 5.

If a choice between them has to be made it can therefore be made solely on basis of their size. Here we see that XTEA occupy an area only 4/5 of 3DES. Taking into account that the wiring doubles the area, again both of them again stay below the limit given in Chapter 5 (0.25 mm^2), and this holds whether we look at the .18 or the .25 technology.

The conclusion must therefore be that it is not limits in the available technology which hinders these two algorithms from being implemented into RFID tags. True, the area and power consumption in the .25 technology are quite close to the limits, but choosing the .18 technology removes the doubt.

Left is the issue about the cost of it. As it can be seen in Section 5.5 there are many opinions about when the cost of a tag is so low that it is feasible to use it in retail. The answer is left open with a conclusion of "earliest four years from now", but still something can be said on basis of this: If it *does in fact* happen earliest four years from now, the most used technology at that time will most likely not be .35 (which many of the sources Chapter 5 build upon) but instead .25 or even .18. Therefore the implementations and simulations performed in this project will be very relevant by then.

As mentioned in Section 6.3, besides determine if it is possible to embed XTEA in an RFID chip, it is also meant to give an indication of whether it will be possible to embed public key algorithms into RFID chips. The implementation of XTEA only makes use of one addition each cycle, while public key algorithms like RSA and ElGamal uses more extensive operations (such as power functions). Implementing these using only one basic operation (subtraction, addition or similar) per cycle will make the number of cycles increase dramatically. To avoid this it will therefore be needed to use more circuits to perform the calculations, thereby increasing both area and power consumption. By how much will have to be explored more closely, but it is highly unlikely that it will come close to staying within the limits set in this report.

7.6.2 The AES Simulations

The three AES implementations only stay inside the timing limit. Neither the area nor the energy limits are kept. A quick conclusion is therefore that one of the following will have to be done before AES can be embedded into an RFID tag:

- Try an even smaller technology like .09.
- Do even more optimization on the present implementation of them. This can both be done in the Gezel or VHDL files, or simply during compilation. However, it is highly unlikely that the compiler will be able to optimize enough.
- Change the way the algorithm is implemented. An example of this is to use more clock cycles performing less operations each cycle.

As expected, only implementing the encryption part of AES reduces the area and energy consumption. Compared to the full AES implementations the reductions are somewhere between 33% – 50%. A natural place to start the improvements to reduce area and power will therefore be this implementation.

That it *actually is* possible to get close to stay below all the limits in a full AES implementation Martin Feldhofer gives an example of [14]. Where our implementation works with all 128 bits in a block at the same time, Martin's only work with 8 at a time. This has increased the number of cycles to 1149, but decreased the power consumption so that it only uses 8.6μA, and the number of gates to 3909 (~ 0.2 mm^2 in his .35 technology).

7.7 Summary

This section presented the programs GEZEL and Synopsys used to implement, synthesize, and simulate the algorithm chosen in Chapter 6. Using XTEA as a test we saw that GEZEL can produce VHDL-files as effectively as if it was done manually, and therefore we chose to implement the more advanced algorithms 3DES and AES in GEZEL.

After having implemented and synthesized the algorithms they were simulated. These simulations showed that it is possible to implement both XTEA and 3DES under the assumptions made about limitations in Chapter 5.

AES on the other hand had quite some way to go before it can be used if you focus on a design which works on all 128 bits in a block at the same time. An example of an implementation which is designed to works with only 8 bits at the time was also given. This showed that it *is actually* possible to get close to staying within the limits.

It therefore seems that the largest technical hindrance for RFID to get into retail is the cost of it.

Also the possibility of implementing public key encryption was commented on. Based on the result for XTEA it was deemed highly unlikely that this form of encryption can be implemented under the limits given in Chapter 5.

Chapter 8

Conclusion

Within recent years radio frequency identification has gained increasingly more attention due to its potential to improve supply chain management. Wal-Mart is one of the driving forces behind this as it has demanded that its top 100 suppliers have to put RFID tags on their cases and pallets from January 1, 2005.

From the level of cases and pallets the next step is to put tags on individual items. Thereby RFID moves into the stores and out to the consumers, giving both parties a powerful tool to helps them perform their everyday tasks. The downside is that it also limits the consumers control over his own privacy.

This problem has been recognized by the parties developing RFID and countermeasures have been explored. These have included physical measures involving carrying some device with you all the time, but as soon as the device is deactivated (or forgotten) you are exposed again. Therefore logical solutions have been suggested instead.

In order to assure only authorized communication with a tag takes place different solutions involving hashing have been suggested. Other solutions presented here have been to

- mask the real ID of a tag with a temporary ID.
- make it impossible to trace communication with tags due to diffusion.
- use optical readings as a base for its security.

In this report the encryption part of the solutions have been examined closer. The focus has been on secret key encryption represented by 3DES and AES. We have also had a closer look at XTEA, and even though this is also a secret key algorithm it has enabled us to say something about public key encryption.

Having implemented the algorithms in GEZEL, and synthesized and simulated them in Synopsys, the following conclusions have been given after setting up limits for RFID tags: With the technology we possess today secret key encryption is possible to embed in tags. The real problem therefore seems to be the cost of it, where the general understanding is that to be feasible for a tag to be implemented into individual items it cannot cost more than 5 cents. Many offer the opinion that this issue will not be solved within the next couple of years. As for the prospect of public key encryption, this seems not to be possible yet, although further investigation into this has to be done before anything can be said with certainty.

8.1 Future Work

We have seen that it is possible to embed secret key encryption algorithms into RFID tags, but the key management which goes along with it has not been explored. This will have to be done. Such examinations can include (but is not limited to) an investigation of: A secure way to substitute the key of one owner with the key of the next, how large a key it is actually possible/feasible to store in a tag, and how many different keys it will be practical to have.

Within the genre of secret key encryption only block ciphers have been investigated in this report, but it will be of interest to us also to examine stream ciphers. Even though block ciphers can use a mode much similar to how stream ciphers work, it might turn out that stream ciphers uses much less space and energy.

Also the issue of public key encryption will have to be examined closer. At least an implementation of such an algorithm has to made, as this report only bases its assumptions for public key encryption on a secret key algorithm using simple arithmetics.

Another topic which can be investigated is a projection on costs of RFID tags. This investigation will for example have to look at methods for assembling tags, different materials for producing a tag (e.g. conductive ink), and the impact of mass producing the tags (which of course will happen when all items will be tagged).

Bibliography

[1] Wikipedia - The Free Encyclopedia, entry = XTEA,
http://en.wikipedia.org/wiki/XTEA

[2] Mark Roberti, on CIO Insight web site, *Analysis: RFID - Wal-Mart's Network Effect*,
http://www.cioinsight.com/article2/0,1397,1455103,00.asp, September 15, 2003.

[3] News release no. 775-03, on United States Department of Defense web site, *DoD Announces Radio Frequency Identification Policy*,
http://www.dod.mil/releases/2003/nr20031023-0568.html, October 23, 2003.

[4] Rick Whiting, on Information Week web site, *Wal-Mart Plans Next Phase Of RFID*, http://www.informationweek.com/story/-showArticle.jhtml?articleID=23903251, July 21, 2004.

[5] RFID Gazette web site, *RFID 101*,
http://www.rfidgazette.org/2004/06/rfid_101.html

[6] Tropical Software web site, *DES Encryption - Overview*,
http://www.tropsoft.com/strongenc/des.htm

[7] William Stallings: *Cryptogrphy and Network Security - Principles and Practice*, 2nd edition, Prentice Hall, 1999.

[8] http://www.almc.army.mil/alog/issues/julaug96/ms075.htm

[9] Computer Security Resource Center homepage, the *"Cryptographic Toolkit"* page, http://csrc.nist.gov/CryptoToolkit/tkencryption.html

[10] Klaus Finkenzeller: *RFID Handbook*, 2nd edition, Wiley, 2003.

[11] Ian Goldberg, *A Pseudonymous Communications Infrastructure for the Internet*, Ph.D. thesis, University of California at Berkeley, fall 2000.

[12] AIM (Assosiation for Automatic Identification and Mobility) web site, http://www.aimglobal.org/technologies/rfid/resources/papers/-rfid_basics_primer.asp

[13] RFID News http://www.rfidnews.org/weblog/2004/10/14/vatican-rfid-at-cnn/

[14] Martin Feldhofer, *Strong Authentication for RFID Systems Using the AES Algorithm*, http://www.iaik.tu-graz.ac.at/aboutus/people/feldhofer/papers/slides_ches04.pdf, slides from presentation at Workshop on Cryptographic Hardware and Embedded Systems - CHES 2004, August 2004.

[15] RFID specification from EPC Global, *13.56 MHz ISM Band Class 1 Radio Frequency (RF) Identification Tag Interface Specification*, http://www.epcglobalinc.com/standards_technology/specifications.html

[16] Wikipedia - The Free Encyclopedia, Entry = Faraday Cage, http://en.wikipedia.org/wiki/Faraday_cage

[17] RSA Security, RSA Laboratories, The Blocker Tag, http://www.rsasecurity.com/rsalabs/node.asp?id=2060

[18] http://www.transcore.com/markets/rail_intermodal.htm

[19] Jeff Lindsay, Walter Reade, and Larry Roth, *Retail RFID Systems without Smart Shelves*, http://www.jefflindsay.com/rfid1.shtml, November 7, 2003.

[20] Warren Hartenstine, *RFID, ROI and the Fashion Vertical*, http://www.techexchange.com/thelibrary/rfid4.html

[21] http://smithsonianchips.si.edu/chiptalk/icevocab.htm

[22] ITRS (The International Technology Roadmap for Semiconductors) web site, *The International Technology Roadmap for Semiconductors: 2003 edition, Executive Summary*, http://public.itrs.net/Files/2003ITRS/Home2003.htm

[23] Trolley Scan(Pty) Ltd web site, *How fast can Trolleyponder protocol scan 1000 items in a single scan?*, http://trolleyscan.co.za/technic1.html

[24] Transponder News web site, *How it works (Part 2)*, http://transpondernews.com/newswrk1.html

[25] Wikipedia - The Free Encyclopedia, Entry = RFID, http://en.wikipedia.org/wiki/RFID

[26] EPC (Electronic Product Code) web site, *Bringing Down the Costs of Tags*, http://archive.epcglobalinc.org/aboutthetech_indepthlook3.asp

[27] Sean Milmo, on Ink World web site, *Potential is Tremendous for RFID and Smart Labels*, http://www.inkworldmagazine.com/Nov032.htm, November 2003.

[28] Automation World web site, *RFID for Perfect Inventory Visibility*, http://www.automationworld.com/articles/Departments/90.html, July 31, 2003.

[29] AIM (Assosiation for Automatic Identification and Mobility) web site, interview with Dan Lawrence (Director of Technology & Commercialization for Precisia), *Printable Tags?*, http://www.aimglobal.org/technologies/rfid/resources/articles/dec03/-PrintedTags.htm, December 2003.

[30] Diane Marie Ward, interview to RFID Journal with Chantal Polsonetti (V.P. of manufacturing advisory services at ARC), *5-Cent Tag Unlikely in 4 Years*, http://www.rfidjournal.com/article/articleview/1098/1/1/, August 26, 2004.

[31] Press release from Gartner, *Companies Should Focus on Business Benefits of RFID, Not 5-Cent Price Myth*, http://www4.gartner.com/5_about/press_releases/asset_112599_11.jsp, October 20, 2004.

[32] The Free Dictionary, Entry = Stream cipher, http://encyclopedia.thefreedictionary.com/stream%20cipher

[33] Synopsys online documentation.

[34] Conversations with Jan Madsen, supervisor of this report.

[35] J. Orlin Grubbe, *The DES Algorithm Illustrated*, http://www.aci.net/kalliste/des.htm

[36] Neal R Wagner, *The Laws of Cryptography: The Finite Field GF(2^8)*, http://www.cs.utsa.edu/~wagner/laws/FFM.html

[37] www.softmatch.com/epc%20eccc%20oct04.ppt

[38] High Tech Aid web site, *The RFID facts*, http://www.hightechaid.com/tech/rfid/rfid_facts.htm

[39] In-Pharma web site, *RFID tags likely to be more costly than expected*, http://www.inpharma.com/news/news-ng.asp?id=55293-rfid-tags-likely, October 11, 2004.

[40] Mary Catherine O'Connor, in The RFID Journal, *Intermec Suspends Royalties for 60 Days*, http://www.rfidjournal.com/article/articleview/1220/1/1/, November 3, 2004.

[41] Impinj web site, *RFID Standards*, http://www.impinj.com/page.cfm?ID=aboutRFIDStandards

[42] Transponder News web site, *Compatibility between the US and Europe radio frequency regions for International trade*, http://transpondernews.com/editori3.html

[43] RFID Journal, *Wal-Mart Details RFID Requirements*, http://www.rfidjournal.com/article/articleview/642/1/1/, November 6, 2003.

[44] EPC Standard Specification, version 1.1 rev. 1.24. http://www.epcglobalinc.org/standards_technology/-EPCTagDataSpecification11rev124.pdf, April 1, 2004.

[45] Stephen A Weis, Sanjay E Sarma, Ronald L Rivest, and Daniel W Engels, *Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems*, http://theory.lcs.mit.edu/~sweis/spc-rfid.pdf, presented at First International Conference on Security in Pervasive Computing in Boppard, Germany, March 12-14, 2003.

