AES Crypto Module for Contactless Smartcards

Markus Krainz
m_krainz@sbox.tugraz.at

Institute for Applied Information
Processing and Communications (IAIK)
Graz University of Technology
Inffeldgasse 16a
8010 Graz, Austria

Master Thesis

Supervisor: Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Karl-Christian Posch
Assessor: Dipl.-Ing. Dr.techn. Martin Feldhofer

April, 2010

I hereby certify that the work presented in this thesis is my own work and that to the best of my knowledge it is original except where indicated by reference to other authors.

Ich bestätige hiermit, diese Arbeit selbständig verfasst zu haben. Teile der Diplomarbeit, die auf Arbeiten anderer Autoren beruhen, sind durch Angabe der entsprechenden Referenz gekennzeichnet.

Markus Krainz

Acknowledgements

First I would like to thank Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Karl-Christian Posch for giving me the opportunity to write this diploma thesis at the Institute for Applied Information Processing and Communications (IAIK).
I would like to thank Dipl.-Ing. Dr.techn. Martin Feldhofer in particular for his excellent mentoring throughout the entire Master's thesis.

Furthermore, I would like to thank my parents Helga and Walter, my grandmother Maria and my sister Lisa who unremittingly supported me throughout my years of studying.

Abstract

This thesis presents a low-power and small-size AES crypto hardware module adapted for the use in contactless smartcards. The datapath is 32 bits wide. The Advanced Encryption Standard (AES) is the most commonly used symmetric-key algorithm. Encryption and decryption datapaths as well as controlpaths are combined to make the architecture more compact. The proposed architecture supports a key length of 128 bits. The key is calculated on the fly for encryption and decryption. The crypto module can operate either in ECB or CBC mode. Countermeasures against power analysis attacks (dummy cycles and shuffling) were also added to the architecture. The crypto module also provides the possibility to create pseudo-random numbers. Therefore, the stream cipher Grain is implemented. Both algorithms AES and Grain share the same latch-based dual output port RAM to decrease the power and area consumption. The crypto module is easily upgradable with other cryptographic algorithms such as ECC or SHA-1. The implementation of the crypto module requires 14601 GE using 0.35 μm standard cells from austriamicrosystems. The crypto module has an average power consumption of 855 μW when operating in AES ECB encryption mode at a clock frequency of 1 MHz and a supply voltage of 3.3 V.

Keywords: Advanced Encryption Standard, Grain, low-power processor, crypto module, standard cell implementation

Kurzfassung

Diese Masterarbeit beschreibt ein leistungs- und platzsparendes Kryptographiemodul, das geeignet ist, auf Chipkarten betrieben zu werden. Der Datenpfad ist 32 Bits breit. Der Advanced Encryption Standard (AES) ist der am weitest verbreitete Algorithmus für symetrische Verschlüsselung. Um die Architektur kompakter zu machen werden die Datenpfade sowie Kontrollpfade für die Ver- und Entschlüsselung kombiniert. Das Kryptographiemodul kann entweder im ECB- oder CBC-Modus betrieben werden. Es werden auch Gegenmaßnahmen gegen Attacken, die die Leistungsaufnahme des Chips ausnützen, implementiert. Das Kryptographiemodul bietet auch die Möglichkeit Pseudo-Zufallszahlen zu generieren. Dafür wird die Stromchiffre Grain verwendet. AES und Grain teilen sich beide denselben RAM, der zwei Ausgänge hat und aus Latch-Registern besteht, was zu einer verringerten Leistungsaufnahme und einem verkleinerten Platzbedarf führt. Das Kryptographiemodul ist leicht mit anderen kryptographischen Algorithmen wie ECC oder SHA-1 erweiterbar. Die Implementierung des Kryptographiemoduls benötigt 14601 GE unter Verwendung von 0.35 μm Standardzellen der Firma austriamicrosystems. Das Kryptographiemodul hat eine durchschnittliche Leistungsaufnahme von 855 μW, wenn es im AES-ECB-Verschlüsselungsmodus mit einer Taktrate von 1 MHz und einer Versorgungsspannung von 3.3 V betrieben wird.

Stichwörter: Advanced Encryption Standard, Grain, leistungssparender Prozessor, Kryptographie-Modul, Standardzellen-Implementierung

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptography is needed in nearly every aspect of digital life where personal, confidential or sensitive data is processed.

If the secret key of an algorithm is revealed, then even the strongest cryptographic algorithm is useless. Therefore, the pivotal question is where to store the secret key. Personal computers have many weaknesses and are prone to viruses and trojans. The problem is to find a protected environment to store secret keys.

The ideal solution for this task is to use smartcards. They are small enough to be carried in one's wallet but are also capable of performing cryptographic algorithms. Since smartcards have limited energy and area constraints, the main focus is to implement the algorithms for low-power and area consumption.

This work is written with the purpose to design an AES crypto module for contactless smartcards. AES means Advanced Encryption Standard and is in fact the Rijndael algorithm. The architecture is optimized for low power and area consumption. The implementation of the AES also offers countermeasures against power analysis attacks. To be more precise, the specific measures are dummy cycles and shuffling. In addition to the AES algorithm, the crypto module also provides the functionality to generate pseudo-random numbers. For example, it is necessary for the generation of the initialization vector for the AES. The Grain algorithm is taken for this purpose. Therefore, the crypto module features the function of a block as well as a stream cipher. This work is easily upgradable with other cryptographic algorithms like Elliptic Curve Cryptography (ECC) or Secure Hash Algorithm (SHA). A common RAM is used for all algorithms to decrease power and area consumption. Both the datapath and the RAM are 32 bits wide. The APB AMBA is used as the interface for the crypto module.

1.1 Outline

Chapter 2 provides an overview about cryptography. It covers the whole spectrum beginning with symmetric-key algorithms, which either can be stream ciphers or block ciphers. The AES and Grain algorithms are considered more in detail. Public-key algorithms (e.g. Elliptic Curve Cryptography), the SHA and ECDSA are discussed only briefly. The chapter finishes with pseudo-random number generators.

Smartcards are the topic of Chapter 3. This chapter includes contactless smartcards followed by side-channel attacks and countermeasures against them. The last section is about the smartcard components.

Chapter 4 deals with the implementation of the crypto module. First the used design methodology is discussed. Then the architecture which consists of controlpath, datapath, RAM and AMBA interface is described in detail.

Chapter 5 presents the results of the crypto module implemented in $0.35\,\mu m$ AMS standard cell technology. The results are then compared to other related works.

The thesis concludes with Chapter 6, in which suggestions for improvement as well as a summary of both the work done and the results obtained is presented.

Chapter 2

Introduction to Cryptography

The word cryptography is derived from the greek words for "secret" and "write". A message is altered in a way that an unauthorized person is not able to read the original information. The authorized recipient who has the special key can easily read the original message.

In addition to providing confidentiality, cryptography is also used in areas in which one or more of the following properties are required [51]:

Authentication: It should be possible for a receiver of a message to verify its origin. An intruder should not be able to masquerade as someone else.

Integrity: It should be possible for the receiver to verify that the message has not been altered by an attacker.

Nonrepudiation: A sender can not falsely deny that he has sent a message.

At least one of these properties is needed in nearly every domain of modern life and therefore, cryptographic algorithms are an issue of substantial importance. The range of applications which use cryptography begins with smartcards (E-card, cash card, ...), includes communication (mobile phones), personal computers and ends in the codes for launching nuclear weapons.

Shannon [52] claims that any ciphertext of a perfect secret-key cipher does not give any information about the plaintext. Ciphers should provide as much confusion and diffusion as possible. Confusion means that the dependence of the ciphertext statistics and the plaintext statistics are too complicated to be exploited by an attacker. Diffusion refers to the property that every bit of the plaintext and every bit of the key influence as many bits of the ciphertext as possible.

When using cryptography Kerckhoffs' Principle should always be kept in mind. It means that the strength of a cryptographic algorithm must not depend on the secrecy of the algorithm but only on the secrecy of the key. An example of violating this principle happened during the Second World War when the Allies captured the encryption machine ENIGMA of the Germans. After the analysis of ENIGMA, the Allies were able to decrypt most of the german messages, which is today considered to be an important factor contributing to the victory of the war [31].

The strength of a cryptographic algorithm is determined by the number of possible different keys. If an attacker has to try all different keys, then the attack is called a brute force attack. The length of the key should be large enough that a brute force attack is not feasible. A key length of 80 bits results in 2^{80} possible keys, which means when an attacker is able to try 10^9 keys per second it would take $2^{80}/10^9 \approx 38$ million years to complete. Every additional bit of key length doubles the efforts of the attacker [54].

There are two differentiations of key-based algorithms:

- Symmetric
- Public-key

2.1 Outline

The first section of this chapter provides an insight into symmetric-key algorithms where only one key (or a key derived from that one key) is needed for encryption and decryption. Symmetric ciphers could either be stream ciphers (e.g. Grain) or block ciphers (e.g. Advanced Encryption Standard). This section also deals with the modes of operation of block ciphers.

The next section is a short overview about public-key algorithms (e.g. Elliptic Curve Cryptography), in which a separate key is used for encryption and decryption and public key infrastructures are used for managing these keys.

The description of the principle of hash functions follows a section about digital signatures and the Elliptic Curve Digital Signature Algorithm.

The last section discusses random-number generators, which can either be true random-number generators or pseudorandom-number generators.

2.2 Symmetric Algorithms

These algorithms use the same key or a derived key for encryption and decryption. Before the sender and the receiver can communicate securely, it is necessary that both agree on a key. The key must be kept secret as long as the communication needs to remain secretive. Encryption of a symmetric algorithm is denoted by $E_K(M) = C$, while decryption is denoted by $D_K(C) = M$. For symmetric algorithms two categories exist:

- Stream ciphers
- Block ciphers

It is possible to operate a block cipher in stream cipher mode.

2.2.1 Stream Ciphers

Stream ciphers encrypt individual bits of a message one at a time using an encryption transformation which varies with time. The advantages of stream ciphers over block ciphers are that they are generally faster in hardware and are less complex. Stream ciphers are preferable over block ciphers when buffering is limited or when bits must be individually processed as they arrive. Another sizeable advantage is that stream ciphers have limited or no error propagation. The basic component of many stream ciphers are linear feedback shift registers (LFSR). In addition to the advantage of easy implementation, LFSR sequences have good statistical properties, can be produced with large periods and can be easily analyzed with the assistance of algebraic techniques. An LFSR consists of N stages of delay elements E. They are numbered from $0...N-1$ and each element can store one bit. In each clock cycle the bits are shifted in a way that $E_0 = E_1$, $E_1 = E_2$ and so on. The only exception is the last element E_{N-1}, which is the result of a feedback function calculated by adding a fixed subset of $E_0, E_1, ..., E_{N-1}$ of previous states modulo 2. A nonlinear feedback shift register (NFSR) is comparable an LFSR, except that the feedback function is a nonlinear boolean function [39].

Stream ciphers can be classified into synchronous or self-synchronizing ciphers.

Synchronous Stream Ciphers The key-stream is generated independently of the plaintext message and of the ciphertext. Both sender and receiver must be synchronized. The internal state of both the stream cipher of the sender and the receiver must be the same. Most proposed synchronous stream ciphers are binary additive stream ciphers, which means that one bit of the message-stream is encrypted with one bit of the key-stream using the XOR function.