[46] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita, Cryptographic Approach to Privacy-Friendly Tags, http://www.rfidprivacy.org/2003/papers/ohkubo.pdf, presented at RFID Privacy Workshop @ MIT in Cambridge, Massachusetts, November 15, 2003.

[47] Sozo Inoue, and Hiroto Yasuura, *RFID Privacy Using User-controllable Uniqueness*, http://www.rfidprivacy.org/2003/papers/sozo_inoue.pdf, presented at RFID Privacy Workshop @ MIT in Cambridge, Massachusetts, November 15, 2003.

[48] Stephan J Engberg, Morten B Harning, and Christian D Jensen, in Proceedings of second annual conference on Privacy, Security and Trust, page 89-101*Zero-Knowledge Device Authentication: Privacy & Security Enhanced RFID preserving Business Value and Consumer Convenience*, Brunwick, Canada, October 13-15, 2004.

[49] Phillippe Golle, Markus Jacobson, Ari Juel, and Paul Syverson, in RSA Conference Cryptographers' Track '04, page 163-178, *Universal Re-encryption for Mixnets*, textsfhttp://www.rsasecurity.com/rsalabs/staff/bios/ajuels/-publications/universal/Universal.pdf, 2004.

[50] Ari Juels, and Ravikanth Pappu, in Financial Cryptography '03, page 103-121, *Squealing euros: Privacy Protection in RFID-enabled banknotes*, http://www.rsasecurity.com/rsalabs/staff/bios/ajuels/-publications/euro/Euro.pdf, 2003.

[51] Gildas Avoine, in International Conference on Smart Card Research and Advanced Applications, *Privacy Issues in RFID Banknote Protection Schemes*, http://www.terminodes.org/getDoc.php?docid=708&docnum=1, Toulouse, August 22-27, 2004.

[52] Intel Press Release, *Intel Drives Moore's Law Forward With 65 Nanometer Process Technology*, http://www.intel.com/pressroom/archive/releases/20040830net.htm, August 30, 2004.

[53] Intel Press Release, *Intel Unveils World's Most Advanced Chip-Making Process*, http://www.intel.com/pressroom/archive/releases/20020813tech.htm, August 13, 2002.

[54] Stephen A Weis, M.Sc. thesis at Massachusetts Institute of Technology, *Security and Privacy in Radio-Frequency Identification Devices*, http://theory.lcs.mit.edu/~cis/theses/weis-masters.pdf, May 9, 2003.

[55] The RFID Journal web site, *EPCglobal Validates Gen 2 Spec*, http://www.rfidjournal.com/article/articleview/1269/1/1/, December 3, 2004.

[56] Mark Roberti, in The RFID Journal, *The 5-Cent Challenge*, http://www.rfidjournal.com/article/articleview/1100/1/2/, August 30, 2004.

(All links referenced in the bibliography were last visited in the period November 3 - December 13, 2004)

Appendix A

The XTEA Algorithm

```
/*
v gives the plaintext of 2 words
k gives the key of 4 words
N gives the number of cycles, 32 are recommended
if negative causes decoding, N must be the same as
     for coding
if zero causes no coding or decoding
assumes 32 bit "long" and same endian coding or
     decoing */

tean(long *v,long *k,long N)
{
unsigned long y = v[0],z = v[1],DELTA = 0x9e3779b9;
unsigned long limit,sum;

if (N > 0) /* the "if" code performs encryption */
   {
   limit = DELTA * N;
   sum = 0;
   while (sum != limit)
      y += (z << 4^z >> 5) + z^sum + k[sum&3],
      sum += DELTA,
      z += (y << 4^y >> 5) + y^sum + k[sum >> 11&3];
   }
else     /* the "else" code performs decryption */
   {     /* IT IS TRULY MINUSCULE */
   sum = DELTA * (-N);
   while (sum)
      z -= (y << 4^y >> 5) + y^sum + k[sum>>11&3],
      sum -= DELTA,
      y -= (z << 4^z >> 5) + z^sum + k[sum&3];
   }
v[0] = y;
v[1] = z;
return;
}
```

Appendix B

The Revised XTEA Algorithm

```
/*
v gives the plaintext of 2 words
k gives the key of 4 words
encrypt decides whether to encrypt or decrypt
     encrypt = 0 gives decrypion
     encrypt = 1 gives encryption */
{
(32 bit) y = v[0],z = v[1],DELTA = 0x9e3779b9;
(32 bit) minusDELTA = 0x61c88647,sum,
(6 bit) noCycles = 0;  /*counter for the 32 round*/
if (encrypt != 0) /* the "if" code performs encryption */
   {
   sum = 0;
   while (noCycles != 32)
      y += (z << 4^z >> 5);
      y += z^sum;
      y += k[sum(bit 1:0)];  /*selecting bit 1 down to 0*/
      sum += DELTA, noCycles++;
      z += (y << 4^y >> 5);
      z += y^sum;
      z += k[sum(bit 12:11)];
   }
else     /* the "else" code performs decryption */
   {
   sum = 0xc6ef3720;  /*= 32*DELTA*/
   while (noCycles != 32)
      z -= (y << 4^y >> 5);
      z -= y^sum;
      z -= k[sum(bit 12:11)];
      sum += minusDELTA, noCycles++;
      y -= (z << 4^z >> 5);
      y -= z^sum;
      y -= k[sum(bit 1:0)];
   }
v[0] = y;
v[1] = z;
return;
}
```

Appendix C

3DES in GEZEL

```
   tabsize
//Implementation of 3-DES
//
//Run time is 199 cycles

dp tripledesDP(in in_t : ns(64);  //text to en-/decrypt
               in key1 : ns(64);  //the key used in first "
                  DES round"
               in key2 : ns(64);  //the key used in second "
                  DES round"
               in key3 : ns(64);  //the key used in third "
                  DES round"
               in encr : ns(1);   //0 = decrypt, 1 = encrypt
               out out_t : ns(64);  //the output text
               out done : ns(1))  //0 = out_t not ready, 1 =
                  out_t ready
{
   lookup sbox1 : ns(4) =
               {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,

                0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,

                4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,

                15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};

   lookup sbox2 : ns(4) =
               {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,

                3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,

                0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
```

```
              13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};

lookup sbox3 : ns(4) =
             {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,

              13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,

              13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,

              1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};

lookup sbox4 : ns(4) =
             {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

              13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,

              10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,

              3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};

lookup sbox5 : ns(4) =
             {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,

              14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,

              4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,

              11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};

lookup sbox6 : ns(4) =
             {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,

              10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,

              9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,

              4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};

lookup sbox7 : ns(4) =
             {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,

              13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,

              1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,

              6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};

lookup sbox8 : ns(4) =
             {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
```

```
            1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,

            7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,

            2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};


reg enc : ns(1);      //0 = decrypt, 1 = encrypt
reg key : ns(64);     //contains the key currently used
reg c, d : ns(28);    //the two halves of the key
reg k : ns(48);       //contains the round keys
reg E : ns(48);       //the expanded R-half
reg L, R : ns(32);    //the halves of the "round message"
reg desRound : ns(2); //keeps track of how many full DES
   en-/decryptions
                      //done (needs to de done three times
                         in 3-DES)

sig sb1i, sb2i, sb3i, sb4i, sb5i, sb6i, sb7i, sb8i : ns(6)
   ;
sig sb1o, sb2o, sb3o, sb4o, sb5o, sb6o, sb7o, sb8o : ns(4)
   ;
sig sboxout, pout : ns(32);
sig RL : ns(64);

sfg output_idle
{
   out_t = 0x0000000000000000;
   done = 0;
}

sfg read_key1
{
   key = key1;
}

sfg read_key2
{
   key = key2;
}

sfg read_key3
{
   key = key3;
}

sfg key_perm1
{
```

```
   c = (key[7]#key[15]#key[23]#key[31]#key[39]#key[47]#key
       [55]#key[63]#
         key[6]#key[14]#key[22]#key[30]#key[38]#key[46]#key
             [54]#key[62]#
         key[5]#key[13]#key[21]#key[29]#key[37]#key[45]#key
             [53]#key[61]#
         key[4]#key[12]#key[20]#key[28]);
   d = (key[1]#key[9]#key[17]#key[25]#
         key[33]#key[41]#key[49]#key[57]#key[2]#key[10]#key
             [18]#key[26]#
         key[34]#key[42]#key[50]#key[58]#key[3]#key[11]#key
             [19]#key[27]#
         key[35]#key[43]#key[51]#key[59]#key[36]#key[44]#
             key[52]#key[60]);
}

sfg get_encr
{
   enc = encr;
}

sfg init_desround
{
   desRound = 0;
}

sfg key_perm2
{
   k = (c[14]#c[11]#c[17]#c[4]#c[27]#c[23]#c[25]#c[0]#
         c[13]#c[22]#c[7]#c[18]#c[5]#c[9]#c[16]#c[24]#
         c[2]#c[20]#c[12]#c[21]#c[1]#c[8]#c[15]#c[26]#
         d[15]#d[4]#d[25]#d[19]#d[9]#d[1]#d[26]#d[16]#
         d[5]#d[11]#d[23]#d[8]#d[12]#d[7]#d[17]#d[0]#
         d[22]#d[3]#d[10]#d[14]#d[6]#d[20]#d[27]#d[24]);
   out_t = 0x0000000000000000;
   done = 0;
}

sfg key_lsh1
{
   c = c[26:0]#c[27];
   d = d[26:0]#d[27];
   out_t = 0x0000000000000000;
   done = 0;
}

sfg key_lsh2
{
   c = c[25:0]#c[27:26];
```

```
   d = d[25:0]#d[27:26];
   out_t = 0x0000000000000000;
   done = 0;
}

sfg key_rsh1
{
   c = c[0]#c[27:1];
   d = d[0]#d[27:1];
   out_t = 0x0000000000000000;
   done = 0;
}

sfg key_rsh2
{
   c = c[1:0]#c[27:2];
   d = d[1:0]#d[27:2];
   out_t = 0x0000000000000000;
   done = 0;
}

sfg text_ip_to_LR
{
   L = (in_t[6]#in_t[14]#in_t[22]#in_t[30]#in_t[38]#in_t
       [46]#in_t[54]#
         in_t[62]#in_t[4]#in_t[12]#in_t[20]#in_t[28]#in_t
            [36]#in_t[44]#
         in_t[52]#in_t[60]#in_t[2]#in_t[10]#in_t[18]#in_t
            [26]#in_t[34]#
         in_t[42]#in_t[50]#in_t[58]#in_t[0]#in_t[8]#in_t
            [16]#in_t[24]#
         in_t[32]#in_t[40]#in_t[48]#in_t[56]);
   R = (in_t[7]#in_t[15]#in_t[23]#in_t[31]#in_t[39]#in_t
       [47]#in_t[55]#
         in_t[63]#in_t[5]#in_t[13]#in_t[21]#in_t[29]#in_t
            [37]#in_t[45]#
         in_t[53]#in_t[61]#in_t[3]#in_t[11]#in_t[19]#in_t
            [27]#in_t[35]#
         in_t[43]#in_t[51]#in_t[59]#in_t[1]#in_t[9]#in_t
            [17]#in_t[25]#
         in_t[33]#in_t[41]#in_t[49]#in_t[57]);
}

sfg R_to_L
{
   L = R;
}

sfg expand_R
```

```
{
   E = (R[0]#R[31]#R[30]#R[29]#R[28]#R[27]#
        R[28]#R[27]#R[26]#R[25]#R[24]#R[23]#
        R[24]#R[23]#R[22]#R[21]#R[20]#R[19]#
        R[20]#R[19]#R[18]#R[17]#R[16]#R[15]#
        R[16]#R[15]#R[14]#R[13]#R[12]#R[11]#
        R[12]#R[11]#R[10]#R[9]#R[8]#R[7]#
        R[8]#R[7]#R[6]#R[5]#R[4]#R[3]#
        R[4]#R[3]#R[2]#R[1]#R[0]#R[31]);
}

sfg k_xor_E
{
   E = k^E;
   out_t = 0x0000000000000000;
   done = 0;
}

sfg sbox
{
   sb1i = E[47]#E[42]#E[46:43];
   sb1o = sbox1(sb1i);
   sb2i = E[41]#E[36]#E[40:37];
   sb2o = sbox2(sb2i);
   sb3i = E[35]#E[30]#E[34:31];
   sb3o = sbox3(sb3i);
   sb4i = E[29]#E[24]#E[28:25];
   sb4o = sbox4(sb4i);
   sb5i = E[23]#E[18]#E[22:19];
   sb5o = sbox5(sb5i);
   sb6i = E[17]#E[12]#E[16:13];
   sb6o = sbox6(sb6i);
   sb7i = E[11]#E[6]#E[10:7];
   sb7o = sbox7(sb7i);
   sb8i = E[5]#E[0]#E[4:1];
   sb8o = sbox8(sb8i);
   sboxout = sb1o#sb2o#sb3o#sb4o#sb5o#sb6o#sb7o#sb8o;
   pout = (sboxout[16]#sboxout[25]#sboxout[12]#sboxout
       [11]#
           sboxout[3]#sboxout[20]#sboxout[4]#sboxout[15]#
           sboxout[31]#sboxout[17]#sboxout[9]#sboxout[6]#
           sboxout[27]#sboxout[14]#sboxout[1]#sboxout[22]#
           sboxout[30]#sboxout[24]#sboxout[8]#sboxout[18]#
           sboxout[0]#sboxout[5]#sboxout[29]#sboxout[23]#
           sboxout[13]#sboxout[19]#sboxout[2]#sboxout[26]#
           sboxout[10]#sboxout[21]#sboxout[28]#sboxout[7])
              ;
   R = pout^L;
}
```

```
   sfg inc_desround
   {
      desRound = desRound+1;
   }

   sfg inv_enc
   {
      enc = ~enc;
   }

   sfg inv_ip
   {
      RL = (R#L);
      out_t = (RL[24]#RL[56]#RL[16]#RL[48]#RL[8]#RL[40]#RL
          [0]#RL[32]#
               RL[25]#RL[57]#RL[17]#RL[49]#RL[9]#RL[41]#RL
                  [1]#RL[33]#
               RL[26]#RL[58]#RL[18]#RL[50]#RL[10]#RL[42]#RL
                  [2]#RL[34]#
               RL[27]#RL[59]#RL[19]#RL[51]#RL[11]#RL[43]#RL
                  [3]#RL[35]#
               RL[28]#RL[60]#RL[20]#RL[52]#RL[12]#RL[44]#RL
                  [4]#RL[36]#
               RL[29]#RL[61]#RL[21]#RL[53]#RL[13]#RL[45]#RL
                  [5]#RL[37]#
               RL[30]#RL[62]#RL[22]#RL[54]#RL[14]#RL[46]#RL
                  [6]#RL[38]#
               RL[31]#RL[63]#RL[23]#RL[55]#RL[15]#RL[47]#RL
                  [7]#RL[39]);
      done = 1;
      $display("cycle ", $cycle, " out_t = ", $hex, out_t);
   }

   sfg xchangeLR
   {
      L = R;
      R = L;
   }
}

fsm desFSM(tripledesDP)
{
   initial s0a;
   state s0b,
         s1a, s2a, s3a, s4a, s5a, s6a, s7a, s8a, s9a, s10a,
         s11a, s12a, s13a, s14a, s15a, s16a,
         s1b, s2b, s3b, s4b, s5b, s6b, s7b, s8b, s9b, s10b,
         s11b, s12b, s13b, s14b, s15b, s16b,
```

```
      s1c, s2c, s3c, s4c, s5c, s6c, s7c, s8c, s9c, s10c,
      s11c, s12c, s13c, s14c, s15c, s16c,
      s1d, s2d, s3d, s4d, s5d, s6d, s7d, s8d, s9d, s10d,
      s11d, s12d, s13d, s14d, s15d, s16d,
      s17;

@s0a (read_key1, text_ip_to_LR, get_encr, init_desround,
   output_idle) -> s0b;
@s0b (key_perm1, output_idle) -> s1a;

@s1a if (enc) then (key_lsh1) -> s1b;
     else (output_idle) -> s1b;
@s1b (key_perm2, expand_R) -> s1c;
@s1c (k_xor_E) -> s1d;
@s1d (sbox, R_to_L, output_idle) -> s2a;

@s2a if (enc) then (key_lsh1) -> s2b;
     else (key_rsh1) -> s2b;
@s2b (key_perm2, expand_R) -> s2c;
@s2c (k_xor_E) -> s2d;
@s2d (sbox, R_to_L, output_idle) -> s3a;

@s3a if (enc) then (key_lsh2) -> s3b;
     else (key_rsh2) -> s3b;
@s3b (key_perm2, expand_R) -> s3c;
@s3c (k_xor_E) -> s3d;
@s3d (sbox, R_to_L, output_idle) -> s4a;

@s4a if (enc) then (key_lsh2) -> s4b;
     else (key_rsh2) -> s4b;
@s4b (key_perm2, expand_R) -> s4c;
@s4c (k_xor_E) -> s4d;
@s4d (sbox, R_to_L, output_idle) -> s5a;

@s5a if (enc) then (key_lsh2) -> s5b;
     else (key_rsh2) -> s5b;
@s5b (key_perm2, expand_R) -> s5c;
@s5c (k_xor_E) -> s5d;
@s5d (sbox, R_to_L, output_idle) -> s6a;

@s6a if (enc) then (key_lsh2) -> s6b;
     else (key_rsh2) -> s6b;
@s6b (key_perm2, expand_R) -> s6c;
@s6c (k_xor_E) -> s6d;
@s6d (sbox, R_to_L, output_idle) -> s7a;

@s7a if (enc) then (key_lsh2) -> s7b;
     else (key_rsh2) -> s7b;
@s7b (key_perm2, expand_R) -> s7c;
```

```
@s7c (k_xor_E) -> s7d;
@s7d (sbox, R_to_L, output_idle) -> s8a;

@s8a if (enc) then (key_lsh2) -> s8b;
     else (key_rsh2) -> s8b;
@s8b (key_perm2, expand_R) -> s8c;
@s8c (k_xor_E) -> s8d;
@s8d (sbox, R_to_L, output_idle) -> s9a;

@s9a if (enc) then (key_lsh1) -> s9b;
     else (key_rsh1) -> s9b;
@s9b (key_perm2, expand_R) -> s9c;
@s9c (k_xor_E) -> s9d;
@s9d (sbox, R_to_L, output_idle) -> s10a;

@s10a if (enc) then (key_lsh2) -> s10b;
      else (key_rsh2) -> s10b;
@s10b (key_perm2, expand_R) -> s10c;
@s10c (k_xor_E) -> s10d;
@s10d (sbox, R_to_L, output_idle) -> s11a;

@s11a if (enc) then (key_lsh2) -> s11b;
      else (key_rsh2) -> s11b;
@s11b (key_perm2, expand_R) -> s11c;
@s11c (k_xor_E) -> s11d;
@s11d (sbox, R_to_L, output_idle) -> s12a;

@s12a if (enc) then (key_lsh2) -> s12b;
      else (key_rsh2) -> s12b;
@s12b (key_perm2, expand_R) -> s12c;
@s12c (k_xor_E) -> s12d;
@s12d (sbox, R_to_L, output_idle) -> s13a;

@s13a if (enc) then (key_lsh2) -> s13b;
      else (key_rsh2) -> s13b;
@s13b (key_perm2, expand_R) -> s13c;
@s13c (k_xor_E) -> s13d;
@s13d (sbox, R_to_L, output_idle) -> s14a;

@s14a if (enc) then (key_lsh2) -> s14b;
      else (key_rsh2) -> s14b;
@s14b (key_perm2, expand_R) -> s14c;
@s14c (k_xor_E) -> s14d;
@s14d (sbox, R_to_L, output_idle) -> s15a;

@s15a if (enc) then (key_lsh2) -> s15b;
      else (key_rsh2) -> s15b;
@s15b (key_perm2, expand_R) -> s15c;
@s15c (k_xor_E) -> s15d;
```

```
   @s15d (sbox, R_to_L, output_idle) -> s16a;

   @s16a if (enc) then (key_lsh1) -> s16b;
         else (key_rsh1) -> s16b;
   @s16b (key_perm2, expand_R) -> s16c;
   @s16c (k_xor_E) -> s16d;
   @s16d (sbox, R_to_L, output_idle) -> s17;

   @s17 if (desRound==0)
          then (read_key2, inc_desround, inv_enc, xchangeLR,
              output_idle) -> s0b;
        else if (desRound==1)
          then (read_key3, inc_desround, inv_enc, xchangeLR,
              output_idle) -> s0b;
        else (inv_ip) -> s17;
}

dp testbenchDP(out input_text, key1, key2, key3 : ns(64);
               out encr : ns(1))
{
   sfg run
   {
//       input_text = 0x0123456789ABCDEF;  //for encryption
      input_text = 0x85E813540F0AB405;  //for decryption
      key1 = 0x133457799BBCDFF1;
      key2 = 0x133457799BBCDFF1;
      key3 = 0x133457799BBCDFF1;
      encr = 0;
   }
}

hardwired test(testbenchDP) {run;}

system S
{
    tripledesDP(input_textS, k1S, k2S, k3S, enS, output_textS
       , doneS);
    testbenchDP(input_textS, k1S, k2S, k3S, enS);
}
```

Appendix D

AES(sym) in GEZEL

```
   tabsize
//Implementation of AES (128 bit key)
//
//Run time is 83 cycles to encrypt
//            264 cycles to decrypt
//                      (due to an extensive inverse
   mixcolumn procedure)

dp aes128DP(in in_t : ns(128);  //text to en-/decrypt
         in key : ns(128);  //the key
         in encr : ns(1);   //0 = decrypt, 1 = encrypt
         out out_t : ns(128);  //the output text
         out done : ns(1))  //0 = out_t not ready, 1 = out_t
            ready
{

   lookup sbox : ns(8) =
{ 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,
    43, 254, 215, 171,
118, 202, 130, 201, 125, 250,  89,
    71, 240, 173, 212, 162, 175, 156, 164,
114, 192, 183, 253, 147,  38,  54,  63, 247, 204,
    52, 165, 229, 241, 113,
216,  49,  21,   4, 199,  35, 195,  24, 150,   5, 154,   7,
    18, 128, 226,
235,  39, 178, 117,   9, 131,  44,  26,  27, 110,  90, 160,
    82,  59, 214,
179,  41, 227,  47, 132,  83, 209,   0, 237,  32, 252, 177,
    91, 106, 203,
190,  57,  74,  76,  88, 207, 208, 239, 170, 251,  67,  77,
    51, 133,  69,
249,   2, 127,  80,  60, 159, 168,  81, 163,
    64, 143, 146, 157,  56, 245,
```

```
188, 182, 218,  33,  16, 255, 243, 210, 205,  12,  19, 236,
    95, 151,  68,
23,  196, 167, 126,  61, 100,  93,  25, 115,  96, 129,
    79, 220,  34,  42,
144, 136,  70, 238, 184,  20, 222,  94,  11, 219, 224,  50,
    58,  10,  73,
  6,  36,  92, 194, 211, 172,
      98, 145, 149, 228, 121, 231, 200,  55, 109,
141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,
     8, 186, 120,  37,
 46,  28, 166, 180, 198, 232, 221, 116,  31,
     75, 189, 139, 138, 112,  62,
181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,
    29, 158, 225,
248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,
    85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,
    15, 176,  84, 187,
 22};

   lookup invsbox : ns(8) =
{82,   9, 106, 213,  48,  54, 165,  56, 191,
    64, 163, 158, 129, 243, 215,
251, 124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,
    68, 196, 222,
233, 203,  84, 123, 148,  50, 166, 194,  35,  61, 238,
    76, 149,  11,  66,
250, 195,  78,   8,  46, 161, 102,  40, 217,  36, 178, 118,
    91, 162,  73,
109, 139, 209,  37, 114, 248, 246, 100, 134, 104, 152,
    22, 212, 164,  92,
204,  93, 101, 182, 146, 108, 112,  72,
    80, 253, 237, 185, 218,  94,  21,
 70,  87, 167, 141, 157, 132, 144, 216, 171,
     0, 140, 188, 211,  10, 247,
228,  88,   5, 184, 179,  69,   6, 208,  44,  30, 143, 202,
    63,  15,   2,
193, 175, 189,   3,   1,  19, 138, 107,  58, 145,  17,  65,
    79, 103, 220,
234, 151, 242, 207, 206, 240, 180, 230, 115, 150, 172, 116,
    34, 231, 173,
 53, 133, 226, 249,  55, 232,  28, 117, 223, 110,  71, 241,
     26, 113,  29,
 41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27, 252,
     86,  62,  75,
198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,
    31, 221, 168,
 51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,
     95,  96,  81,
```

```
127, 169,  25, 181,  74,  13,
    45, 229, 122, 159, 147, 201, 156, 239, 160,
224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,
    83, 153,  97,
 23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,
     85,  33,  12,
125};

   reg enc : ns(1);        //0 = decrypt, 1 = encrypt
   reg wtmp : ns(32);      //tmp used in key expansion
   reg rcon : ns(32);      //var used in key expansion
   reg k : ns(128);        //the currently expanded key (newest
       4 words only)
   reg s : ns(128);        //the (manipulated) text, the state
   reg roundno : ns(5);  //keeps track of the round number
   reg mixreg, mixtos : ns(32);  //contain bytes used in
      mixcolumns

   sig stmpFromSbox, swtmp, ssbitmp: ns(32);
   sig sb0i, sb1i, sb2i, sb3i : ns(8); //for kxpand
   sig sb0o, sb1o, sb2o, sb3o : ns(8); //for kxpand
   sig s0i, s1i, s2i, s3i, s4i, s5i, s6i, s7i,
       s8i, s9i, s10i, s11i, s12i, s13i, s14i, s15i : ns(8)
           ; //for subbytes
   sig s0o, s1o, s2o, s3o, s4o, s5o, s6o, s7o,
       s8o, s9o, s10o, s11o, s12o, s13o, s14o, s15o : ns(8)
           ; //for subbytes
   sig ss0, ss1, ss2, ss3 : ns(8);  //for mixcolumns
   sig si0, si1, si2, si3 : ns(8);  //for inv_mixcolumns
   sig si0tmp1 : ns(3);  //for inv_mixcolumns
   sig si0tmp2 : ns(2);  //for inv_mixcolumns

   sfg output_idle
   {
      out_t = 0x00000000000000000000000000000000;
      done = 0;
   }

   sfg get_key
   {
      k = key;
   }

   sfg get_in_t
   {
      s = in_t;
   }

   sfg get_encr
```

```
{
   enc = encr;
}

sfg init_roundno
{
   roundno = 1;
}

sfg inc_roundno
{
   roundno = roundno + 1;
}

sfg init_rcon
{
   rcon = 0x01000000;
}

sfg inv_init_rcon
{
   rcon = 0x6c000000;
}

sfg inv_init_wtmp
{
   wtmp = k[63:32]#k[31:0];
}

sfg shift_k
{
   wtmp = k[127:96];
   k = k[95:0]#k[31:0];
}

sfg inv_shift_k
{
   wtmp = k[31:0];
   k = k[127:32]#k[63:32];
}

sfg adv_Kxpand
{
   sb0i = k[31:24];
   sb0o = sbox(sb0i);
   sb1i = k[23:16];
   sb1o = sbox(sb1i);
   sb2i = k[15:8];
   sb2o = sbox(sb2i);
```

```
   sb3i = k[7:0];
   sb3o = sbox(sb3i);
   stmpFromSbox = sb1o#sb2o#sb3o#sb0o;
   swtmp = stmpFromSbox ^ rcon;
}

sfg next_rcon
{
   rcon = rcon<<1;
}

sfg inv_next_rcon
{
   rcon = rcon>>1;
}

sfg adv_next_rcon
{
   rcon = 0x1b000000;
}

sfg inv_adv_next_rcon
{
   rcon = 0x80000000;
}

sfg simple_Kxpand
{
   swtmp = k[31:0];
}

sfg new_K
{
   k = k[127:32]#(swtmp ^ wtmp);
}

sfg inv_new_K
{
   k = (swtmp ^ wtmp)#k[127:32];
}

sfg s_xor_k
{
   s = s^k;
}

sfg subbytes
{
   s0i = s[127:120];
```

```
    s0o = sbox(s0i);
    s1i = s[119:112];
    s1o = sbox(s1i);
    s2i = s[111:104];
    s2o = sbox(s2i);
    s3i = s[103:96];
    s3o = sbox(s3i);
    s4i = s[95:88];
    s4o = sbox(s4i);
    s5i = s[87:80];
    s5o = sbox(s5i);
    s6i = s[79:72];
    s6o = sbox(s6i);
    s7i = s[71:64];
    s7o = sbox(s7i);
    s8i = s[63:56];
    s8o = sbox(s8i);
    s9i = s[55:48];
    s9o = sbox(s9i);
    s10i = s[47:40];
    s10o = sbox(s10i);
    s11i = s[39:32];
    s11o = sbox(s11i);
    s12i = s[31:24];
    s12o = sbox(s12i);
    s13i = s[23:16];
    s13o = sbox(s13i);
    s14i = s[15:8];
    s14o = sbox(s14i);
    s15i = s[7:0];
    s15o = sbox(s15i);
    s = s0o#s1o#s2o#s3o#s4o#s5o#s6o#s7o#s8o#s9o#s10o#s11o#
        s12o#s13o#s14o#s15o;
}

sfg inv_subbytes
{
    s0i = s[127:120];
    s0o = invsbox(s0i);
    s1i = s[119:112];
    s1o = invsbox(s1i);
    s2i = s[111:104];
    s2o = invsbox(s2i);
    s3i = s[103:96];
    s3o = invsbox(s3i);
    s4i = s[95:88];
    s4o = invsbox(s4i);
    s5i = s[87:80];
    s5o = invsbox(s5i);
```

```
   s6i = s[79:72];
   s6o = invsbox(s6i);
   s7i = s[71:64];
   s7o = invsbox(s7i);
   s8i = s[63:56];
   s8o = invsbox(s8i);
   s9i = s[55:48];
   s9o = invsbox(s9i);
   s10i = s[47:40];
   s10o = invsbox(s10i);
   s11i = s[39:32];
   s11o = invsbox(s11i);
   s12i = s[31:24];
   s12o = invsbox(s12i);
   s13i = s[23:16];
   s13o = invsbox(s13i);
   s14i = s[15:8];
   s14o = invsbox(s14i);
   s15i = s[7:0];
   s15o = invsbox(s15i);
   s = s0o#s1o#s2o#s3o#s4o#s5o#s6o#s7o#s8o#s9o#s10o#s11o#
      s12o#s13o#s14o#s15o;
}

sfg shiftrows
{
   s = s[127:120]#s[87:80]#s[47:40]#s[7:0]#s[95:88]#s
       [55:48]#s[15:8]#s[103:96]#s[63:56]#s[23:16]#s
       [111:104]#s[71:64]#s[31:24]#s[119:112]#s[79:72]#s
       [39:32];
}

sfg inv_shift_rows
{
   s = s[127:120]#s[23:16]#s[47:40]#s[71:64]#s[95:88]#s
       [119:112]#s[15:8]#s[39:32]#s[63:56]#s[87:80]#s
       [111:104]#s[7:0]#s[31:24]#s[55:48]#s[79:72]#s
       [103:96];
}

sfg mixcolumns1
{
   ss0 = s[126:120]<<1^s[118:112]<<1^s[119:112]   ^s
       [111:104]  ^s[103:96]   ^((s[127]^s[119])?0x1b:0);
   ss1 = s[127:120]   ^s[118:112]<<1^s[110:104]<<1^s
       [111:104]  ^s[103:96]   ^((s[119]^s[111])?0x1b:0);
   ss2 = s[127:120]   ^s[119:112]   ^s[110:104]<<1^s
       [102:96]<<1^s[103:96]   ^((s[111]^s[103])?0x1b:0);
   ss3 = s[126:120]<<1^s[127:120]   ^s[119:112]   ^s
```

```
      [111:104]  ^s[102:96]<<1^((s[103]^s[127])?0x1b:0);