Self-Synchronizing Stream Ciphers On the contrary to synchronous stream ciphers, the cipher stream also depends on a fixed number of previous ciphertexts. That is the reason why such ciphers are able to re-establish the decryption after a loss of synchronization. The disadvantage is that they have limited error propagation in comparison to synchronous stream ciphers which have none.

2.2.2 Stream Cipher Grain

Grain [23] is a stream cipher which was submitted to eSTREAM in 2004. The purpose of eSTREAM was to find new stream ciphers appropriate for wide-spread use. Grain is intended to be very fast and suitable for limited hardware environments like RFID tags. The inventors of Grain claim that it offers a higher security than several other well known ciphers intended for use in hardware applications such as E0, which is used in Bluetooth and A5/1, which is used in GSM. Both ciphers have been proven to be insecure [36, 20]. The advantage of Grain is that it provides both high security as well as small chip size.

Grain is based on a 80-bit linear feedback shift register and a 80-bit non linear feedback shift register. The LFSR provides a "balanced" output and a minimal period for the key stream. The NFSR and the output function are responsible for bringing nonlinearity to the cipher. The key size is 80 bits and the IV size is 64 bits. Besides the exception of exhaustive key search, no other successful attack against this cipher is known.

Design of Grain

Figure 2.1 gives an overview of the Grain cipher. The cipher consists of an NFSR, an LFSR and an output function. The update functions of the NFSR and LFSR and the output function will be described in detail in this subsection. The bits of the LFSR are denoted by $s_i, s_{i+1}, ..., s_{i+79}$ and the bits of the NFSR are denoted by $b_i, b_{i+1}, ..., b_{i+79}$.

The update function of the LFSR is defined as:
$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i$.

The update function of the NFSR is defined as:
$b_{i+80} = s_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} + b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} + b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} + b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} + b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} + b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}$.

The two shift registers contain the internal state. 5 variables are taken as input for the boolean function $h(x)$ which is defined as:
$h = s_{i+25} + b_{i+63} + s_{i+3}s_{i+64} + s_{i+46}s_{i+64} + s_{i+64}b_{i+63} + s_{i+3}s_{i+25}s_{i+46} + s_{i+3}s_{i+46}s_{i+64} + s_{i+3}s_{i+46}b_{i+63} + s_{i+25}s_{i+46}b_{i+63} + s_{i+46}s_{i+64}b_{i+63}$.
The filter function is first-order correlation-immune and has an algebraic degree of 3. Finally, the output function is defined as $z = h + b_{i+1} + b_{i+2} + b_{i+4} + b_{i+10} + b_{i+31} + b_{i+43} + b_{i+56}$.

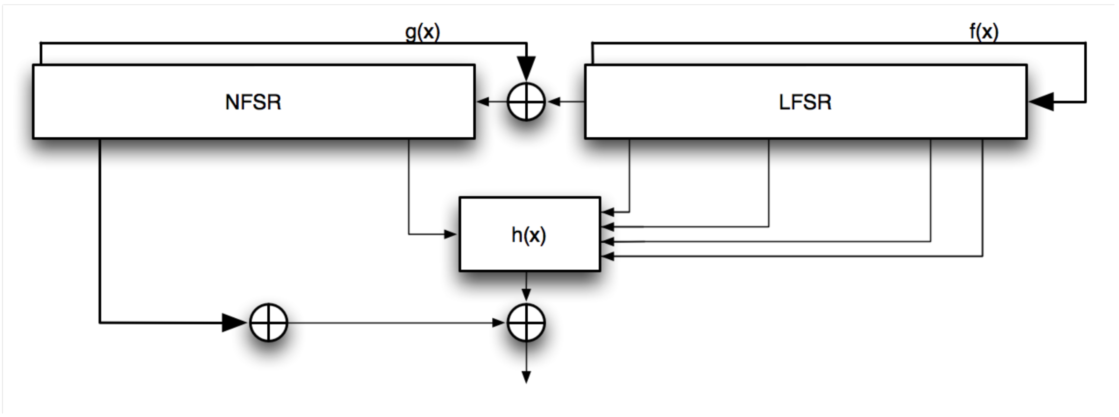


Figure 2.1: Stream cipher Grain.

Initialization

The 80-bit key is loaded into the NFSR and the 64-bit IV is loaded into the first 64 bits of the LFSR ($s_0 - s_{63}$). The remaining 16 bits of the LFSR ($s_{64} - s_{79}$) are set to 1. This is necessary in order that the cipher can not be initialized to the all-zero state. The output function z is fed back to the input of the LFSR and to the input of the NFSR the first 160 cycles. After this initialization process the output z offers the first valid key bit.

Increasing the Throughput Rate vs. Lowering the Power Consumption

After initialization, the throughput of the cipher is 1 bit/cycle. Due to parallelization, it is possible to increase the throughput to 16 bits/cycle. In this case the initialization process would be decreased to 10 cycles. Although the architecture of the Grain cipher is 16 times faster, the size of the datapath is only twice as big. This architecture implements 10 16-bit registers holding the state of the LFSR and NFSR and one 16-bit register for the temporary values. If the optimization goal is reducing the power consumption, there is the possibility to only clock a single 16-bit register at the same point in time via clock gating, which leads to a lower power consumption of the factor 3 compared to the minimum size 1 bit/cycle solution. The drawback is that the throughput is decreased to 1.23 bits/cycle [13].

2.2.3 Block Ciphers

Block ciphers have a defined block size k. On the contrary to stream ciphers which operate only on single bits, a block cipher operates on blocks the size of k-bits. Each k-bit input block is mapped to a k-bit output block. The mapping has to be invertible and the encryption must be a one-to-one function.

Most block ciphers are iterated ciphers which means that the ciphertext is calculated in N rounds. N depends on the used algorithm. Each round can be considered as a key-dependent function. A key schedule is used to expand the secret key to the needed round keys. Many block ciphers consist of a layer of substitution boxes (S-boxes) and a layer of bit permutations. These ciphers are called substitution-permutation networks [57].

The most commonly used block cipher nowadays is the AES algorithm, which will be discussed in Section 2.2.5.

2.2.4 Block Ciphers-Modes of Operation

In order to cover the whole spectra of applications where a block cipher can be used, the NIST [42] releases five "modes of operation" for a block cipher. The modes of operation define the way a block cipher is used:

- Electronic code book (ECB)
- Cipher block chaining (CBC)
- Cipher feedback (CFB)
- Output feedback (OFB)
- Counter (CTR)

For the CBC, CFB and OFB mode in addition to the secret key, there is also an initialization vector needed to process the encryption and decryption.

Initialization Vector

An initialization vector is a block of bits used in some operation modes. The length of the IV depends on the used algorithm, but in most cases it has the same length as the block size of the cipher. It is not necessary that the IV remains secret, but for CBC and CFB mode the IV must be unpredictable. For the OFB mode a unique IV must be used for each encryption. The other party has to know the IV for decryption. A solution to this problem is to send the IV with the encrypted message. The best way to create an IV is using a secure random generator which ensures that the IV is unpredictable.

Electronic Code Book Mode (ECB)

In this mode the cipher encodes each block of the plaintext independently (see Figure 2.2). The decryption is also done on each block independently. A certain block of plaintext will always result in the same block of ciphertext when the same key is used, which is the main drawback of this mode. An advantage of this method is that parallel computing is possible because of the block-to-block independence. This mode is used for secure transmission of single values.

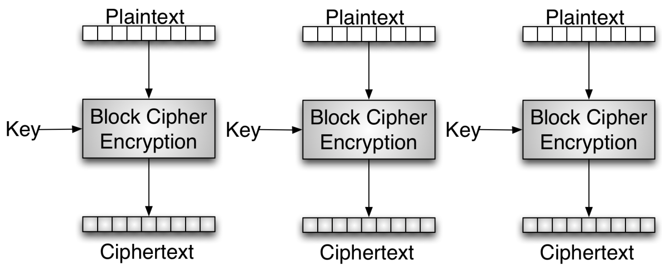


Figure 2.2: ECB encryption.

Cipher Block Chaining Mode (CBC)

This mode is the most commonly used mode of operation. The idea is that the output of the previous cipher is XORed with the following plaintext block (see Figure 2.3). (The first plaintext block is XORed with the IV). The result is used as input for the next cipher. Because of this chain structure parallel processing is impossible. The applications for this mode are general purpose block oriented transmission and authentication.

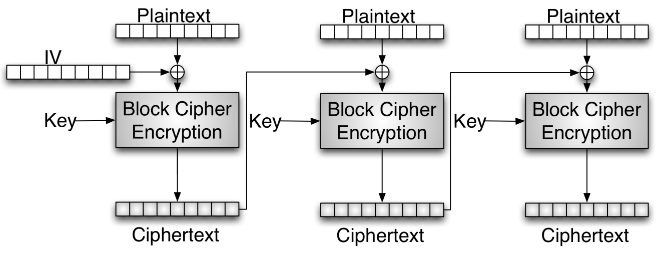


Figure 2.3: CBC encryption.

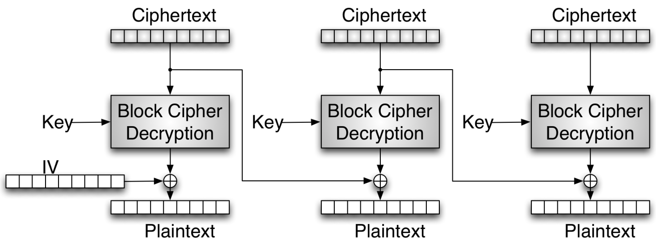


Figure 2.4: CBC decryption.

To perform the decryption the first ciphertext is used as input for the inverse cipher function. Then the result is XORed using the IV to retrieve the first plaintext block. The second ciphertext is input to the inverse cipher and the result is XORed using the first ciphertext to recover the second plaintext. Each output of the inverse cipher has to be XORed with the previous ciphertext to successfully execute the encryption. It is possible to parallelize the decryption process (see Figure 2.4).

Cipher Feedback Mode (CFB)

This mode requires an integer parameter s such that $1 \leq s \leq b$. The variable b is determined by the block size of the cipher, which in the case of AES is 128 bits. First the IV is encrypted. The most significant s bits of the result are XORed with the most significant bits of the plaintext to receive the first s bits of the ciphertext. To get the input of the next cipher, the input of the previous cipher is s positions rotated and then the least significant s bits are substituted using the previous cipher (see Figure 2.5). For the decryption there is no need for an inverse cipher function. The decryption is done in exactly the same way as the encryption except that the XORing happens between the output of the cipher and s bits of the encrypted text, instead of s bits of the plaintext. This mode is used for general purpose stream transmission and authentication.

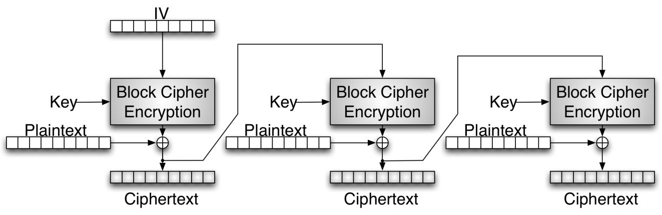


Figure 2.5: CFB encryption.