   s = ss0#ss1#ss2#ss3#s[95:0];
}

sfg mixcolumns2
{
   ss0 = s[94:88]<<1^s[86:80]<<1^s[87:80]   ^s[79:72]   ^s
      [71:64]   ^((s[95]^s[87])?0x1b:0);
   ss1 = s[95:88]   ^s[86:80]<<1^s[78:72]<<1^s[79:72]   ^s
      [71:64]   ^((s[87]^s[79])?0x1b:0);
   ss2 = s[95:88]   ^s[87:80]   ^s[78:72]<<1^s[70:64]<<1^s
      [71:64]   ^((s[79]^s[71])?0x1b:0);
   ss3 = s[94:88]<<1^s[95:88]   ^s[87:80]   ^s[79:72]   ^s
      [70:64]<<1^((s[71]^s[95])?0x1b:0);
   s = s[127:96]#ss0#ss1#ss2#ss3#s[63:0];
}

sfg mixcolumns3
{
   ss0 = s[62:56]<<1^s[54:48]<<1^s[55:48]   ^s[47:40]   ^s
      [39:32]   ^((s[63]^s[55])?0x1b:0);
   ss1 = s[63:56]   ^s[54:48]<<1^s[46:40]<<1^s[47:40]   ^s
      [39:32]   ^((s[55]^s[47])?0x1b:0);
   ss2 = s[63:56]   ^s[55:48]   ^s[46:40]<<1^s[38:32]<<1^s
      [39:32]   ^((s[47]^s[39])?0x1b:0);
   ss3 = s[62:56]<<1^s[63:56]   ^s[55:48]   ^s[47:40]   ^s
      [38:32]<<1^((s[39]^s[63])?0x1b:0);
   s = s[127:64]#ss0#ss1#ss2#ss3#s[31:0];
}

sfg mixcolumns4
{
   ss0 = s[30:24]<<1^s[22:16]<<1^s[23:16]  ^s[15:8]  ^s
      [7:0]   ^((s[31]^s[23])?0x1b:0);
   ss1 = s[31:24]   ^s[22:16]<<1^s[14:8]<<1^s[15:8]  ^s
      [7:0]   ^((s[23]^s[15])?0x1b:0);
   ss2 = s[31:24]   ^s[23:16]   ^s[14:8]<<1^s[6:0]<<1^s
      [7:0]   ^((s[15]^s[7])?0x1b:0);
   ss3 = s[30:24]<<1^s[31:24]   ^s[23:16]  ^s[15:8]  ^s
      [6:0]<<1^((s[7]^s[31])?0x1b:0);
   s = s[127:32]#ss0#ss1#ss2#ss3;
}

sfg init_inv_mixcolumns1
{
   mixreg = s[127:96];
}

sfg init_inv_mixcolumns2
```

```
{
   mixreg = s[95:64];
}

sfg init_inv_mixcolumns3
{
   mixreg = s[63:32];
}

sfg init_inv_mixcolumns4
{
   mixreg = s[31:0];
}

sfg mixcolumns_to_s1
{
   s = mixtos#s[95:0];
}

sfg mixcolumns_to_s2
{
   s = s[127:96]#mixtos#s[63:0];
}

sfg mixcolumns_to_s3
{
   s = s[127:64]#mixtos#s[31:0];
}

sfg mixcolumns_to_s4
{
   s = s[127:32]#mixtos;
}

sfg inv_mixcolumns
{
   si0tmp1 = mixreg[31:29]^mixreg[23:21]^mixreg[15:13]^
      mixreg[7:5];
   si0tmp2 = mixreg[31:30]^mixreg[15:14];
   si0 = mixreg[30:24]<<1^mixreg[29:24]<<2^mixreg
      [28:24]<<3^
         mixreg[23:16]^mixreg[22:16]<<1^mixreg[20:16]<<3^
         mixreg[15:8]^mixreg[13:8]<<2^mixreg[12:8]<<3^
         mixreg[7:0]^mixreg[4:0]<<3^
         ((mixreg[31]^mixreg[23]^si0tmp1[0]^si0tmp2[0])?0
            x1b:0)^
         ((si0tmp1[1]^si0tmp2[1])?0x36:0)^
         ((si0tmp1[2])?0x6c:0);
   mixtos = mixtos[23:0]#si0;
```

```
      mixreg = mixreg[23:0]#mixreg[31:24];
   }

   sfg output_ready
   {
      out_t = s;
      done = 1;
      $display("finished at ", $dec, $cycle, "   output = ",
         $hex, out_t);
   }
}

fsm aes128FSM(aes128DP)
{
   initial s0a;
   state s0b,
         s1a, s2a, s3a, s4a, s5a, s6a, s7a, s8a, s9a, s10a,
         s11a, s12a, s13a, s14a, s15a,
         s1b, s2b, s3b, s4b, s5b, s6b, s7b, s8b, s9b, s10b,
         s11b, s12b, s13b, s14b, s15b,
         s1c, s2c, s3c, s4c, s5c, s6c, s7c, s8c, s9c, s10c,
         s11c, s12c, s13c, s14c, s15c,
         s1d, s2d, s3d, s4d, s5d, s6d, s7d, s8d, s9d, s10d,
         s11d, s12d, s13d, s14d, s15d,
         s16m, s16n, s16o, s16p, s16q, s16r,
         s2e, s2f, s2g,
         s3e, s3f, s3g,
         s4e, s4f, s4g,
         s5e, s5f, s5g,
         s17;

   @s0a (get_key, get_in_t, get_encr, init_roundno,
      output_idle) -> s0b;
   @s0b if (enc == 0)
        then (inv_init_rcon, inv_init_wtmp, output_idle) ->
           s16m;
        else (init_rcon, shift_k, s_xor_k, output_idle) ->
           s1a;

   @s1a if (enc == 0)
        then if (roundno == 10)
             then (s_xor_k, output_idle) -> s9a;
             else (inc_roundno, s_xor_k, inv_shift_k,
                output_idle) -> s2a;
        else (adv_Kxpand, new_K, subbytes, output_idle) ->
           s2a;

   @s2a if (enc == 0)
        then if ((rcon[28]&rcon[27]) == 0)
```

```
        then (init_inv_mixcolumns1, inv_next_rcon,
            simple_Kxpand, inv_new_K, output_idle) -> s2c
            ;
        else (init_inv_mixcolumns1, inv_adv_next_rcon,
            simple_Kxpand, inv_new_K, output_idle) -> s2c
            ;
    else if (rcon[31] == 0)
        then (next_rcon, shift_k, shiftrows, output_idle
            ) -> s3a;
        else (adv_next_rcon, shift_k, shiftrows,
            output_idle) -> s3a;
@s2c (inv_mixcolumns, output_idle) -> s2d;
@s2d (inv_mixcolumns, output_idle) -> s2e;
@s2e (inv_mixcolumns, output_idle) -> s2f;
@s2f (inv_mixcolumns, output_idle) -> s2g;
@s2g (mixcolumns_to_s1, output_idle) -> s3a;

@s3a if (enc == 0)
    then (init_inv_mixcolumns2, inv_shift_k, output_idle)
        -> s3c;
    else if(roundno == 10)
        then (simple_Kxpand, new_K, output_idle) -> s4b;
        else (simple_Kxpand, new_K, mixcolumns1,
            output_idle) -> s4a;
@s3c (inv_mixcolumns, output_idle) -> s3d;
@s3d (inv_mixcolumns, output_idle) -> s3e;
@s3e (inv_mixcolumns, output_idle) -> s3f;
@s3f (inv_mixcolumns, output_idle) -> s3g;
@s3g (mixcolumns_to_s2, output_idle) -> s4a;

@s4a if (enc == 0)
    then (init_inv_mixcolumns3, simple_Kxpand, inv_new_K
        , output_idle) -> s4c;
    else (shift_k, mixcolumns2, output_idle) -> s5a;
@s4b if (enc == 0)
    then (output_idle) -> s5b;
    else (shift_k, output_idle) -> s5b;
@s4c (inv_mixcolumns, output_idle) -> s4d;
@s4d (inv_mixcolumns, output_idle) -> s4e;
@s4e (inv_mixcolumns, output_idle) -> s4f;
@s4f (inv_mixcolumns, output_idle) -> s4g;
@s4g (mixcolumns_to_s3, output_idle) -> s5a;

@s5a if (enc == 0)
    then (init_inv_mixcolumns4, inv_shift_k, output_idle)
        -> s5c;
    else (simple_Kxpand, new_K, mixcolumns3, output_idle)
        -> s6a;
@s5b if (enc == 0)
```

```
        then (output_idle) -> s6b;
        else (simple_Kxpand, new_K, output_idle) -> s6b;
   @s5c (inv_mixcolumns, output_idle) -> s5d;
   @s5d (inv_mixcolumns, output_idle) -> s5e;
   @s5e (inv_mixcolumns, output_idle) -> s5f;
   @s5f (inv_mixcolumns, output_idle) -> s5g;
   @s5g (mixcolumns_to_s4, output_idle) -> s6a;

   @s6a if (enc == 0)
        then (inv_shift_rows, simple_Kxpand, inv_new_K,
           output_idle) -> s7a;
        else (shift_k, mixcolumns4, output_idle) -> s7a;
   @s6b if (enc == 0)
        then (output_idle) -> s7b;
        else (shift_k, output_idle) -> s7b;

   @s7a if (enc == 0)
        then (inv_subbytes, inv_shift_k, output_idle) -> s8a;
        else (simple_Kxpand, new_K, output_idle) -> s8a;
   @s7b if (enc == 0)
        then (output_idle) -> s8b;
        else (simple_Kxpand, new_K, output_idle) -> s8b;

   @s8a if (enc == 0)
        then (adv_Kxpand, inv_new_K, output_idle) -> s1a;
        else (shift_k, s_xor_k, inc_roundno, output_idle) ->
           s1a;
   @s8b if (enc == 0)
        then (output_idle) -> s9a;
        else (s_xor_k, output_idle) -> s9a;

   @s9a (output_ready) -> s9a;

   //all the s16-states are for preliminary key expansion
      when decrypting
   @s16m (s_xor_k, output_idle) -> s16n;
   @s16n (inv_shift_k, output_idle) -> s16o;
   @s16o (inv_next_rcon, simple_Kxpand, inv_new_K,
      output_idle) -> s16p;
   @s16p (inv_shift_k, output_idle) -> s16q;
   @s16q (simple_Kxpand, inv_new_K, output_idle) -> s16r;
   @s16r (inv_shift_k, output_idle) -> s6a;

}

dp testbenchDP(out input_text, key : ns(128);
               out encr : ns(1))
{
   sfg run
```

```
   {
      input_text = 0x00112233445566778899aabbccddeeff; //for
          encryption
//       input_text = 0x3243f6a8885a308d313198a2e0370734;  //
   for encryption(2)
//       input_text = 0x69c4e0d86a7b0430d8cdb78070b4c55a;  //
   for decryption
//       input_text = 0x3925841d02dc09fbdc118597196a0b32; //
   for decryption(2)
      key = 0x000102030405060708090a0b0c0d0e0f; //for
          encryption
//       key = 0x13111d7fe3944a17f307a78b4d2b30c5; //for
   decryption
//       key = 0x2b7e151628aed2a6abf7158809cf4f3c; //for
   encryption(2)
//       key = 0xd014f9a8c9ee2589e13f0cc8b6630ca6; //for
   decryption(2)
      encr = 1;
   }
}

hardwired test(testbenchDP) {run;}

system S
{
    aes128DP(input_textS, kS, enS, output_textS, doneS);
    testbenchDP(input_textS, kS, enS);
}
```