Output Feedback Mode (OFB)

The IV is encrypted and then XORed with the plaintext to receive the encrypted message block. The output of the cipher is the input of the next cipher. The result is XORed with the next cipher (see Figure 2.6). The decryption is processed exactly the same way as the encryption, except that the XORing happens between the output of the cipher and the encrypted bytes, instead of the plaintext. This mode is used when a stream oriented transmission over a noisy environment is needed.

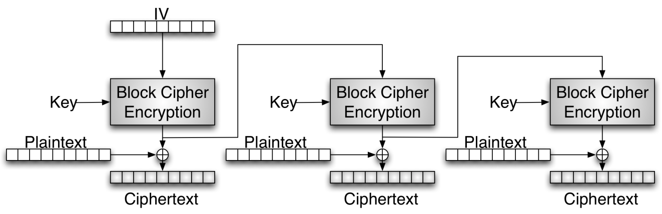


Figure 2.6: OFB encryption.

Counter Mode (CTR)

A counter is used as input for the block cipher and the output is XORed with the plaintext. After encrypting a block the counter is incremented before encrypting the next block (see Figure 2.7). For decryption the initialization vector for the counter must be known. The counter is the input for the cipher and the result is XORed with the encrypted text. This mode is used for high-speed requirements.

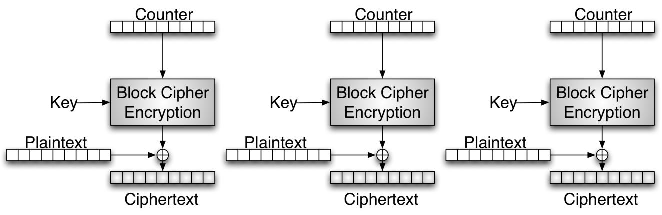


Figure 2.7: CTR encryption.

2.2.5 Advanced Encryption Standard

In 1997 the US National Institute of Standards and Technology (NIST) initiated a contest to find the successor of the DES and 3-DES. Submission was open to everyone. The cipher had to fulfill the following requirements:

- Symmetric (encryption and decryption use the same secret key)
- Block size of 128 bits
- Key length of 128, 192 and 256 bits

The goal of the NIST was a block cipher as secure as 3-DES but more efficient. The key length of the DES algorithm (56 bits) is too small because a brute force attack is possible. The submitted proposals were evaluated using various assessment criteria. The most important category was the security of the cipher. The computational efficiency, program size and memory requirements in software implementation and chip area in hardware implementation also had an impact on the evaluation process. The AES should provide flexibility. This means it should fit on a smartcard and it should be possible to reach high throughputs on non-limited hardware. The last criterion examines the simplicity of the cipher in regard to the size of the description, which is necessary in order to understand the algorithm [12].

In 2000, Rijndael [11] officially became the AES. The reasons were the good performance in software and hardware implementation, the wide spectrum of possible target platforms, low memory requirements and a good degree of parallelism. Furthermore, Rijndael reaches an excellent key setup time and offers good potential to implement countermeasures against side-channel attacks. Rijndael is an iterated block cipher with variable block length and variable key length.

The AES standard [15] describes the unmodified Rijndael-Algorithm using a fixed block size of 128 bits and a key length of 128, 192 or 256 bits.

Fundamentals

A finite field has a limited number of elements which is always a prime power p^n, denoted as $GF(p^n)$. In the field the only allowed integers range from 0 to $p^n - 1$. The smallest processing unit in the AES is one byte, which leads to the $GF(2^8)$. The byte $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$ could also be interpreted as a polynomial $\sum_{i=0}^{7} b_i x^i$. The mathematical functions are redefined in the finite field.

Addition

The addition is processed by adding the coefficient of the corresponding powers of the polynomials modulo 2. That can be reached by the exclusive or operation which is denoted by $\oplus$.

Multiplication

The multiplication is processed by the multiplication of the corresponding polynomials modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, which is denoted by $\bullet$.

Polynomials with coefficients in $GF(2^8)$

The 4-byte words $a = \{a_3, a_2, a_1, a_0\}$ and $b = \{b_3, b_2, b_1, b_0\}$ are added using their polynomial representations $a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0$ and $b(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0$. Thus $a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$. The modular product $a(x) \otimes b(x)$ is given by $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$. $d(x)$ can be calculated using the following formulas:

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \tag{2.1}$$

$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \tag{2.2}$$

$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \tag{2.3}$$

$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \tag{2.4}$$

This can be expressed in matrix notation:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.5}$$

State

The AES algorithm operates on a two dimensional array of bytes. This 4x4-array is called the State. Each byte has two indices which lie both in the range 0-3. The first index is the row number, the second is the column number. At the beginning of the encryption, 16 input bytes are copied to the State and then the algorithm is performed on these bytes. Finally, the State consists of the encrypted bytes. At the beginning of the decryption, the encrypted bytes are copied into the State. Then the inverse cipher is executed on the State, resulting in the plaintext lying in the State. The State is organized as an array of columns, which means that every column could be considered as a 32-bit word.

If the input bytes are the array $\left[\begin{array}{cccccccccccccccc} a_0 & a_1 & a_2 & a_3 & b_0 & b_1 & b_2 & b_3 & c_0 & c_1 & c_2 & c_3 & d_0 & d_1 & d_2 & d_3 \end{array}\right]$, the corresponding State will be:

$$\begin{bmatrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{bmatrix} \tag{2.6}$$

.

Algorithm Specification

The cipher consists of the following components which will be described later:

- SubBytes()
- ShiftRow()
- MixColumn()
- AddRoundKey()

These four commands form one round of the cipher. The number of the rounds depends on the key length (see Table 2.1).

key length	**rounds**
128 bit	10
192 bit	12
256 bit	14

Table 2.1: AES rounds depending on key length.

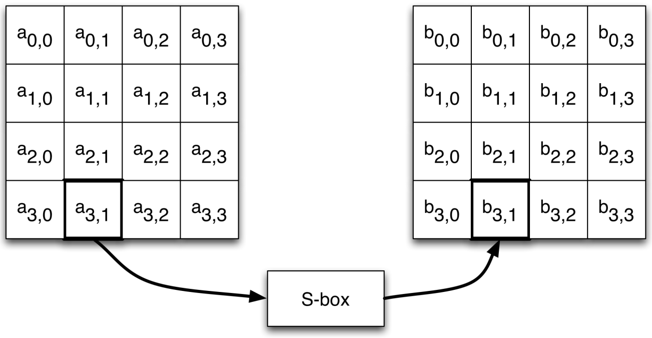
The following listing is the pseudo code of the AES algorithm:

```
Cipher(state, expanded_key)
begin
        round=0
        AddRoundKey(state,expanded_key,round)

        for round=1 to 9
                SubBytes(state)
                ShiftRows(state)
                MixColumns(state)
                AddRoundKey(state,expanded_key,round)
        end for

        round=10
        SubBytes(state)
        ShiftRows(state)
        AddRoundKey(state,expanded_key,round)

        out =state
end
```

Listing 2.1: AES cipher pseudo code.

SubBytes()

Each byte is replaced using a substitution table (S-box).



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
S-box
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

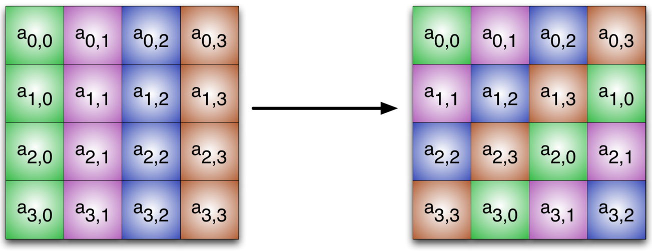
Figure 2.8: AES S-box operation.

The S-box is created by performing the multiplicative inverse in the finite field $GF(2^8)$, followed by an affine transformation which is expressed in matrix notation as:

$$byte_b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.7}$$

ShiftRows()

This function consists only of cyclical shifts performed on the second, third and fourth row. The first row does not change. The second row is cyclically left shifted one byte which increases to two bytes for the third row and three bytes for the fourth row.



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 2.9: AES ShiftRows() operation.

MixColumns()

In this operation each column is interpreted as polynomial $s(x)$ (see Section 2.2.5). $s(x)$ is multiplied with $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ and is written as

$$\begin{bmatrix} b_{0,x} \\ b_{1,x} \\ b_{2,x} \\ b_{3,x} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,x} \\ a_{1,x} \\ a_{2,x} \\ a_{3,x} \end{bmatrix} \tag{2.8}$$

where the index x stands for the row number. This multiplication results in:

$$b_{0,x} = (02 \bullet a_{0,x}) \oplus (03 \bullet a_{1,x}) \oplus a_{2,x} \oplus a_{1,x} \tag{2.9}$$

$$b_{1,x} = a_{0,x} \oplus (02 \bullet a_{1,x}) \oplus (03 \bullet a_{2,x}) \oplus a_{3,x} \tag{2.10}$$

$$b_{2,x} = a_{0,x} \oplus a_{1,x} \oplus (02 \bullet a_{2,x}) \oplus (03 \bullet a_{3,x}) \tag{2.11}$$

$$b_{3,x} = (03 \bullet a_{0,x}) \oplus a_{1,x} \oplus a_{2,x} \oplus (02 \bullet a_{3,x}) \tag{2.12}$$

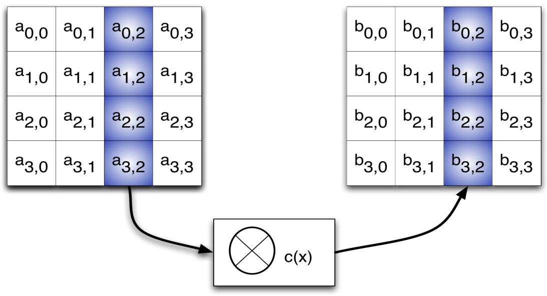


Figure 2.10: AES MixColumns() operation.

AddRoundKey()

This function consists of a simple XOR operation with the round key. The round key is different every round and is constructed from the secret key using a key expansion routine.

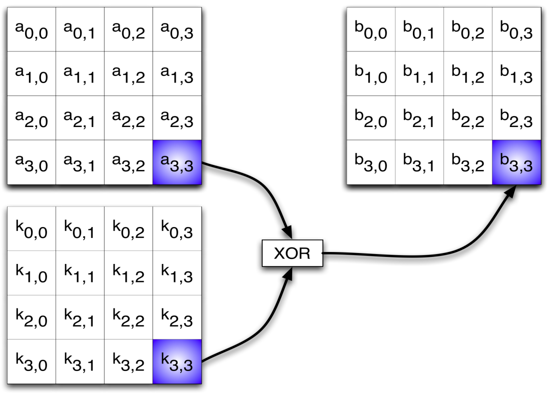


Figure 2.11: AES AddRoundKey() operation.