Appendix E

AES(asym) in GEZEL

```
   tabsize
//Implementation of AES (128 bit key)
//
//Run time is 83 cycles to encrypt
//            343 cycles to decrypt (due to key expansion
   prior to decryption)
//                             (and an extensive inverse
   mixcolumn procedure)

dp aes128DP(in in_t : ns(128);  //text to en-/decrypt
         in key : ns(128);  //the key
         in encr : ns(1);   //0 = decrypt, 1 = encrypt
         out out_t : ns(128);  //the output text
         out done : ns(1))  //0 = out_t not ready, 1 = out_t
            ready
{

   lookup sbox : ns(8) =
{ 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,
    43, 254, 215, 171,
118, 202, 130, 201, 125, 250,  89,
    71, 240, 173, 212, 162, 175, 156, 164,
114, 192, 183, 253, 147,  38,  54,  63, 247, 204,
    52, 165, 229, 241, 113,
216,  49,  21,   4, 199,  35, 195,  24, 150,   5, 154,   7,
    18, 128, 226,
235,  39, 178, 117,   9, 131,  44,  26,  27, 110,  90, 160,
    82,  59, 214,
179,  41, 227,  47, 132,  83, 209,   0, 237,  32, 252, 177,
    91, 106, 203,
190,  57,  74,  76,  88, 207, 208, 239, 170, 251,  67,  77,
    51, 133,  69,
249,   2, 127,  80,  60, 159, 168,  81, 163,
```

```
    64, 143, 146, 157,  56, 245,
188, 182, 218,  33,  16, 255, 243, 210, 205,  12,  19, 236,
    95, 151,  68,
23,  196, 167, 126,  61, 100,  93,  25, 115,  96, 129,
    79, 220,  34,  42,
144, 136,  70, 238, 184,  20, 222,  94,  11, 219, 224,  50,
    58,  10,  73,
  6,  36,  92, 194, 211, 172,
     98, 145, 149, 228, 121, 231, 200,  55, 109,
141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,
     8, 186, 120,  37,
 46,  28, 166, 180, 198, 232, 221, 116,  31,
     75, 189, 139, 138, 112,  62,
181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,
    29, 158, 225,
248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,
    85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,
    15, 176,  84, 187,
 22};

   lookup invsbox : ns(8) =
{82,   9, 106, 213,  48,  54, 165,  56, 191,
    64, 163, 158, 129, 243, 215,
251, 124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,
    68, 196, 222,
233, 203,  84, 123, 148,  50, 166, 194,  35,  61, 238,
    76, 149,  11,  66,
250, 195,  78,   8,  46, 161, 102,  40, 217,  36, 178, 118,
    91, 162,  73,
109, 139, 209,  37, 114, 248, 246, 100, 134, 104, 152,
    22, 212, 164,  92,
204,  93, 101, 182, 146, 108, 112,  72,
    80, 253, 237, 185, 218,  94,  21,
 70,  87, 167, 141, 157, 132, 144, 216, 171,
     0, 140, 188, 211,  10, 247,
228,  88,   5, 184, 179,  69,   6, 208,  44,  30, 143, 202,
    63,  15,   2,
193, 175, 189,   3,   1,  19, 138, 107,  58, 145,  17,  65,
    79, 103, 220,
234, 151, 242, 207, 206, 240, 180, 230, 115, 150, 172, 116,
    34, 231, 173,
 53, 133, 226, 249,  55, 232,  28, 117, 223, 110,  71, 241,
     26, 113,  29,
 41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27, 252,
     86,  62,  75,
198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,
    31, 221, 168,
 51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,
```

```
    95,  96,  81,
127, 169,  25, 181,  74,  13,
    45, 229, 122, 159, 147, 201, 156, 239, 160,
224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,
    83, 153,  97,
 23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,
    85,  33,  12,
125};

   reg enc : ns(1);       //0 = decrypt, 1 = encrypt
   reg wtmp : ns(32);     //tmp used in key expansion
   reg rcon : ns(32);     //var used in key expansion
   reg k : ns(128);       //the currently expanded key (newest
       4 words only)
   reg s : ns(128);       //the (manipulated) text, the state
   reg roundno : ns(5);  //keeps track of the round number
   reg mixreg, mixtos : ns(32);  //contain bytes used in
      mixcolumns

   sig stmpFromSbox, swtmp, ssbitmp: ns(32);
   sig sb0i, sb1i, sb2i, sb3i : ns(8); //for kxpand
   sig sb0o, sb1o, sb2o, sb3o : ns(8); //for kxpand
   sig s0i, s1i, s2i, s3i, s4i, s5i, s6i, s7i,
       s8i, s9i, s10i, s11i, s12i, s13i, s14i, s15i : ns(8)
           ; //for subbytes
   sig s0o, s1o, s2o, s3o, s4o, s5o, s6o, s7o,
       s8o, s9o, s10o, s11o, s12o, s13o, s14o, s15o : ns(8)
           ; //for subbytes
   sig ss0, ss1, ss2, ss3 : ns(8);  //for mixcolumns
   sig si0, si1, si2, si3 : ns(8);  //for inv_mixcolumns
   sig si0tmp1 : ns(3);  //for inv_mixcolumns
   sig si0tmp2 : ns(2);  //for inv_mixcolumns

   sfg output_idle
   {
      out_t = 0x00000000000000000000000000000000;
      done = 0;
   }

   sfg get_key
   {
      k = key;
   }

   sfg get_in_t
   {
      s = in_t;
   }
```

```
sfg get_encr
{
   enc = encr;
}

sfg init_roundno
{
   roundno = 1;
}

sfg inc_roundno
{
   roundno = roundno + 1;
}

sfg init_rcon
{
   rcon = 0x01000000;
}

sfg shift_k
{
   wtmp = k[127:96];
   k = k[95:0]#k[31:0];
}

sfg inv_shift_k
{
   wtmp = k[31:0];
   k = k[127:32]#k[63:32];
}

sfg adv_Kxpand
{
   sb0i = k[31:24];
   sb0o = sbox(sb0i);
   sb1i = k[23:16];
   sb1o = sbox(sb1i);
   sb2i = k[15:8];
   sb2o = sbox(sb2i);
   sb3i = k[7:0];
   sb3o = sbox(sb3i);
   stmpFromSbox = sb1o#sb2o#sb3o#sb0o;
   swtmp = stmpFromSbox ^ rcon;
}

sfg next_rcon
{
   rcon = rcon<<1;
```

```
}

sfg inv_next_rcon
{
   rcon = rcon>>1;
}

sfg adv_next_rcon
{
   rcon = 0x1b000000;
}

sfg inv_adv_next_rcon
{
   rcon = 0x80000000;
}

sfg simple_Kxpand
{
   swtmp = k[31:0];
}

sfg new_K
{
   k = k[127:32]#(swtmp ^ wtmp);
}

sfg inv_new_K
{
   k = (swtmp ^ wtmp)#k[127:32];
}

sfg s_xor_k
{
   s = s^k;
}

sfg subbytes
{
   s0i = s[127:120];
   s0o = sbox(s0i);
   s1i = s[119:112];
   s1o = sbox(s1i);
   s2i = s[111:104];
   s2o = sbox(s2i);
   s3i = s[103:96];
   s3o = sbox(s3i);
   s4i = s[95:88];
   s4o = sbox(s4i);
```

```
   s5i = s[87:80];
   s5o = sbox(s5i);
   s6i = s[79:72];
   s6o = sbox(s6i);
   s7i = s[71:64];
   s7o = sbox(s7i);
   s8i = s[63:56];
   s8o = sbox(s8i);
   s9i = s[55:48];
   s9o = sbox(s9i);
   s10i = s[47:40];
   s10o = sbox(s10i);
   s11i = s[39:32];
   s11o = sbox(s11i);
   s12i = s[31:24];
   s12o = sbox(s12i);
   s13i = s[23:16];
   s13o = sbox(s13i);
   s14i = s[15:8];
   s14o = sbox(s14i);
   s15i = s[7:0];
   s15o = sbox(s15i);
   s = s0o#s1o#s2o#s3o#s4o#s5o#s6o#s7o#s8o#s9o#s10o#s11o#
      s12o#s13o#s14o#s15o;
}

sfg inv_subbytes
{
   s0i = s[127:120];
   s0o = invsbox(s0i);
   s1i = s[119:112];
   s1o = invsbox(s1i);
   s2i = s[111:104];
   s2o = invsbox(s2i);
   s3i = s[103:96];
   s3o = invsbox(s3i);
   s4i = s[95:88];
   s4o = invsbox(s4i);
   s5i = s[87:80];
   s5o = invsbox(s5i);
   s6i = s[79:72];
   s6o = invsbox(s6i);
   s7i = s[71:64];
   s7o = invsbox(s7i);
   s8i = s[63:56];
   s8o = invsbox(s8i);
   s9i = s[55:48];
   s9o = invsbox(s9i);
   s10i = s[47:40];
```

```
    s10o = invsbox(s10i);
    s11i = s[39:32];
    s11o = invsbox(s11i);
    s12i = s[31:24];
    s12o = invsbox(s12i);
    s13i = s[23:16];
    s13o = invsbox(s13i);
    s14i = s[15:8];
    s14o = invsbox(s14i);
    s15i = s[7:0];
    s15o = invsbox(s15i);
    s = s0o#s1o#s2o#s3o#s4o#s5o#s6o#s7o#s8o#s9o#s10o#s11o#
        s12o#s13o#s14o#s15o;
}

sfg shiftrows
{
    s = s[127:120]#s[87:80]#s[47:40]#s[7:0]#s[95:88]#s
        [55:48]#s[15:8]#s[103:96]#s[63:56]#s[23:16]#s
        [111:104]#s[71:64]#s[31:24]#s[119:112]#s[79:72]#s
        [39:32];
}

sfg inv_shift_rows
{
    s = s[127:120]#s[23:16]#s[47:40]#s[71:64]#s[95:88]#s
        [119:112]#s[15:8]#s[39:32]#s[63:56]#s[87:80]#s
        [111:104]#s[7:0]#s[31:24]#s[55:48]#s[79:72]#s
        [103:96];
}

sfg mixcolumns1
{
    ss0 = s[126:120]<<1^s[118:112]<<1^s[119:112]    ^s
        [111:104]  ^s[103:96]    ^((s[127]^s[119])?0x1b:0);
    ss1 = s[127:120]    ^s[118:112]<<1^s[110:104]<<1^s
        [111:104]  ^s[103:96]    ^((s[119]^s[111])?0x1b:0);
    ss2 = s[127:120]    ^s[119:112]    ^s[110:104]<<1^s
        [102:96]<<1^s[103:96]    ^((s[111]^s[103])?0x1b:0);
    ss3 = s[126:120]<<1^s[127:120]    ^s[119:112]    ^s
        [111:104]  ^s[102:96]<<1^((s[103]^s[127])?0x1b:0);
    s = ss0#ss1#ss2#ss3#s[95:0];
}

sfg mixcolumns2
{
    ss0 = s[94:88]<<1^s[86:80]<<1^s[87:80]    ^s[79:72]    ^s
        [71:64]    ^((s[95]^s[87])?0x1b:0);
    ss1 = s[95:88]    ^s[86:80]<<1^s[78:72]<<1^s[79:72]    ^s
```

```
      [71:64]   ^((s[87]^s[79])?0x1b:0);
   ss2 = s[95:88]   ^s[87:80]   ^s[78:72]<<1^s[70:64]<<1^s
      [71:64]   ^((s[79]^s[71])?0x1b:0);
   ss3 = s[94:88]<<1^s[95:88]   ^s[87:80]   ^s[79:72]   ^s
      [70:64]<<1^((s[71]^s[95])?0x1b:0);
   s = s[127:96]#ss0#ss1#ss2#ss3#s[63:0];
}

sfg mixcolumns3
{
   ss0 = s[62:56]<<1^s[54:48]<<1^s[55:48]   ^s[47:40]   ^s
      [39:32]   ^((s[63]^s[55])?0x1b:0);
   ss1 = s[63:56]   ^s[54:48]<<1^s[46:40]<<1^s[47:40]   ^s
      [39:32]   ^((s[55]^s[47])?0x1b:0);
   ss2 = s[63:56]   ^s[55:48]   ^s[46:40]<<1^s[38:32]<<1^s
      [39:32]   ^((s[47]^s[39])?0x1b:0);
   ss3 = s[62:56]<<1^s[63:56]   ^s[55:48]   ^s[47:40]   ^s
      [38:32]<<1^((s[39]^s[63])?0x1b:0);
   s = s[127:64]#ss0#ss1#ss2#ss3#s[31:0];
}

sfg mixcolumns4
{
   ss0 = s[30:24]<<1^s[22:16]<<1^s[23:16]  ^s[15:8]  ^s
      [7:0]   ^((s[31]^s[23])?0x1b:0);
   ss1 = s[31:24]   ^s[22:16]<<1^s[14:8]<<1^s[15:8]  ^s
      [7:0]   ^((s[23]^s[15])?0x1b:0);
   ss2 = s[31:24]   ^s[23:16]   ^s[14:8]<<1^s[6:0]<<1^s
      [7:0]   ^((s[15]^s[7])?0x1b:0);
   ss3 = s[30:24]<<1^s[31:24]   ^s[23:16]  ^s[15:8]  ^s
      [6:0]<<1^((s[7]^s[31])?0x1b:0);
   s = s[127:32]#ss0#ss1#ss2#ss3;
}

sfg init_inv_mixcolumns1
{
   mixreg = s[127:96];
}

sfg init_inv_mixcolumns2
{
   mixreg = s[95:64];
}

sfg init_inv_mixcolumns3
{
   mixreg = s[63:32];
}
```

```
sfg init_inv_mixcolumns4
{
   mixreg = s[31:0];
}

sfg mixcolumns_to_s1
{
   s = mixtos#s[95:0];
}

sfg mixcolumns_to_s2
{
   s = s[127:96]#mixtos#s[63:0];
}

sfg mixcolumns_to_s3
{
   s = s[127:64]#mixtos#s[31:0];
}

sfg mixcolumns_to_s4
{
   s = s[127:32]#mixtos;
}

sfg inv_mixcolumns
{
   si0tmp1 = mixreg[31:29]^mixreg[23:21]^mixreg[15:13]^
      mixreg[7:5];
   si0tmp2 = mixreg[31:30]^mixreg[15:14];
   si0 = mixreg[30:24]<<1^mixreg[29:24]<<2^mixreg
      [28:24]<<3^
         mixreg[23:16]^mixreg[22:16]<<1^mixreg[20:16]<<3^
         mixreg[15:8]^mixreg[13:8]<<2^mixreg[12:8]<<3^
         mixreg[7:0]^mixreg[4:0]<<3^
         ((mixreg[31]^mixreg[23]^si0tmp1[0]^si0tmp2[0])?0
            x1b:0)^
         ((si0tmp1[1]^si0tmp2[1])?0x36:0)^
         ((si0tmp1[2])?0x6c:0);
   mixtos = mixtos[23:0]#si0;
   mixreg = mixreg[23:0]#mixreg[31:24];
}

sfg output_ready
{
   out_t = s;
   done = 1;
   $display("finished at ", $dec, $cycle, "   output = ",
      $hex, out_t);
```

```
   }
}

fsm aes128FSM(aes128DP)
{
   initial s0a;
   state s0b,
         s1a,
         s2a, s2c, s2d, s2e, s2f, s2g,
         s3a, s3c, s3d, s3e, s3f, s3g,
         s4a, s4b, s4c, s4d, s4e, s4f, s4g,
         s5a, s5b, s5c, s5d, s5e, s5f, s5g,
         s6a, s6b,
         s7a, s7b,
         s8a, s8b,
         s9a,
         s16a, s16b, s16c, s16d, s16e, s16f, s16g, s16h, s16i
            ,
         s16j, s16k, s16l, s16m, s16n, s16o, s16p, s16q, s16r
            ;

   @s0a (get_key, get_in_t, get_encr, init_roundno,
      output_idle) -> s0b;
   @s0b if (enc == 0)
        then (init_rcon, shift_k, output_idle) -> s16a;
        else (init_rcon, shift_k, s_xor_k, output_idle) ->
           s1a;

   @s1a if (enc == 0)
        then if (roundno == 10)
             then (s_xor_k, output_idle) -> s9a;
             else (inc_roundno, s_xor_k, inv_shift_k,
                output_idle) -> s2a;
        else (adv_Kxpand, new_K, subbytes, output_idle) ->
           s2a;

   @s2a if (enc == 0)
        then if ((rcon[28]&rcon[27]) == 0)
             then (init_inv_mixcolumns1, inv_next_rcon,
                simple_Kxpand, inv_new_K, output_idle) -> s2c
                ;
             else (init_inv_mixcolumns1, inv_adv_next_rcon,
                simple_Kxpand, inv_new_K, output_idle) -> s2c
                ;
        else if (rcon[31] == 0)
             then (next_rcon, shift_k, shiftrows, output_idle
                ) -> s3a;
             else (adv_next_rcon, shift_k, shiftrows,
                output_idle) -> s3a;
```

```
@s2c (inv_mixcolumns, output_idle) -> s2d;
@s2d (inv_mixcolumns, output_idle) -> s2e;
@s2e (inv_mixcolumns, output_idle) -> s2f;
@s2f (inv_mixcolumns, output_idle) -> s2g;
@s2g (mixcolumns_to_s1, output_idle) -> s3a;

@s3a if (enc == 0)
     then (init_inv_mixcolumns2, inv_shift_k, output_idle)
          -> s3c;
     else if(roundno == 10)
           then (simple_Kxpand, new_K, output_idle) -> s4b;
           else (simple_Kxpand, new_K, mixcolumns1,
               output_idle) -> s4a;
@s3c (inv_mixcolumns, output_idle) -> s3d;
@s3d (inv_mixcolumns, output_idle) -> s3e;
@s3e (inv_mixcolumns, output_idle) -> s3f;
@s3f (inv_mixcolumns, output_idle) -> s3g;
@s3g (mixcolumns_to_s2, output_idle) -> s4a;

@s4a if (enc == 0)
     then (init_inv_mixcolumns3, simple_Kxpand, inv_new_K
        , output_idle) -> s4c;
     else (shift_k, mixcolumns2, output_idle) -> s5a;
@s4b if (enc == 0)
     then (output_idle) -> s5b;
     else (shift_k, output_idle) -> s5b;
@s4c (inv_mixcolumns, output_idle) -> s4d;
@s4d (inv_mixcolumns, output_idle) -> s4e;
@s4e (inv_mixcolumns, output_idle) -> s4f;
@s4f (inv_mixcolumns, output_idle) -> s4g;
@s4g (mixcolumns_to_s3, output_idle) -> s5a;

@s5a if (enc == 0)
     then (init_inv_mixcolumns4, inv_shift_k, output_idle)
          -> s5c;
     else (simple_Kxpand, new_K, mixcolumns3, output_idle)
          -> s6a;
@s5b if (enc == 0)
     then (output_idle) -> s6b;
     else (simple_Kxpand, new_K, output_idle) -> s6b;
@s5c (inv_mixcolumns, output_idle) -> s5d;
@s5d (inv_mixcolumns, output_idle) -> s5e;
@s5e (inv_mixcolumns, output_idle) -> s5f;
@s5f (inv_mixcolumns, output_idle) -> s5g;
@s5g (mixcolumns_to_s4, output_idle) -> s6a;

@s6a if (enc == 0)
     then (inv_shift_rows, simple_Kxpand, inv_new_K,
        output_idle) -> s7a;
```

```
        else (shift_k, mixcolumns4, output_idle) -> s7a;
    @s6b if (enc == 0)
        then (output_idle) -> s7b;
        else (shift_k, output_idle) -> s7b;

    @s7a if (enc == 0)
        then (inv_subbytes, inv_shift_k, output_idle) -> s8a;
        else (simple_Kxpand, new_K, output_idle) -> s8a;
    @s7b if (enc == 0)
        then (output_idle) -> s8b;
        else (simple_Kxpand, new_K, output_idle) -> s8b;

    @s8a if (enc == 0)
        then (adv_Kxpand, inv_new_K, output_idle) -> s1a;
        else (shift_k, s_xor_k, inc_roundno, output_idle) ->
           s1a;
    @s8b if (enc == 0)
        then (output_idle) -> s9a;
        else (s_xor_k, output_idle) -> s9a;

    @s9a (output_ready) -> s9a;

    //all the s16-states are for preliminary key expansion
       when decrypting
    @s16a (adv_Kxpand, new_K, output_idle) -> s16b;
    @s16b if (rcon[31] == 0)
          then (next_rcon, shift_k, output_idle) -> s16c;
          else (adv_next_rcon, shift_k, output_idle) -> s16c;
    @s16c if(roundno == 10)
          then (simple_Kxpand, new_K, output_idle) -> s16i;
          else (simple_Kxpand, new_K, output_idle) -> s16d;
    @s16d (inc_roundno, shift_k, output_idle) -> s16e;
    @s16e (simple_Kxpand, new_K, output_idle) -> s16f;
    @s16f (shift_k, output_idle) -> s16g;
    @s16g (simple_Kxpand, new_K, output_idle) -> s16h;
    @s16h (shift_k, output_idle) -> s16a;
    @s16i (init_roundno, shift_k, output_idle) -> s16j;
    @s16j (simple_Kxpand, new_K, output_idle) -> s16k;
    @s16k (shift_k, output_idle) -> s16l;
    @s16l (simple_Kxpand, new_K, output_idle) -> s16m;
    @s16m (s_xor_k, output_idle) -> s16n;
    @s16n (inv_shift_k, output_idle) -> s16o;
    @s16o (inv_next_rcon, simple_Kxpand, inv_new_K,
       output_idle) -> s16p;
    @s16p (inv_shift_k, output_idle) -> s16q;
    @s16q (simple_Kxpand, inv_new_K, output_idle) -> s16r;
    @s16r (inv_shift_k, output_idle) -> s6a;

}
```

```
dp testbenchDP(out input_text, key : ns(128);
               out encr : ns(1))
{
   sfg run
   {
//      input_text = 0x00112233445566778899aabbccddeeff; //
   for encryption
//      input_text = 0x3243f6a8885a308d313198a2e0370734;  //
   for encryption(2)
      input_text = 0x69c4e0d86a7b0430d8cdb78070b4c55a;  //for
          decryption
//      input_text = 0x3925841d02dc09fbdc118597196a0b32; //
   for decryption(2)
      key = 0x000102030405060708090a0b0c0d0e0f;
//      key = 0x2b7e151628aed2a6abf7158809cf4f3c; //for(2)
      encr = 0;
   }
}

hardwired test(testbenchDP) {run;}

system S
{
    aes128DP(input_textS, kS, enS, output_textS, doneS);
    testbenchDP(input_textS, kS, enS);
}
```