KeyExpansion()

This routine takes the secret key and expands it. The size of the expanded key depends on the number of rounds (see Table 2.1). It is the multiplication of the result of the number of rounds plus one and the block-size of 16 bytes.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
     word temp
     i = 0
     while (i < Nk)
        w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
        i = i+1
     end while
     i = Nk
     while (i < Nb * (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
             temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
             temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
     end while
end
```

Listing 2.2: AES key expansion pseudo code.

SubWord() This function uses the S-box for substituting 4 bytes.

RotWord() This function takes as input a word $[a_0, a_1, a_2, a_3]$ and performs a 8 bit cyclic left shift resulting in the output word $[a_1, a_2, a_3, a_0]$.

Rcon[i] It consists of the values given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ where x is $\{02\}$.

Decryption

For the decryption it is necessary to inverse the routines of the cipher. AddRoundKey() remains the same operation because the inverse of a XOR is a XOR again.

InvShiftRows() The rows are shifted right instead of left.

InvSubBytes() The substitution table for the byte wise replacement is the inverse of the S-box. InvS-box(S-box(a))=a.

InvMixColumns() Each column is multiplied by
$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

Inverse Cipher vs. Equivalent Inverse Cipher For the decryption algorithm there are two possibilities. The inverse cipher is the straightforward attempt of doing the decryption (see Listing 2.3). For the implementation in Section 4 the equivalent inverse cipher is necessary (see Listing 2.4). Using the inverse routines of the cipher, the sequences of the operations are:

```
InvCipher(state,expanded_key)
begin
        round=10 //for AES-128
        AddRoundKey(state,expanded_key,round)
        for round=9 down to 1
                InvShiftRows(state)
                InvSubBytes(state)
                AddRoundKey(state,expanded_key,round)
                InvMixColumns(state)
        end for
        round=0
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state,expanded_key,round)
        out =state
end
```

Listing 2.3: AES-128 inverse cipher pseudo code.

Due to the following two properties, it is possible to change the sequence of operations so that the inverse cipher has the same sequence as the cipher.

1. The order of InvSubBytes() and InvShiftRows() operations can be changed because InvSubBytes() is a byte-wise operation. Therefore, the result remains the same and is not dependent on the sequence of these two operations.

2. InvMixColumns() is linear, meaning that $InvMixColumns(\textit{state}\, XOR\, \textit{roundkey}) = InvMixColumns(\textit{state})\, XOR\, InvMixColumns(\textit{roundkey})$.

These two properties allow the following sequence:

```
EqInvCipher(state,expanded_key)
begin
        round=10 //for AES-128
        AddRoundKey(state,expanded_key,round)
        for round=9 down to 1
                InvSubBytes(state)
                InvShiftRows(state)
                InvMixColumns(state)//InvMixColumns() must be
                        //also performed on the round_key
                AddRoundKey(state,expanded_key,round)
        end for
        round=0
        InvSubBytes(state)
        InvShiftRows(state)
        AddRoundKey(state,expanded_key,round)
        out =state
end
```

Listing 2.4: AES-128 equivalent inverse cipher pseudo code.

2.3 Public-Key Algorithms

Public-key algorithms are also called asymmetric algorithms. They are designed in a way that the key used for encryption is different from the key used for decryption. The decryption key can not be calculated from the encryption key in any reasonable amount of time. Since the encryption key must not be kept secret it is called a public key. A person's public key can be used to encrypt a message only this person can decrypt using his private key. The authenticity of the public key must be guaranteed. When an attacker forges the public key of a person, the attacker can decrypt every message destined for the person.

Public-key algorithms are slower than symmetric-key algorithms. Therefore, public key cryptography is most commonly used for exchanging the symmetric key and doing the encryption using symmetric key algorithms and also for encrypting small data items like PINs and credit card numbers.

The most common public-key algorithm is RSA, but the elliptic curve cryptography, which is often used as an alternative, is superior to RSA and any other public key algorithm.

2.3.1 Elliptic Curve Cryptography

Elliptic Curve Cryptography was first proposed by Koblitz [32] and Miller [40] in 1985. ECC is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP). The best algorithms known to solve the problem have exponential running time. As a consequence, the same desired security level can be reached using smaller key sizes in comparison to RSA. An 160-bit ECC key provides the same security as a 1024-bit RSA key. The smaller key size results in less power consumption and less storage [22].

Elliptic Curve Discrete Logarithm Problem

An elliptic curve E over $\mathbb{F}_p$ (the field of integers modulo p) is defined by the equation:

$$y^2 = x^3 + ax + b, \tag{2.13}$$

where p is a prime number and a,$b \in \mathbb{F}_p$. There is the requirement for a and b which must satisfy the equation:

$$4a^3 + 27b^2 \neq 0(mod\, p). \tag{2.14}$$

Let P be a point of the elliptic curve E, and P has prime order n (prime order n means that the factorization of P consists of n prime numbers). Then the cyclic subgroup of E generated by P can be defined as:

$$\langle P \rangle = \{\infty, P, 2P, 3P, ..., (n-1)P\}. \tag{2.15}$$

P, n and E are public. The private key d is an integer chosen from the interval $[1, n-1]$. The public key is $Q = dP$. The problem of determining d, when the parameters of the elliptic curve and Q are given, is called the Elliptic Curve Discrete Logarithm Problem [22].

Encryption and Decryption

The encryption and decryption works on the base of the ElGamal [17] encryption scheme. A plaintext must be represented as a point M. It is encrypted by adding kQ to it, where k is a randomly chosen integer and Q is the public key of the receiver of the message. The sender has to transmit $C_1 = kP$ and $C_2 = M + kQ$ to the receiver. The receiver uses his private key d to compute $dC_1 = d(kP) = k(dP) = kQ$. Additionally, the message can be discovered using the equation $M = C_2 - kQ$.

2.3.2 Public-Key Infrastructure

Asymmetric cryptography holds the advantage of easier key management over symmetric cryptography. The key used for the encryption does not need to be secret. Only the secret key which is necessary for the encryption must be kept secret. The public keys need to be protected against abuse and forgery. The distribution and storage of the public and private keys is the function of a Public-Key Infrastructure [6].

Personal Security Environment

In public-key systems the private key is necessary for decrypting and signing messages. This key needs to be secret, otherwise anyone who knows the key could forge signatures and decrypt messages of the key owner. The environment in which the private key is stored is called Personal Security Environment (PSE). The private key must not leave this environment. That is the reason why decryption and signing are always executed within the PSE. The private key is often created in the PSE. This practice is advantageous because no one else can know the private key. However, this practice also has a drawback because errors may be made during the creation process, such as using a weak random generator.

The alternative is to create the key by another trusted party. The PSE can be a part of the hard disk which is secured with a password. Then the security of the PSE correlates with the security of the operating system. Therefore, smartcards are an appropriate solution for the use as PSE. It is very hard for an attacker to manipulate the smartcard hardware and software. The drawback is that smartcard calculations are very slow compared to a PC. A solution is to use session keys for encryption of the data and then encrypt these session keys using the public key. Here the receiver only has to decrypt the session key using the smartcard and the encrypted message can be decrypted using a PC.

Sometimes a problem in the signature process occurs. The attacker can change and afterwards sign the data that is sent from the PC to the smartcard without the user noticing the attack. The problem is that the user is not able to see the data which is sent to the smartcard. Another approach is to use a mobile phone as a PSE and to program the mobile phone to display every message that should be signed.

Certification Authority

Apart from protecting the private key, it is also important to guarantee the authenticity of the public key. An attacker who succeeds in forging the private key of a user can decrypt every message which is addressed to the user and fake signatures of the user.

One way to assure that other users get the right public key of a certain user is to allocate each user to a trusted certification authority (CA). The CA guarantees with its own signature for the correctness of the public key of the allocated users. If two users belong to two different CAs there is a certification chain. If user 1 wants to validate the public key of user 2, user 1 must first get the public key of the other users' CA signed from his CA. After verifying the public key of the other's CA, the public key of the other user can be verified. This method only works if the assumption can be made that trust is transitive, which means when a user trusts his CA, the user will also trust the people his CA trusts.

2.4 Hash Functions

A hash function is a one way function which is important in message authentication and used for creating digital signatures. A hash function H produces a "fingerprint" of a file, message or other data block and should provide the following properties [53]:

1. The input of H can be a data block of any size.
2. The output produced by H has to be of a fixed length.
3. The computation of $H(x)$ is relatively easy for any given x. Hardware and software implementations should be practical.
4. For any given value h it should be practically impossible to find x such that $H(x) = h$. It is a one way function which means that it is very hard to inverse this function. This property is important when the authentication algorithm uses a secret value. From the hashed secret value it is not feasible to calculate the secret value.
5. For any given x it should be practically impossible to find a $y \neq x$ so that $H(x) = H(y)$. This feature is called weak collision resistance which means that it is very hard for an attacker to find another message that results in the same hash value. This property prevents an attacker from forging a message.
6. It is practically impossible to find any pair (x, y) so that $H(x) = H(y)$. This property is called strong collision resistance.
7. If the input changes only in one bit, every bit of the output is affected.

A commonly used set of hash functions is the Secure Hash Algorithm.

2.4.1 Secure Hash Algorithm

The Secure Hash Algorithm is a set of hash functions invented by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) [43].

The different SHA functions differ in output size and security. While SHA-0 is considered to be insecure, because a collision can be easily found, SHA-1 is more secure. To find a collision in SHA-0 the best attack needs 2^{36} operations [41].

The best attack on SHA-1 needs 2^{62} operations [10]. Therefore, the use of SHA-2 is preferable because up to now there are no known collision attacks better than brute force attacks. Regardless, SHA-1 is still used for many applications including TLS, SSL, IPSec, SSH, PGP, S/MIME.

2.5 Digital Signatures

A digital signature is used to authenticate the origin of a document and guarantee that a document has not been altered by an attacker. A hash algorithm is combined with public-key cryptography. The sender of a message hashes the message and encrypts the hash value using his private key. Then the message and the encrypted hash value are sent to the receiver. The receiver can then decrypt the encrypted hash value using the public key of the sender. In order for the receiver to verify that the message has indeed been sent by the sender and has not been altered, the decrypted hash value and the self calculated hash value must be the same.

A widely spread algorithm is the Elliptic Curve Digital Signature Algorithm.

2.5.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm (DSA) where Elliptic Curve Cryptography (ECC) is used as an asymmetric encryption algorithm. ECDSA is a wide spread algorithm for creating digital signatures and is an ANSI and IEEE as well as an NIST standard [28].

2.5.2 ECDSA Signature Generation and Verication

P and n are elliptic curve parameters (see Section 2.3.1), where d is the private key, Q is the public key and m is the message. The following pseudo code describes the signature generation using the private key d:

1. A random integer k is selected that $1 \leq k \leq n-1$.
2. $kP = (x_1, y_1)$ is computed.
3. $r = x_1 mod\ n$ is computed and if $r = 0$, then go back to step 1.
4. Compute $e =$SHA-1(m).
5. $s = k^{-1}(e + dr)\ mod\ n$ is computed and if $s = 0$, then go to step 1.
6. The signature for the message m is (r, s).

After the generation, the signature can be verified using the public key Q:

1. r and s must be integers of the interval $[1, n-1]$.
2. Compute $e =$SHA-1(m).
3. $w = s^{-1} mod\ n$ is computed.
4. $u_1 = ew\ mod\ n$ and $u_2 = rw\ mod\ n$ is computed.
5. $X = P^{u_1} Q^{u_2}\ mod\ n$ is computed.
6. If $X = 0$ the signature is rejected. Otherwise $v = X\ mod\ n$ is computed.
7. The signature is accepted if $v = r$.

2.6 Random Number Generator

A random-number generator is a device which can create a sequence of numbers that is statistically random. RNGs are needed in cryptography for the generation of the cryptographic keys and initial vectors. When obeying Kerckhoffs' Principle it is even more important to have RNGs which really produce numbers nobody can guess, because the security of the cryptographic system only depends on the secrecy of the key.

The produced numbers have to be normally distributed on the whole key space. As a consequence, after generating a key using a RNG each value that can be expressed with a k-bit length number is equally likely for a key length of k. When the key space is decreased or some values are more likely than others the security of the cryptographic algorithm is decreased.

It can be distinguished between true random-number generators and pseudorandom-number generators.

2.6.1 True Random Number Generator

True random number (TRNG) generators have the property that each bit is truly random. They are also called non-deterministic random generators because they are based on unpredictable physical processes:

- Radioactive decay
- Thermal noise
- Photoelectric effect

The processes share the common property that in theory they are all completely unpredictable.

2.6.2 Pseudorandom Number Generator

Pseudorandom number generators are also known as deterministic RNGs. A function is defined to be deterministic when the output can be calculated from a given input and the same input always results in the same output. Random number generators are often used for security critical applications which is the reason of building secure random number generators. The PRNG must be initialized with an entropy input. Entropy is defined as a measure of disorder, randomness or variability in a closed system.

The recommended methods for generating random bits using deterministic methods are either hash functions, block ciphers, or number theoretic problems. The value used for initializing the PRNG is called seed. A seed can consist of entropy input, a nonce (a time varying value) and a personalization string. The personalization string should be a bitstream which is as unique as possible. The entropy input is provided by either an TRNG or an appropriate entropy source. There are two terms in regard to the security of the random generator: backtracking resistance and prediction resistance. Backtracking resistance means that it is not possible to determine the previous states of the PRNG after knowing a later state. Backtracking resistance can be reached by using a one-way function as a generator function for the PRNG. Prediction resistance means that after knowing the state, neither future states nor outputs of future states can be predicted. Prediction resistance can be reached only by reseeding the PRNG. The problem if a

PRNG is implemented on a smartcard is that due to the limited resources it is very hard to gain entropy input. That is the reason why smartcards often use a voltage-controlled oscillator as TRNG for the generation of a seed for the PRNG [4].

The goal is that each of the 2^n values of an n-bit number are equally likely. In practice this is achieved by an entropy input as seed of a PRNG. The problem is to find an adequate source which offers the required entropy. Especially in embedded systems, there is the problem of finding access to "cheap" entropy. A requirement for the algorithm is that if the algorithm and the output are known but the internal state is not, an attacker should not be able to find another state which produces the same output. The more entropy there is in the input, the more collision resistant the PRNG is. A well-seeded PRNG is not a profitable target for an attacker. Apart from security the processing speed of the PRNG is also very important. It should be possible to produce a high amount of numbers in a very short time. The best choice for the cryptographic algorithm used for the PRNG is a well analyzed block cipher in CTR mode. It will leak a mere one bit of information after producing $2^{(n/2)}$ blocks of output, where n is the block size of the cipher in bits [59].

If stream ciphers are used they need to resist related key attacks. The use of the stream cipher RC4 as a PRNG causes security problems [16].

An appropriate candidate for use as a PRNG is the stream cipher Grain which has been described in Section 2.2.2.

Chapter 3

Smartcards

A smartcard is a plastic card which meets the ISO 7810 standard. The card has a bank-card's size and thickness. On the card there are implemented one or more ICs within the required thickness. The advantage of smartcards is that they are able to execute cryptographic algorithms locally in their internal circuitry. Smartcards can help whenever secure portable objects are needed and whenever the "external world" needs to work with data without knowing its actual value. The card's tamper resistance combined with public-key cryptography provides an adequate solution to many "everyday security problems" [18].

Since many applications today are used in the finance and payment sector and also deal with sensitive issues such as identification and health, security is an issue of great importance. It is advantageous for the user that with one smartcard it is possible to handle multiple applications. In regard to security there are several requirements a smart-card system must fulfill [24].

Confidentiality: Data is only accessible to authorized persons.

Authentication: The identity of persons must be verifiable by other parties.

Integrity: It ensures that the data could not be altered by another person.

Nonrepudiation: It ensures that after completing a transaction one is not able to deny one's actions.

The importance of smartcards is increasing rapidly. They are used for IDs, passports, credit cards and e-tickets. The overall shipments within the smartcard industry totaled 2.02 billion cards in 2003 and were projected to reach 3.11 billion cards by 2008 [25]. The global shipment of smartcards has exceeded an estimated amount of about 5 billion units in 2008 [48].

In addition to contact smartcards, there are contactless smartcards which have no physical connection with the reader and are powered by the electromagnetic field of the reader. Here there is a problem because some applications need a contact smartcard while others need a contactless smartcard. An approach to solve this problem is a dual-interface design. The disadvantages of contactless smartcards are their limited power due to the electromagnetic field and their slower data transmission rate in comparison to contact smartcards. Dual interface smartcards combine the strengths of contactless and contact cards.

3.1 Outline

To begin with, contactless smartcards are described. This description includes the ISO 14443 standard and attacks on contactless smartcards. As a consequence of the security-related applications of smartcards, the next section discusses smartcard security including a description of side-channel attacks and countermeasures. The last section explains the main components of smartcards with a special focus on memory including latches, flip-flops, RAM and a power-saving technique called "clock gating".

3.2 Contactless Smartcards