Appendix F

AES(half) in GEZEL

```
   tabsize
//Implementation of AES (128 bit key) - encryption only
//
//Run time is 83 cycles

dp aes128DP(in in_t : ns(128);  //text to en-/decrypt
         in key : ns(128);  //the key used in first "DES
            round"
         out out_t : ns(128);  //the output text
         out done : ns(1))  //0 = out_t not ready, 1 = out_t
            ready
{

   lookup sbox : ns(8) =
{ 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,
    43, 254, 215, 171,
118, 202, 130, 201, 125, 250,  89,
    71, 240, 173, 212, 162, 175, 156, 164,
114, 192, 183, 253, 147,  38,  54,  63, 247, 204,
    52, 165, 229, 241, 113,
216,  49,  21,   4, 199,  35, 195,  24, 150,   5, 154,   7,
    18, 128, 226,
235,  39, 178, 117,   9, 131,  44,  26,  27, 110,  90, 160,
    82,  59, 214,
179,  41, 227,  47, 132,  83, 209,   0, 237,  32, 252, 177,
    91, 106, 203,
190,  57,  74,  76,  88, 207, 208, 239, 170, 251,  67,  77,
    51, 133,  69,
249,   2, 127,  80,  60, 159, 168,  81, 163,
    64, 143, 146, 157,  56, 245,
188, 182, 218,  33,  16, 255, 243, 210, 205,  12,  19, 236,
    95, 151,  68,
23,  196, 167, 126,  61, 100,  93,  25, 115,  96, 129,
```

```
    79, 220,  34,  42,
144, 136,  70, 238, 184,  20, 222,  94,  11, 219, 224,  50,
    58,  10,  73,
  6,  36,  92, 194, 211, 172,
      98, 145, 149, 228, 121, 231, 200,  55, 109,
141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,
     8, 186, 120,  37,
 46,  28, 166, 180, 198, 232, 221, 116,  31,
     75, 189, 139, 138, 112,  62,
181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,
    29, 158, 225,
248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,
    85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,
    15, 176,  84, 187,
 22};

   lookup invsbox : ns(8) =
{82,   9, 106, 213,  48,  54, 165,  56, 191,
    64, 163, 158, 129, 243, 215,
251, 124, 227,  57, 130, 155,  47, 255, 135,  52, 142,  67,
    68, 196, 222,
233, 203,  84, 123, 148,  50, 166, 194,  35,  61, 238,
    76, 149,  11,  66,
250, 195,  78,   8,  46, 161, 102,  40, 217,  36, 178, 118,
    91, 162,  73,
109, 139, 209,  37, 114, 248, 246, 100, 134, 104, 152,
    22, 212, 164,  92,
204,  93, 101, 182, 146, 108, 112,  72,
    80, 253, 237, 185, 218,  94,  21,
 70,  87, 167, 141, 157, 132, 144, 216, 171,
      0, 140, 188, 211,  10, 247,
228,  88,   5, 184, 179,  69,   6, 208,  44,  30, 143, 202,
    63,  15,   2,
193, 175, 189,   3,   1,  19, 138, 107,  58, 145,  17,  65,
    79, 103, 220,
234, 151, 242, 207, 206, 240, 180, 230, 115, 150, 172, 116,
    34, 231, 173,
 53, 133, 226, 249,  55, 232,  28, 117, 223, 110,  71, 241,
     26, 113,  29,
 41, 197, 137, 111, 183,  98,  14, 170,  24, 190,  27, 252,
     86,  62,  75,
198, 210, 121,  32, 154, 219, 192, 254, 120, 205,  90, 244,
    31, 221, 168,
 51, 136,   7, 199,  49, 177,  18,  16,  89,  39, 128, 236,
     95,  96,  81,
127, 169,  25, 181,  74,  13,
    45, 229, 122, 159, 147, 201, 156, 239, 160,
224,  59,  77, 174,  42, 245, 176, 200, 235, 187,  60, 131,
```

```
    83, 153,  97,
 23,  43,   4, 126, 186, 119, 214,  38, 225, 105,  20,  99,
     85,  33,  12,
125};

   reg wtmp : ns(32);    //tmp used in key expansion
   reg rcon : ns(32);    //var used in key expansion
   reg k : ns(128);      //the currently expanded key (newest
       4 words only)
   reg s : ns(128);      //the (manipulated) text, the state
   reg roundno : ns(5);  //keeps track of the round number

   sig stmpFromSbox, swtmp : ns(32);
   sig sb0i, sb1i, sb2i, sb3i : ns(8); //for kxpand
   sig sb0o, sb1o, sb2o, sb3o : ns(8); //for kxpand
   sig s0i, s1i, s2i, s3i, s4i, s5i, s6i, s7i,
       s8i, s9i, s10i, s11i, s12i, s13i, s14i, s15i : ns(8)
           ; //for subbytes
   sig s0o, s1o, s2o, s3o, s4o, s5o, s6o, s7o,
       s8o, s9o, s10o, s11o, s12o, s13o, s14o, s15o : ns(8)
           ; //for subbytes
   sig ss0, ss1, ss2, ss3 : ns(8);  //for mixcolumns

   sfg output_idle
   {
      out_t = 0x00000000000000000000000000000000;
      done = 0;
   }

   sfg get_key
   {
      k = key;
   }

   sfg get_in_t
   {
      s = in_t;
   }

   sfg init_roundno
   {
      roundno = 1;
   }

   sfg inc_roundno
   {
      roundno = roundno + 1;
   }
```

```
sfg init_rcon
{
   rcon = 0x01000000;
}

sfg shift_k
{
   wtmp = k[127:96];
   k = k[95:0]#k[31:0];
}

sfg adv_Kxpand
{
   sb0i = k[31:24];
   sb0o = sbox(sb0i);
   sb1i = k[23:16];
   sb1o = sbox(sb1i);
   sb2i = k[15:8];
   sb2o = sbox(sb2i);
   sb3i = k[7:0];
   sb3o = sbox(sb3i);
   stmpFromSbox = sb1o#sb2o#sb3o#sb0o;
   swtmp = stmpFromSbox ^ rcon;
}

sfg next_rcon
{
   rcon = rcon<<1;
}

sfg adv_next_rcon
{
   rcon = 0x1b000000;
}

sfg simple_Kxpand
{
   swtmp = k[31:0];
}

sfg new_K
{
   k = k[127:32]#(swtmp ^ wtmp);
}

sfg s_xor_k
{
   s = s^k;
}
```

```
sfg subbytes
{
   s0i = s[127:120];
   s0o = sbox(s0i);
   s1i = s[119:112];
   s1o = sbox(s1i);
   s2i = s[111:104];
   s2o = sbox(s2i);
   s3i = s[103:96];
   s3o = sbox(s3i);
   s4i = s[95:88];
   s4o = sbox(s4i);
   s5i = s[87:80];
   s5o = sbox(s5i);
   s6i = s[79:72];
   s6o = sbox(s6i);
   s7i = s[71:64];
   s7o = sbox(s7i);
   s8i = s[63:56];
   s8o = sbox(s8i);
   s9i = s[55:48];
   s9o = sbox(s9i);
   s10i = s[47:40];
   s10o = sbox(s10i);
   s11i = s[39:32];
   s11o = sbox(s11i);
   s12i = s[31:24];
   s12o = sbox(s12i);
   s13i = s[23:16];
   s13o = sbox(s13i);
   s14i = s[15:8];
   s14o = sbox(s14i);
   s15i = s[7:0];
   s15o = sbox(s15i);
   s = s0o#s1o#s2o#s3o#s4o#s5o#s6o#s7o#s8o#s9o#s10o#s11o#
      s12o#s13o#s14o#s15o;
}

sfg shiftrows
{
   s = s[127:120]#s[87:80]#s[47:40]#s[7:0]#s[95:88]#s
       [55:48]#s[15:8]#s[103:96]#s[63:56]#s[23:16]#s
       [111:104]#s[71:64]#s[31:24]#s[119:112]#s[79:72]#s
       [39:32];
}

sfg mixcolumns1
{
```

```
    ss0 = s[126:120]<<1^s[118:112]<<1^s[119:112]   ^s
        [111:104]  ^s[103:96]   ^((s[127]^s[119])?0x1b:0);
    ss1 = s[127:120]   ^s[118:112]<<1^s[110:104]<<1^s
        [111:104]  ^s[103:96]   ^((s[119]^s[111])?0x1b:0);
    ss2 = s[127:120]   ^s[119:112]   ^s[110:104]<<1^s
        [102:96]<<1^s[103:96]   ^((s[111]^s[103])?0x1b:0);
    ss3 = s[126:120]<<1^s[127:120]   ^s[119:112]   ^s
        [111:104]  ^s[102:96]<<1^((s[103]^s[127])?0x1b:0);
    s = ss0#ss1#ss2#ss3#s[95:0];
}

sfg mixcolumns2
{
    ss0 = s[94:88]<<1^s[86:80]<<1^s[87:80]   ^s[79:72]   ^s
        [71:64]   ^((s[95]^s[87])?0x1b:0);
    ss1 = s[95:88]   ^s[86:80]<<1^s[78:72]<<1^s[79:72]   ^s
        [71:64]   ^((s[87]^s[79])?0x1b:0);
    ss2 = s[95:88]   ^s[87:80]   ^s[78:72]<<1^s[70:64]<<1^s
        [71:64]   ^((s[79]^s[71])?0x1b:0);
    ss3 = s[94:88]<<1^s[95:88]   ^s[87:80]   ^s[79:72]   ^s
        [70:64]<<1^((s[71]^s[95])?0x1b:0);
    s = s[127:96]#ss0#ss1#ss2#ss3#s[63:0];
}

sfg mixcolumns3
{
    ss0 = s[62:56]<<1^s[54:48]<<1^s[55:48]   ^s[47:40]   ^s
        [39:32]   ^((s[63]^s[55])?0x1b:0);
    ss1 = s[63:56]   ^s[54:48]<<1^s[46:40]<<1^s[47:40]   ^s
        [39:32]   ^((s[55]^s[47])?0x1b:0);
    ss2 = s[63:56]   ^s[55:48]   ^s[46:40]<<1^s[38:32]<<1^s
        [39:32]   ^((s[47]^s[39])?0x1b:0);
    ss3 = s[62:56]<<1^s[63:56]   ^s[55:48]   ^s[47:40]   ^s
        [38:32]<<1^((s[39]^s[63])?0x1b:0);
    s = s[127:64]#ss0#ss1#ss2#ss3#s[31:0];
}

sfg mixcolumns4
{
    ss0 = s[30:24]<<1^s[22:16]<<1^s[23:16]  ^s[15:8]  ^s
        [7:0]   ^((s[31]^s[23])?0x1b:0);
    ss1 = s[31:24]   ^s[22:16]<<1^s[14:8]<<1^s[15:8]  ^s
        [7:0]   ^((s[23]^s[15])?0x1b:0);
    ss2 = s[31:24]   ^s[23:16]   ^s[14:8]<<1^s[6:0]<<1^s
        [7:0]   ^((s[15]^s[7])?0x1b:0);
    ss3 = s[30:24]<<1^s[31:24]   ^s[23:16]  ^s[15:8]  ^s
        [6:0]<<1^((s[7]^s[31])?0x1b:0);
    s = s[127:32]#ss0#ss1#ss2#ss3;
}
```

```
   sfg output_ready
   {
      out_t = s;
      done = 1;
      $display("finished at ", $dec, $cycle, "    output = ",
         $hex, out_t);
   }
}

fsm aes128FSM(aes128DP)
{
   initial s0a;
   state s0b,
         s1a, s2a, s3a, s4a, s5a, s6a, s7a, s8a, s9a,
         s4b, s5b, s6b, s7b, s8b;

   @s0a (get_key, get_in_t, init_roundno, output_idle) -> s0b
      ;
   @s0b (init_rcon, shift_k, s_xor_k, output_idle) -> s1a;

   @s1a (adv_Kxpand, new_K, subbytes, output_idle) -> s2a;

   @s2a if (rcon[31] == 0)
        then (next_rcon, shift_k, shiftrows, output_idle) ->
           s3a;
        else (adv_next_rcon, shift_k, shiftrows, output_idle)
             -> s3a;

   @s3a if(roundno == 10)
        then (simple_Kxpand, new_K, output_idle) -> s4b;
        else (simple_Kxpand, new_K, mixcolumns1, output_idle)
             -> s4a;

   @s4a (shift_k, mixcolumns2, output_idle) -> s5a;
   @s4b (shift_k, output_idle) -> s5b;

   @s5a (simple_Kxpand, new_K, mixcolumns3, output_idle) ->
      s6a;
   @s5b (simple_Kxpand, new_K, output_idle) -> s6b;

   @s6a (shift_k, mixcolumns4, output_idle) -> s7a;
   @s6b (shift_k, output_idle) -> s7b;

   @s7a (simple_Kxpand, new_K, output_idle) -> s8a;
   @s7b (simple_Kxpand, new_K, output_idle) -> s8b;

   @s8a (shift_k, s_xor_k, inc_roundno, output_idle) -> s1a;
   @s8b (s_xor_k, output_idle) -> s9a;
```

```
   @s9a (output_ready) -> s9a;
}

dp testbenchDP(out input_text, key : ns(128))
{
   sfg run
   {
      input_text = 0x00112233445566778899aabbccddeeff; //for
         encryption
//      0x3243f6a8885a308d313198a2e0370734;  //for encryption
      key = 0x000102030405060708090a0b0c0d0e0f;
//      0x2b7e151628aed2a6abf7158809cf4f3c;
   }
}

hardwired test(testbenchDP) {run;}

system S
{
    aes128DP(input_textS, kS, output_textS, doneS);
    testbenchDP(input_textS, kS);
}
```