Contactless smartcards are a very flexible technology. They contain solutions tailored to nearly any application. They have a reduced maintenance cost and an increased lifetime because they have no mechanical wear. A further advantage is their convenient usability. They do not have to be inserted into a reader or oriented in a specific way. It is possible that in most cases they can be kept in a wallet. Therefore, fast transactions are possible. Because contactless smartcards have no power supply, they are powered by the electromagnetic field of the reader. This limited energy source is the reason why cryptographic algorithms are more difficult to implement on contactless smart cards than on contact smartcards. Data is exchanged through amplitude modulation. Through time multiplexing it is possible that a reader communicates with several cards at a time.

Because of the nonexistence of the physical link, it is necessary that the communication between the contactless card and the reader remains secure. The smartcard and the reader have to authenticate that they have the permission to talk to each other. This can happen by exchanging a shared secret [2]. Some attacks have recently been proposed on contactless smart cards (see Section 3.2.2).

There are several standards describing contactless smartcards but the ISO 14443 standard [27] is the most important.

3.2.1 ISO 14443

In the ISO/IEC 14443 a contactless protocol communication is defined. It is the most popular standard for high frequency proximity cards, and it is commonly used for access control systems, credit cards and passports. The standard does not specify any specific cryptographic algorithm but security features of the smartcard standard ISO 7816 are also compatible to ISO 14443 devices. In ISO/IEC 14443 the required working distance of the smartcard is between 0 and 10 cm. The ISO 14443 standard consists of 4 parts:

1. Physical characteristics
2. Radio frequency power and signal interface
3. Initialization and anticollision
4. Transmission protocol

These parts are described in more detail in the following sections.

Physical Characteristics

The dimensions should be compatible with the ISO 7810 standard (i.e. 85.72 mm x 54.03 mm x 0.76 mm). In this section tolerance levels for the following topics are also specified:

- Ultra-violet light
- X-rays
- Dynamic bending stress
- Dynamic torsional stress
- Alternating magnetic fields
- Alternating electric field
- Static electricity
- Static magnetic field
- Operating temperature

Radio Frequency Power and Signal Interface

This section describes the interface between the reader and the smartcard. The operating field of the reader has to be 13.56 MHz and is used for powering the smartcard. The modulated field is used for bidirectional communication. Two different signal interfaces exist and differ in the modulation used for communication (i.e. Type A, Type B). When in the Idle state the reader has to switch periodically between the two types. The data rate is 106 kbps. Some smartcards support higher data rates of 212/424/848 kbps but these higher data rates can only be chosen after the anticollision is processed.

Type A The channel from the reader to the smartcard uses 100% Amplitude Shift Keying (ASK) with Modified Miller encoding. The channel from the smartcard to the reader uses load modulation ASK with Manchester encoding.

Type B The channel from the reader to the smartcard uses 10% ASK with Non-Return-to-Zero (NRZ) coding while the channel from the smartcard to the reader uses load modulation Binary Phase-Shift Keying (BPSK) with NRZ encoding.

Initialization and Anticollision

The reader has to detect if there are smartcards in the field by sending repeated request commands. A smartcard in the unmodulated field should be able to respond within 5 ms. There are three different types of frame formats defined which are used in the ISO 14443 standard:

- Short frame
- Standard frame
- Bit-oriented anticollision frame

When more than one smartcard is in the same field, there are collisions in the communication. That is the reason why this part of the standard provides an anticollision protocol. In order to avoid this problem, a unique ID is needed for each card. The anticollision sequence is different depending on the type of the card (A or B). For practical reasons only the anticollision sequence for type A cards is discussed here. For implementing the anticollision sequence the smartcards need to have a state variable which can either be:

POWER-OFF: The smartcard is not in the field.

IDLE: The smartcard is in the field and should recognize REQA and WUPA commands.

READY: In this state the anticollision method is applied.

ACTIVE: The smartcard listens to higher layer messages.

HALT: The smartcard should only respond to WUPA commands.

The necessary commands for the reader for the anticollision sequence are:

REQA and WUPA: These commands are used by the reader to probe the cards in the field. A smartcard in the IDLE state answers to both commands with ATQA while a smartcard in the HALT state only answers to WUPA.

ANTICOLLISION and SELECT: These commands are used during the anticollision loop. As long as no complete ID of a smartcard is transmitted the ANTICOLLISION command is sent. The method for retrieving an ID is a binary search. After the ID of one card is retrieved the SELECT command changes the state of the card to ACTIVE.

HLTA: All smartcards which are in the ACTIVE state change their state to HALT.

Transmission Protocol

The transmission protocol specifies a half-duplex block transmission protocol. "Type A" smartcards setup parameters for the configuration of the protocol are determined (frame size, card identifier, etc.). When a smartcard is selected, the SELECT acknowledgment contains the information if the smartcard supports the ISO 14443-4 protocol or uses a proprietary protocol. If ISO 14443-4 is supported the reader transmits a Request for Answer To Select (RATS) command. The smartcard responses with an Answer to Select (ATS) command. After that a Protocol and Parameter Selection (PPS) is sent, which allows the change of the data rate if supported. Now Application Data Units (APDUs) containing any data can be exchanged [38].

3.2.2 Practical Attacks on Contactless Smartcards

Kfir et al. [30] show that contactless smartcard technology is vulnerable to relay attacks in which an attacker tricks the reader into communicating with a victim's smartcard that is far away. The attacker can remotely use the victim's card without the victim's knowledge.

Hancke [21] executes a successful relay attack against an ISO 14443A contactless smartcard up to a distance of 50 meters with low-cost hardware. A countermeasure against this attack is to use distance bounding protocols, but the implementation on a low-cost passive smartcard with limited resources remains a problem.

3.3 Smartcard Security

As a consequence of the wide spread use of smartcards in security-related applications, smartcard attacks and security is a heavily studied area. There are three different approaches for attacking smartcards [38]:

Invasive Attacks: The microprocessor of the smartcard is removed and directly attacked through physical means. This attack can, in theory, break the security of any given secure microprocessor. The drawback is that these attacks require very expensive equipment and are very time-consuming.

Semi-Invasive Attacks: The surface of the chip is exposed but an attacker does not modify the chip.

Non-Invasive Attacks: The attacker does not modify neither the microprocessor nor the plastic card. The attacker tries to gather the information using side-channels. More details of this attack are shown in Section 3.3.1.

The equipment used for the last two attacks is quite inexpensive. Hence much research on these forms of attacks has been conducted.

3.3.1 Side-Channel Attacks

Cryptographic algorithms are used in most commercial environments and while many of them are standardized and well analyzed, the practical implementations of these algorithms are open points for an attack. The poor design and implementation of the system cause many security vulnerabilities. It is expected that these systems are insecure, but there is also a problem concerning well-designed and well-implemented systems. The physical realization of the abstract model leaks information about its processed data. This leakage is called side channel and can break a system's security. Such side channels can be the power consumption of a cryptographic device, externally observable timing of certain operations, electromagnetic radiation of the device, the sound produced during computation and so on [5].

Each side channel can be used to retrieve secret information of the cryptographic device like a secret key. All of these attacks have an underlying physical principle in common which causes various physical effects, depending on the cryptographic device. Side-channel attacks are very effective against smartcards because an attacker can gain full control of the card.

Timing Attacks

The principle behind timing attacks is that the execution time of the cryptographic algorithm is data dependent. An attacker who can measure the time a device needs to execute cryptographic operations can also determine the secret key. Because these types of attacks require a very large number of measurements, they are generally not as serious as power-analysis attacks to devices like smartcards [22].

Power Analysis Attacks

Power analysis attacks describe crypto analytic attacks in which the attacker's goal is to retrieve the secret key of a cryptographic device by analyzing its power consumption. These attacks are not against the cryptographic algorithm itself but against its implementations. There is no need for expensive equipment. These attacks work because the power consumption of the device depends on the data it processes. Power analysis attacks can be classified into simple power analysis and differential power analysis [37].

Simple Power Analysis (SPA) It is a very strong attack but requires some easily detectable leakage in the power signal. The attacker tries to retrieve the secret key from one given power trace or a few traces. In these cases a simple visual inspection is used to retrieve the secret key. Often a detailed knowledge about the implementation of the algorithm is necessary. Implementations of algorithms which have key-dependent branches are extremely vulnerable to this kind of attacks. Many hardware implementations of symmetric cryptographic algorithms have such a small power consumption variation that a successful simple power analysis attack is impossible [33].

In an implementation which performs key dependent operations, the shape of the power signal can reveal the secret key. SPA can also be possible when key-dependent data is manipulated by instructions that leak information. For example, the carry bit of some smartcards leaks information in the power channel [5].

Differential Power Analysis (DPA) If the algorithms are implemented using low-leakage instructions and a fixed instruction execution sequence, SPA attacks are no longer possible. However, there is still a very small variation in the power consumption caused by key and data dependences of the cryptographic device. DPA was first proposed by Kocher [33] and is based on the fact that the power consumption of a cryptographic device depends on the manipulated data. After analyzing a large number of power traces and the use of statistical methods, it is possible to find the correct secret key. A correlation function which correlates a guessed key with the unknown secret key is used. To be exact, only a part of the key is guessed and correlated against the secret key while the remainder key is constant. The correlation function has a peak where the guessed sub key is the secret sub key.

A five step strategy for processing a successful attack is as follows:

1. The first step is to choose an intermediate value that is the result of a function using a known non-constant value and a part of the key as input. This leads to better results if the function is a non-linear function because a difference of one bit in the input influences several bits of the output.

2. The next step is to record the power traces and store them together with the corresponding plaintext.

3. The intermediate values are calculated using all possible values of the partial key and all the known plaintexts. A matrix is stored where each column represents the encryption runs with the different plaintext but the same key. The number of columns equals the number of different partial keys.

4. The hypothetical power consumption values are calculated using a power analysis model of the given cryptographic device and mapped to the matrix of step three.
5. Each column of the hypothetical power consumption matrix is compared to each column of the power traces. This results in a matrix where each element is the correlation coefficient of two certain columns. The columns containing the highest value can be mapped back to the correct partial key.

These steps have to be repeated until the whole key is received.

3.3.2 Countermeasures against Power Analysis Attacks

The two approaches of countermeasures that exist are hiding and masking. Hiding can be seen as the engineering approach decreasing the signal-to-noise ratio of correlation. On the contrary, masking is the mathematical approach to decorrelate both the power consumption and the key.

Hiding

The goal is either a random or a constant power consumption in every clock cycle. This can be reached using various methods. The power consumption can be changed in the time dimension as well as in the amplitude dimension. In practice, it is impossible to completely avoid the correlation between processed data and power consumption.

Time Dimension The power consumption is randomized by the following methods:

Dummy Operations Dummy operations can be inserted at random times in the algorithm. It is important that the number of dummy cycles is constant in each round of the algorithm. The drawback is that this method decreases the throughput of the cryptographic device. There is also a possibility of changing the clock signal. Clock pulses can be skipped. Another way to randomly change the clock signal is to have multiple clock domains. Chari et al. [8] claim that the insertion of dummy operations can be undone.

Shuffling If there are independent parts of the algorithm these parts can be shuffled. This means that the different parts are processed in a random order. The disadvantage of this method is that there is only a limited number of independent parts. For the AES algorithm the 16 S-box look-ups can be shuffled. At first sight 16^2 times more power traces are necessary to retrieve the secret key but Clavier et al. [9] shows that the number of needed power traces only grows linearly with the number of shuffled operations.

Amplitude Dimension The power consumption of the performed operations is changed directly.

Adding Noise Adding noise makes it harder to find the correlation between the key and the key-dependent power consumption. Noise is added by parallel processing and random-switching processes.

Constant Power Consumption Logic styles can be used which have constant power consumption. This can be implemented by doubling the number of cells and wires using complimentary logic so that every transition has an inverted transition. The disadvantage is that this method also doubles the area and the power consumption. Another attempt to achieve constant power consumption requires the use of filters.

Masking

Decorrelation is achieved by masking the intermediate values. The masked intermediate value is independent of the plaintext and the intermediate value. There are two types of masking. Boolean Masking is a simple XOR operation whereas Arithmetic Masking is an addition or a multiplication modulo of a certain number. Because the non-linearity of the S-box would require much more effort for Boolean Masking, Arithmetic Masking is preferred.

3.3.3 Electromagnetic (EM) Attacks

These kinds of attacks use the electromagnetic field of a cryptographic device as a side channel. They can be launched from distances where the power line is inaccessible. An advantage of EM attacks is that EM emanations leak information not readily available from the power side channel. There are two different types of EM emanations [5]:

Direct Emanation: They result from intentional current flows which consist of short bursts of current with a sharp rising edge. These emanations have wide frequency spectra but the higher frequency bands are more useful due to the lower level of noise. It is often difficult to isolate direct emanation because of interferences from other signals.

Unintentional Emanation: As a consequence of the large number of circuits placed in a very small area, there are many unintentional electrical and electromagnetic couplings between these circuits. These couplings do not influence the function of the device so they are ignored by the designers. These emanations are modulated signals of the carrier signal. The carrier signals can be the clock signal or signals used for internal or external communication. Analyzing the unintentional emanation is more practical because the carrier signals are stronger, and they propagate further. This makes it possible to launch attacks over a very large distance.

Countermeasures against EM attacks are techniques for signal-strength reduction including circuit redesign to reduce egregious unintentional emanations and the use of shielding and physically secured zones to reduce the strength of compromising signals available to an adversary, relative to ambient thermal noise [1].

3.4 Smartcard Components

The microcontroller is the central component of the smartcard. It can be considered as a complete computer. A typical smart-card architecture consists of a communication interface and a CPU for data processing and memory. Some smartcards offer, for example, additional cryptographic functionality, achieved by using special co-processors. ISO 7816 describes a card with an array of 8 contacts but only 6 are connected to the chip (power supplies (V_{CC},V_{PP}), ground, clock, reset and a serial data communication link).

3.4.1 CPU