Appendix G

The Manually Implementation of XTEA in VHDL

```
   tabsize
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

package types is
        subtype bit_t     is std_logic;
        subtype round_t   is std_logic_vector(4 downto 0);
        subtype word_t    is std_logic_vector(31 downto 0);
        subtype text_t    is std_logic_vector(63 downto 0);
        subtype key_t     is std_logic_vector(127 downto 0);

        type state_t is (s0, s1, s2, s3, s4, s5, s6, s7, s8,
           s9, s10, s15);

        -- (no_rounds +1) cycles are taken in the while-loop
        -- constant used to increment round-register by one
        constant one :round_t := b"00001";

        constant delta :word_t := x"9e3779b9";
        constant minus_delta :word_t := x"61c88647";

        -- constant for initial value of sum when decrypting
        constant inv_sum :word_t := x"c6ef3720";   --  32
           cycles

end types;
```

```
   tabsize
-- implementation of the XTEA algorithm, manually implemented
```

```
LIBRARY IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
use work.types.all;

entity xtea2 is
        port(in_t :in text_t;       -- [v0, v1]  (or [y,z])
             key :in key_t;         -- [k0, k1, k2, k3]
             encr :in bit_t;        -- encr=0    decryption
                                    --      1    encryption
             out_t :out text_t;     -- [new_v0, new_v1]
             done :out bit_t;       -- output valid when done
                 = 1
             RST :in bit_t;         -- reset when RST = 1
             clk :in bit_t);        -- the clock
end xtea2;

architecture structure of xtea2 is

        signal finished :bit_t;
        signal state :state_t;

        component datapath is
                port(in_t :in text_t;
                     key :in key_t;
                     encr :in bit_t;
                     out_t :out text_t;
                     done :out bit_t;
                     fin :out bit_t;
                     state :in state_t;
                     clk :in bit_t);
        end component;

        component controller is
                port (RST :in bit_t;
                      fin :in bit_t;
                      state : out state_t;
                      clk : in bit_t);
        end component;

begin

        ctl:controller port map
                (RST=>RST, fin=>finished, state=>state, clk=>
                   clk);

        dp:datapath port map
                (in_t=>in_t, key=>key, encr=>encr, out_t=>
```

```
                   out_t,
                 done=>done, fin=>finished, state=>state, clk
                    =>clk);

end structure;


   tabsize
-- the controller of XTEA, manually implemented

LIBRARY IEEE;
use IEEE.std_logic_1164.ALL;
use work.types.all;

entity controller is
        port (RST : in bit_t;   -- reset when RST = 1
              fin : in bit_t;   -- 32 cycles done when fin
                  = 1
              state : out state_t;
              clk : in bit_t);  -- the clock
end controller;

architecture behaviour of controller is

        -- "temporary" signals
        signal sstate : state_t;

        -- "registers"
        signal statereg : state_t;
begin
        clkToRegAndOut : process(clk) is
        begin
                if clk'event and clk='1' then
                        if RST='1' then
                                statereg <= s0;
                                state <= s0;
                        else
                                statereg <= sstate;
                                state <= sstate;
                        end if;
                end if;
        end process clkToRegAndOut;

        transition :process(fin, statereg)
        begin
                case statereg is
                when s0 =>
                        sstate <= s1;
```

```
                when s1 =>
                        sstate <= s2;

                when s2 =>
                        if fin = '0' then
                                sstate <= s3;
                        else
                                sstate <= s15;
                        end if;
--                      sstate <= s3;

                when s3 =>
                        sstate <= s4;

                when s4 =>
                        sstate <= s5;

                when s5 =>
                        sstate <= s6;

                when s6 =>
                        sstate <= s7;

                when s7 =>
                        sstate <= s8;

                when s8 =>
                        sstate <= s10;

                when s9 =>
                        sstate <= s10;

                when s10 =>
                        sstate <= s2;

                when s15 =>
                        -- datapath finished, do nothing (
                           reset happens elsewhere!)
                        sstate <= s15;

                when others =>
                        -- undefined states(!) -> goto idle (
                           state = s15)
                        sstate <= s15;
                end case;
        end process;
end behaviour;
```

```
   tabsize
-- the datapath of XTEA, manually implemented

use work.types.all;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- for testing purposes only
--use work.txt_util.all;

entity datapath is
        port(in_t :in text_t;   -- [v0, v1]  (or [y,z])
             key :in key_t;     -- [k0, k1, k2, k3]
             encr :in bit_t;    -- encr=0    decryption
                                --      1    encryption
             out_t :out text_t; -- [new_v0, new_v1]
             done :out bit_t;   -- output valid when done = 1
             fin :out bit_t;    -- 32 cycle done when fin = 1
             state :in state_t;
             clk :in bit_t);    -- the clock
end datapath;

architecture behaviour of datapath is

        -- "temporary" signals
        signal sy, sz :word_t;
        signal sk0, sk1, sk2, sk3 :word_t;
        signal ssum :word_t;
        signal sround : round_t;
        signal sout_t : text_t;
        signal sdone : bit_t;
        signal sstate : state_t;
        signal sfin : bit_t;

        -- "registers"
        signal y, z :word_t;
        signal k0, k1, k2, k3 :word_t;
        signal sum :word_t;
        signal round : round_t;
begin
        clkToRegAndOut : process(clk) is
        begin
                if clk'event and clk='1' then
                        y <= sy;
                        z <= sz;
                        k0 <= sk0;
                        k1 <= sk1;
                        k2 <= sk2;
```

```
                k3 <= sk3;
                sum <= ssum;
                round <= sround;
                out_t <= sout_t;
                done <= sdone;
                sstate <= state;
        end if;
end process clkToRegAndOut;

setFin : process(sfin) is
begin
        fin <= sfin;
end process setFin;

xtea : process(y, z, k0, k1, k2, k3, sum, round,
   sstate) is
begin
        case sstate is
        when s0 =>
        -- idle state
                sdone <= '0';
                sout_t <= x"0000000000000000";
                sfin <= '0';
        -- specify rest of signals and out ports
                sy <= y;
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;

        when s1 =>
        -- load input, and sum and round registers
                sy <= in_t(63 downto 32);
                sz <= in_t(31 downto 0);
                sk0 <= key(127 downto 96);
                sk1 <= key(95 downto 64);
                sk2 <= key(63 downto 32);
                sk3 <= key(31 downto 0);
                sround <= b"00000";
                if encr = '0' then
                        ssum <= inv_sum(31 downto 0);
                elsif encr = '1' then
                        ssum <= x"00000000";
                end if;
        -- specify rest of signals and out ports
                sdone <= '0';
```

```
        sout_t <= x"0000000000000000";
        sfin <= '0';

when s2 =>
        if sfin = '1' then
                sz <= z;
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '1';
                sout_t <= y&z;
                sfin <= '1';
        elsif encr = '0' then
        -- assign new value to z (part 1)
                sz <= z - ((y(27 downto 0)&b
                   "0000") xor (b"00000"&y
                   (31 downto 5)));
        -- specify rest of signals and out
           ports
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        elsif encr = '1' then
        -- assign new value to y (part 1)
                sy <= y + ((z(27 downto 0)&b
                   "0000") xor (b"00000"&z
                   (31 downto 5)));
        -- specify rest of signals and out
           ports
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
```

```
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        end if;

when s3 =>
        if encr = '0' then
        -- assign new value to z (part 2)
                sz <= z - (y xor sum);
        -- specify rest of signals and out
           ports
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        elsif encr = '1' then
        -- assign new value to y (part 2)
                sy <= y + (z xor sum);
        -- specify rest of signals and out
           ports
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        end if;

when s4 =>
        if encr = '0' then
        -- assign new value to z (part 3)
                case sum(12 downto 11) is
                        when "00" =>
                                sz <= z - k0;
                        when "01" =>
                                sz <= z - k1;
                        when "10" =>
```

```
                                sz <= z - k2;
                        when "11" =>
                                sz <= z - k3;
                        when others =>
                end case;
        -- specify rest of signals and out
           ports
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        elsif encr = '1' then
        -- assign new value to y (part 3)
                case sum(1 downto 0) is
                        when "00" =>
                                sy <= y + k0;
                        when "01" =>
                                sy <= y + k1;
                        when "10" =>
                                sy <= y + k2;
                        when "11" =>
                                sy <= y + k3;
                        when others =>
                end case;
        -- specify rest of signals and out
           ports
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        end if;

when s5 =>
-- update sum
        if encr = '0' then
```

```
                ssum <= sum + minus_delta;
        elsif encr = '1' then
                ssum <= sum + delta;
        end if;
-- specify rest of signals and out ports
        sy <= y;
        sz <= z;
        sk0 <= k0;
        sk1 <= k1;
        sk2 <= k2;
        sk3 <= k3;
        sround <= round;
        sdone <= '0';
        sout_t <= x"0000000000000000";
        sfin <= '0';

when s6 =>
        if encr = '0' then
        -- assign new value to y (part 1)
                sy <= y - ((z(27 downto 0)&b
                   "0000") xor (b"00000"&z
                   (31 downto 5)));
        -- specify rest of signals and out
           ports
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        elsif encr = '1' then
        -- assign new value to z (part 1)
                sz <= z + ((y(27 downto 0)&b
                   "0000") xor (b"00000"&y
                   (31 downto 5)));
        -- specify rest of signals and out
           ports
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
```

```
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        end if;

when s7 =>
        if encr = '0' then
        -- assign new value to y (part 2)
                sy <= y - (z xor sum);
        -- specify rest of signals and out
           ports
                sz <= z;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        elsif encr = '1' then
        -- assign new value to z (part 2)
                sz <= z + (y xor sum);
        -- specify rest of signals and out
           ports
                sy <= y;
                sk0 <= k0;
                sk1 <= k1;
                sk2 <= k2;
                sk3 <= k3;
                ssum <= sum;
                sround <= round;
                sdone <= '0';
                sout_t <= x
                   "0000000000000000";
                sfin <= '0';
        end if;

when s8 =>
        if encr = '0' then
        -- assign new value to y (part 3)
                case sum(1 downto 0) is
                        when "00" =>
                                sy <= y - k0;
                        when "01" =>
                                sy <= y - k1;
```

```
                                when "10" =>
                                        sy <= y - k2;
                                when "11" =>
                                        sy <= y - k3;
                                when others =>
                        end case;
                -- specify rest of signals and out
                   ports
                        sz <= z;
                        sk0 <= k0;
                        sk1 <= k1;
                        sk2 <= k2;
                        sk3 <= k3;
                        ssum <= sum;
                        sround <= round;
                        sdone <= '0';
                        sout_t <= x
                           "0000000000000000";
                        sfin <= '0';
                elsif encr = '1' then
                -- assign new value to z (part 3)
                        case sum(12 downto 11) is
                                when "00" =>
                                        sz <= k0 + z;
                                when "01" =>
                                        sz <= k1 + z;
                                when "10" =>
                                        sz <= k2 + z;
                                when "11" =>
                                        sz <= k3 + z;
                                when others =>
                        end case;
                -- specify rest of signals and out
                   ports
                        sy <= y;
                        sk0 <= k0;
                        sk1 <= k1;
                        sk2 <= k2;
                        sk3 <= k3;
                        ssum <= sum;
                        sround <= round;
                        sdone <= '0';
                        sout_t <= x
                           "0000000000000000";
                        sfin <= '0';
                end if;

        when s9 =>
        -- check whether we take another round in the
```

```
    while-loop
        if round = b"11111" then
        -- xtea finished, output result
                sout_t <= y&z;
                sdone <= '1';
                sfin <= '1';
        else
        -- xtea NOT finished, output nothing
                sout_t <= x
                   "0000000000000000";
                sdone <= '0';
                sfin <= '0';
        end if;
-- specify rest of signals and out ports
        sy <= y;
        sz <= z;
        sk0 <= k0;
        sk1 <= k1;
        sk2 <= k2;
        sk3 <= k3;
        ssum <= sum;
        sround <= round;

when s10 =>
-- check whether we take another round in the
    while-loop
        if round = b"11111" then
        -- xtea finished, output result
                sround <= round;
                sout_t <= y&z;
                sdone <= '1';
                sfin <= '1';
        else
        -- xtea NOT finished, output nothing
                sround <= round + one;
                sout_t <= x
                   "0000000000000000";
                sdone <= '0';
                sfin <= '0';
        end if;
-- specify rest of signals and out ports
        sy <= y;
        sz <= z;
        sk0 <= k0;
        sk1 <= k1;
        sk2 <= k2;
        sk3 <= k3;
        ssum <= sum;
```

```
                when s15 =>
                -- specify rest of signals and out ports
                        sy <= y;
                        sz <= z;
                        sk0 <= k0;
                        sk1 <= k1;
                        sk2 <= k2;
                        sk3 <= k3;
                        ssum <= sum;
                        sround <= round;
                        sout_t <= y&z;
                        sdone <= '1';
                        sfin <= '1';

                when others =>
                        sy <= y;
                        sz <= z;
                        sk0 <= k0;
                        sk1 <= k1;
                        sk2 <= k2;
                        sk3 <= k3;
                        ssum <= sum;
                        sround <= round;
                        sdone <= '0';
                        sout_t <= x"0000000000000000";
                        sfin <= '0';
                end case;
        end process xtea;
end behaviour;
```