The used CPUs in smartcards are usually very well studied and reliable. Most of the currently used smartcard microcontrollers still have a 8-bit CPU. If the operating system of the smartcard needs to use an interpreter such as Java a 16-bit CPU can be used and has more processing power. There are also smartcard microprocessors based on well known 32-bit architectures like ARM 7 or MIPS. The limiting factor for the use of these processors is the chip area which results in a higher price. Smartcards have to be low cost, mass-production items [47].

3.4.2 Memory

In order to store inputs of sequential circuits, memory elements are needed. In this section memory is described in more detail because one part of this thesis is to design custom memory for the crypto module. Memory elements consist of the same gates like combinatorial logic. The only difference is that memory elements have feedback loops, where the output is fed back to the input of the element.

The memory part in sequential circuits is implemented mostly using bistable devices. A bistable device has two states, therefore it can only store one bit. It can remain in one of these states until an input signal changes the state of the device. Two types of bistable devices can be distinguished, namely latches and flip-flops. They are similar in many ways but the method by which they change their state is different [34].

Latch

A latch changes its state when the input value changes under the condition that the enable signal is asserted. The output only needs the propagation time delay of the combinatorial gates between input and output to change. This is called transparency and is common to all latches. When the enable signal is disabled the state remains constant.

D Latch Figure 3.1(a) illustrates the principle of a bistable element which consists of two inverters. The output of the first inverter is fed back to the input of the second inverter and the output of the second inverter is taken as input for the first inverter. To change the state of the bistable element the two inverters are simply substituted by two NAND gates. This circuit is called SR Latch, and has the two inputs set and reset (S' and R').

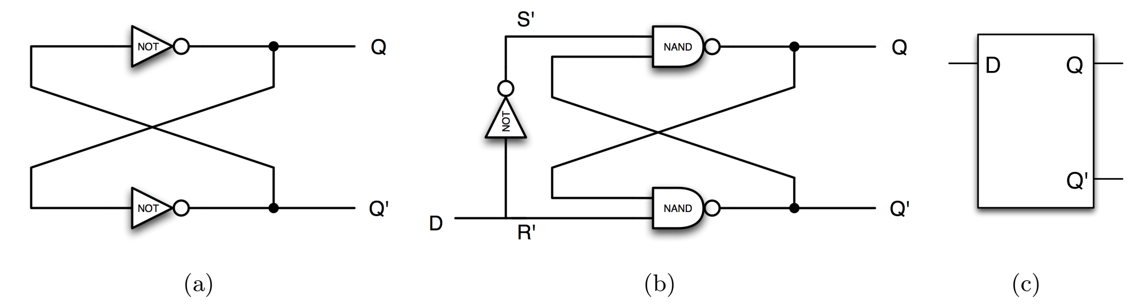


Figure 3.1: (a)Bistable element, (b)D latch, (c)D latch symbol.

When S' and R' are set to 0 both Q and Q' are 1. This state will cause serious problems if in the next state S' and R' are set to 1 at exactly the same time. Q and Q' are both 1 and fed back to the two NAND gates. Both inputs of the two NAND gates are now 1, causing the outputs to become 0. After the two 0's are fed back, the outputs become 1 again. This problem causes an oscillating behavior and is addressed by adding an inverter in a way that $R' = D$ and $S' = D'$ (see Figure 3.1(b)). It can be guaranteed that S' and R' never have the same value. The resulting circuit is called a D latch.

D	Q	Q_{next}	Q'_{next}
0	x	0	1
1	x	1	0

Table 3.1: D latch truth table.

D Latch With Enable The D latch has the disadvantage that it can not keep its state. It can only be set or reset. Consequently, an enable input, which triggers a multiplexer to select whether the D signal or the Q signal is the input of the D latch, is added.

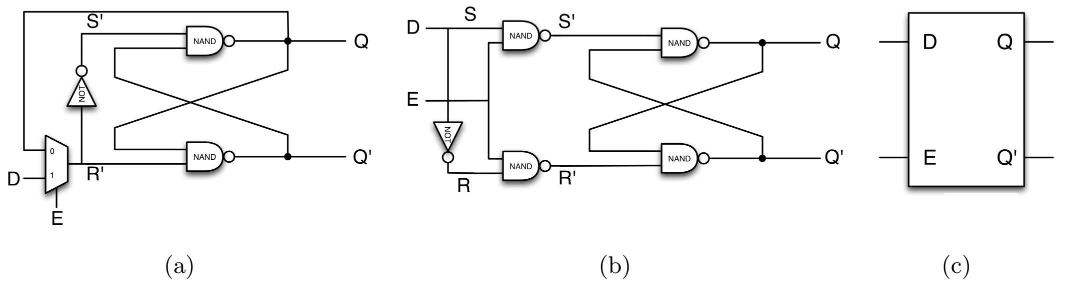


Figure 3.2: (a)D latch with enable, (b)alternative D latch with enable, (c)symbol.

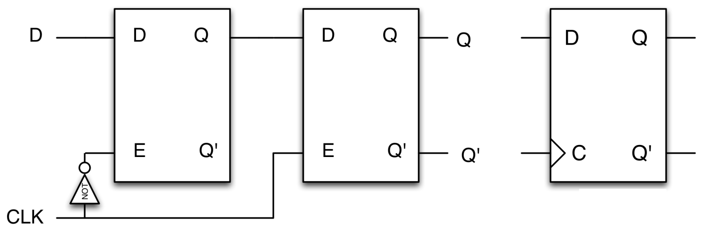
A construction of the D latch with enable at base of the SR latch with enable is also possible. The S and R inputs are simply connected together with a converter so that $R = D'$ and $S = D$.

E	D	Q	Q_{next}	Q'_{next}
0	x	0	0	1
0	x	1	1	0
1	0	x	0	1
1	1	x	1	0

Table 3.2: D latch with enable truth table.

Flip-Flop

Flip-flops do not have the transparency property, therefore flip-flops need a control signal (clock signal) and are able to change the state only when this signal makes a transition. As a consequence, the input of the flip-flop does not influence the state of the flip-flop until the next transition of the clock signal. To be exact, a flip-flop could be triggered either to the positive edge, the negative edge or the positive and negative edge of the clock signal. In microprocessor systems flip-flops are preferable over latches because the changes occur in exactly the same moment controlled by the clock signal.



(a) (b)

Figure 3.3: (a)D flip-flop, (b)D flip-flop symbol.

There are four main types of flip-flops: SR, D, JK and T. They are different in number of inputs and how their internal state changes as well as their circuit size. Every sequential circuit can be built with each of these types of flip-flops but nowadays D-flip-flops are mostly used because they are user friendly. They have a good trade off between usability and circuit size [26].

D Flip-Flop

A D flip-flop consists of two D latches with enable. This flip-flop is positive edge triggered. The first latch is called master and the second is called slave. The output Q of the first latch is the input D of the second latch. The clock signal is the enable signal of the slave latch and the inverted clock signal is the enable signal for the master latch. That is the reason why only one latch at a time can be transparent and therefore, the output of the flip-flop is the input value of the previous cycle.

An alternative method for constructing a D flip-flop requires the use of three interconnected SR latches (see Figure 3.4). The advantage is that this circuit uses fewer gates.

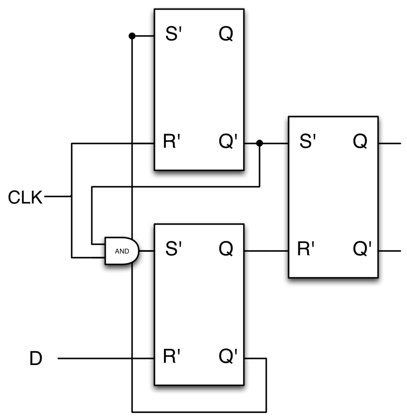


Figure 3.4: Alternative D flip-flop.

To get back the storing ability an enable signal is used that triggers a multiplexer. The two input signals of the multiplexer are D and the fed back output of the slave latch Q.

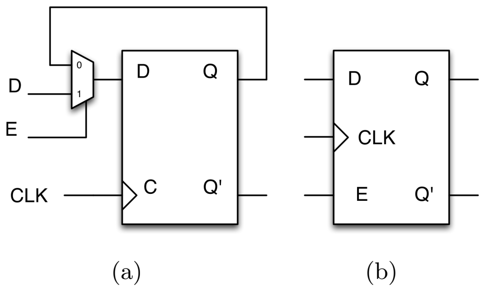


Figure 3.5: (a)D flip-flop with enable, (b)D flip-flop with enable symbol.

RAM

RAM is only a temporary memory which means that after switching off the power supply all data is lost. There is an unlimited number of accesses. The RAM is used for data needed only during the calculation of the smartcard. An exception is an active battery-powered device in which it is possible to store data in the RAM permanently [14].

In smartcards the used type of RAM is static (SRAM). This means that the contents of the RAM do not need to be refreshed periodically such as with dynamic RAM (DRAM). The storage of information is done in a two-dimensional array of memory cells called memory core. Individual cells can be accessed by activating a word line and reading data out on a particular set of bit lines. It is also necessary to have a word decoder that generates the word line signals from a given address [49].

SRAM cells typically use two cross-coupled inverters which form a latch and are able to access transistors. The access transistors are necessary for the read and write process during operation. An SRAM cell is capable of non-destructive reading access, writing capability and can store data as long as the cell is powered [45].

Clock-Gating

An effective design technique for reducing the switching activity in CMOS logic circuits is a conditional gated clock signal. All blocks which are not used during a certain clock cycle do not need a clock signal. This technique saves switching power that is otherwise wasted. For the design of an effective clock-gating strategy, it is necessary that the signal flow and the interrelations among the various operations performed by the circuit are analyzed carefully [29].

A distinction is drawn between two kinds of clock gating, global clock gating and local clock gating. In global clock gating the gating logic is global while in local clock gating the clock gating logic is included in each hardware module. When using clock gating it should be kept in mind that the power consumption of the clock gating cells is below the power saved through using this technique. Extra caution has to be taken because timing analysis becomes more complicated when using clock-gating. In a normal D flip-flop with enable an enable signal triggers a multiplexer whether D is switched to the input or to the fed back output Q. Using clock gating this multiplexer is no longer necessary. An AND gate is used instead and interconnected before the clock input of the flip-flop. Consequently, it is possible for the flip-flop to store the input D only when E and the clock is 1. Otherwise the previous state is saved [44].

In real world circuits an additional D latch is required to ensure that there are no unwanted transient states because the enable signal and the clock signal are not perfectly synchronized (see Figure 3.6).

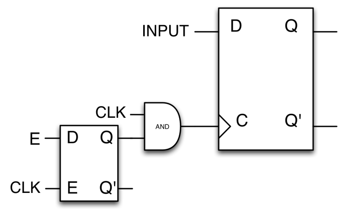


Figure 3.6: Clock gating.

Chapter 4

Implementation

This chapter deals with the implementation of a low-power AES-128 processor with a 32-bit wide datapath. Additionally, the cryptographic device also offers the functionality of a PRNG. The architecture is kept as modular as possible to provide an easy integration of further cryptographic algorithms like ECC or SHA. The first section gives an overview about the design methodologies and concepts used for modern chip design (see Section 4.1). The high-level model (see Section 4.2) is then used to generate test data and estimate the run-time. Finally, the hardware architecture is described (see Section 4.3). A separate section is dedicated for each key component of the hardware architecture:

- AMBA interface (see Section 4.4)
- AES algorithm (see Section 4.5)
- Grain algorithm (see Section 4.6)
- RAM (see Section 4.7)

4.1 Design Flow

The ideas of this section are primarily taken from the two books *Modern VLSI Design* [61] and *CMOS VLSI Design* [60].
The problem in modern chip design is that the complexity increases very fast. While the number of transistors continues to increase, the designer must maintain the productivity. There are several design principles that a designer has to keep in mind. Abstraction is the key word for hardware developers. There are several abstraction layers in which only the issues of the specific layer will be covered:

System level: The overall functionality and system requirements are defined.

Behavioral level: The Algorithm is defined.

Architectural level: Partitioning is performed.

Register-transfer level: HDL is used to create a cycle-accurate model.

Circuit level: Technology mapping is performed.

A short overview of the main design principles follows in Section 4.1.1. The design flow should be as simple as possible. Synchronous models are easier to implement than asynchronous models. The integration of IP modules (see Section 4.1.2) is encouraged and modules should be designed in a way that promotes reuse. There is a wide range of automated tools used to speed up the whole design process, namely:

- Implementation and synthesis
- Analysis and verification
- Testability and manufacturability

Section 4.1.7 describes the IAIK design flow used for this work.

4.1.1 Design Principles

There are several design principles that should be obeyed from hardware designers to minimize the time and effort for successful hardware design.

Hierarchy

An approach to successfully designing complex systems involves splitting up a large, difficult problem into smaller, simpler problems. Therefore, a design hierarchy is created. "Divide and conquer" is this approach's motto. Every module consists of smaller submodules. The process continues on every submodule until the complexity of the modules reaches a reasonable level. Figure 4.1 presents an example showing the hierarchy of the crypto module.

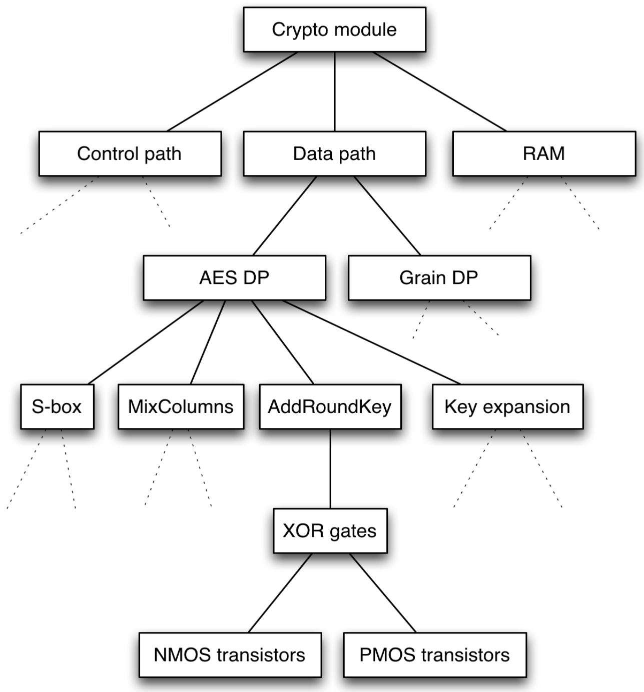


Figure 4.1: Hierarchy of the crypto module.

When using hierarchy the approach should always be top-down, which means splitting up a bigger problem into smaller ones. On the contrary, the bottom-up approach is like playing Lego. It is nearly impossible to create a complex system by experimenting with logic gates.

Regularity

Regularity means that the hierarchy is divided into similar building blocks. It can be used at any hierarchical level:

Circuit level: Uniformly sized transistors are used.

Gate level: Standard cells from a library with fixed height and variable width are used.

Logic level: Parameterized RAMs and ROMs are used in different places.

Architectural level: Multiple identical processors are used.

By using the concept of regularity, the number of different submodules is limited to a minimum. Reusing a design is also supported.

Modularity

The principle of modularity implies that every individual module has a defined interface and function. The process of putting together modules is a much easier task after following the principle of modularity. Apart from the defined interface and function, electrical and timing constraints should also be defined.

Locality

In the software world locality means the reduction of global variables. Data should be processed in the appropriate modules and complexity should be hidden inside the modules as long as it is not needed by other modules. Locality can also mean temporal locality, implying that all signals refer to a clock. Input signals with required setup and hold times as well as delayed outputs are related to the edge of the clock.

4.1.2 Intellectual Property

Systems-on-chip are often very complex systems consisting of a wide variety of different modules. Due to time limitations, it is often impossible to create all the required components from scratch. Most SoCs would not be ready in time for the market. Consequently, intellectual property (IP) blocks are used instead. IP blocks are pre-designed modules which can be used in large systems to increase the productivity. There are two distinctions between types of IP blocks:

Hard IP: It is a physical layout which is fabricated in a fixed target technology. Therefore, it can not be tailored for another target technology. They are full custom circuits based on bus communication.

Soft IP: It is a VHDL or Verilog module which can be synthesized to the desired target technology.

The companies that provide IP modules protect them from being altered or from extracting their know how. The companies do not deliver the behavioral model for soft IPs, whereas for hard IP modules this is done implicitly. They provide as little information as possible. When using IP modules it is critical to ensure that the IP block works properly in context of the whole system. IP modules include the following blocks:

- CPUs
- Memory
- I/O devices
- Cryptographic algorithms

4.1.3 High-Level Model

The high-level model is fast to construct and describes the functionality of the system. It is used to evaluate different algorithms. The high-level model is not cycle-accurate and has only an input-output relationship. Apart from that, it aids the generation of test data for the HDL model. The most frequently used programming languages are:

- Matlab
- Java
- C++

4.1.4 Hardware Description Language

HDLs are considered to be the most important tools used to describe hardware. They support the use of higher levels of abstraction perfectly. The most used hardware description languages are Verilog and VHDL. Both are based on the same underlying key concepts. Verilog and VHDL were actually invented for efficient simulations of digital systems. They have been designed to be executed.

A very important feature is that they are able to describe parallel execution of hardware components. This feature lacks a sequential programming language. For example, when a C source code is analyzed, every line is processed sequentially in a fixed order. By contrast, an HDL language offers the functionality of describing a few logic gates which change their outputs simultaneously.

Another feature of HDL languages is that time in the simulations is measured in discrete time units (often nanoseconds are used). As a consequence, it is possible to simulate things like gate delays and clock cycles. VHDL and Verilog are simulation languages which are built on top of event-driven simulators. In a circuit not every net changes its state every clock cycle. Event driven simulation keeps track of this fact and ignores inactive nets. Every event consists of two parts specifically a value and a time. In order to determine the state of the whole system at a desired time, the simulator must record every event.

HDL languages provide the possibility to test the simulated hardware creating a test-bench. Input vectors can be created and the output vectors can be compared to the expected output vectors. The IAIK design flow (see Section 4.1.7) does not use test-benches for simulation purposes. TCL scripts are used instead. Apart from the possibility to simulate hardware models, HDL languages provide a synthesis subset. This subset is used to create an HDL model which can be synthesized on the logic gates abstraction layer. The compiler is able to synthesize the model only when this subset is used . Furthermore, the synthesized model can be checked for correct functionality.

Register-transfer synthesis is the most commonly used approach. Here logic synthesis is used to optimize the combinational logic blocks and the synthesizer chooses the appropriate flip-flops for the registers placed by the designer. Two coding approaches are distinguished:

Behavioral HDL: It describes what a cell does and how an output is computed as a function of inputs. Sequential statements are used.

Structural HDL: It describes how a cell is placed in relation to other cells or primitive gates. It reflects the architecture and hierarchy with modules and submodules.

Because mainly Verilog is used to implement the crypto chip, the following section provides a short overview of the hardware description language Verilog.

Verilog

On contrary to VHDL, Verilog has a much simpler syntax. Verilog uses two types of assignments:

Continuous assignments: These assignments are used for combinational logic. The output only depends on the inputs and combinational logic is memoryless.

Always blocks: They can be used for combinational logic as well as sequential logic.

Listing 4.1 shows the Verilog model of a full adder. In line 1 the ports of the module with the name *adder* are defined. The following two lines define which of the ports is an input and which is an output. In line 5 and 6 continuous assigns are used. They always drive values to the net, independent of whether the inputs have changed.

```
1 module adder(a,b,cin,sum,cout);
2    input a, b, cin;
3    output sum, cout;
4
5    assign sum = a ^ b ^ cin;
6    assign cout = (a and b) or (a and cin) or (b and cin);
7
8 endmodule;
```

Listing 4.1: Verilog model of a full adder.

4.1.5 Design Styles

Design styles can be distinguished into three different design styles:

1. Field programmable gate array (FPGA)
2. Standard cell
3. Full custom

The advantage of the FPGA design is that it has the shortest design time, with the drawback of the lowest performance and the largest chip size. On the other hand, the full custom design has a good performance and a small chip size but a high design time. The standard cell design can be considered as a good compromise between chip size, performance and design time. The style used for this work is a standard cell design.

FPGA

As the name suggests FPGAs are freely programmable. They are composed of configurable logic blocks (CLBs). These CLBs are arranged in arrays and surrounded by programmable routing resources. Apart from that, FPGAs also consist of other components such as RAMs and ROMs.

Standard Cell

Standard cells are arranged in rows of equal height. Only the width of the cell varies depending on its functionality (inverter, NAND, NOR, flip-flops, ...). Standard cells have two metal layers for routing channels. Apart from that, these cells are available with different output strengths. The manufacturer provides libraries with standard cells. This includes data for synthesis, timing and power characterization as well as simulation models. A large amount of effort is put into the design and the optimization of standard cells, because they are used millions of times in various chips.

Full Custom

Full custom design is used when the focus lies on optimization. When developing circuits using full custom design, the time and attention required is very high. Consequently, it only pays off if the chip is mass-produced. This approach offers advantages such as a very dense layout and the possibility of a free logic style. It is often used for regular layouts like a datapath with bit-slice architecture or memories.

4.1.6 Simulation

Simulation is used before the chip is manufactured in order to prove the correct functionality. It is very important to find errors in the chip design as early as possible. Since it is impossible to test all possible cases, a subset of test patterns is generated to test the device. Often randomly generated test vectors are used for the simulations. A high-level model is used to generate the test data.

4.1.7 IAIK Design Flow

The IAIK design flow is an implementation of a workflow in consideration of the principles and ideas presented in Section 4.1. It supports the implementation of mixed Verilog/VHDL models on CMOS standard cells (0.35 μm, 0.25 μm, 180 nm, 130 nm, 95 nm) as well as on Xilinx and Altera FPGAs. The design flow covers the whole spectrum of abstraction layers from high-level description to physical implementation (see Figure 4.2). It provides tools for synthesis, layout generation, verification and simulation. The design flow is $Makefile$ based. There are several $Makefiles$ which support:

- Compiling and simulating the high-level model
- Compiling and simulating the HDL module
- Synthesizing the module and simulating the synthesized module
- Place and route and simulating the place and route results
- Various design checks
- Power simulation

For the different target technologies different $Makefiles$ exist. Apart from the $Makefiles$, TCL ("Tool Command Language") scripts are used between the different tools. The output of one tool needs to be altered by a TCL script before it can serve as input for another tool. Another function of these scripts is to adjust parameters using the TCL interface of the tools. TCL is a scripting language which is very flexible and offers the same constructs as common programming languages.

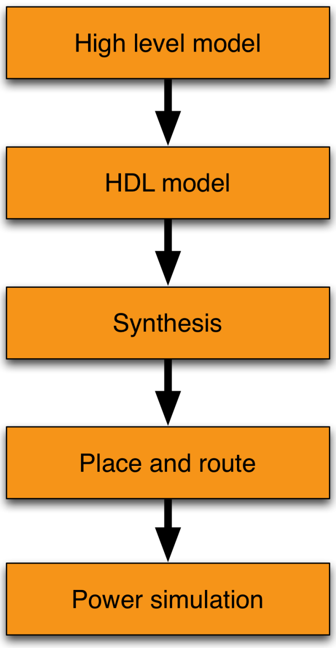


Figure 4.2: IAIK design flow.

High-Level Model

The design flow is able to cope with every programming language for the high-level model. The first high-level model is only an executable specification with a fixed input-output relation. Later the model is refined and different algorithms are evaluated. The source code is then compiled and the output is written to generate test data.

Register-Transfer Model

This model can be written either in Verilog, VHDL or both. It is cycle- and bit-accurate and can be simulated. The next step is to synthesize the HDL description to a structural representation on the gate level. A detailed netlist of cell instances for a specific target technology is created. The synthesis result is influenced by constraints such as area, performance and power consumption. The place and route process places the cells to minimize the wire length and inserts the clock tree.

Verification

Verification is of great importance in this design flow. In every design step the correct functionality should be proved as well as whether the constraints are met. There are two distinctions, the distinction between dynamic verification (simulation) and static verification (equivalence checking, equivalence proofing).

Simulation Test vectors from the high-level model are used to obtain waveforms. Therefore, the correct functionality can be tested. In addition, timing information and power consumption can be determined. There are two different types of simulation:

HDL simulation: Either behavior level or gate level is used for the simulation.

Spice simulation: Transistor level is used for the simulation.

The Spice simulation is very accurate but also very slow. A reasonable compromise is the use of transistor-level simulation with "Near Spice accuracy". Several simplifications help to lower the simulation time. The $\frac{U}{I}$ relationship is stored in a table. Additionally, the capacitance between nodes is not considered and the output does not influence the input. This approach provides accurate results for timing and power simulations.

The HDL simulator has a TCL interface. TCL script is used to input the test data generated from the high-level java model to the HDL model. The TCL script checks if the HDL model is functionally equal to the high-level model. This verification using TCL scripts is performed only after the following design steps:

- Compilation of the HDL model
- Synthesis
- Place and route

Static Verification The correct functionality can be ensured by comparing two different representations:

- RTL- RTL
- RTL- gates
- Gates- gates
- Gates- layout

It is also possible to analyze whether the layout and geometric requirements have been met. This test is called design rule check (DRC). After passing the DRC a layout can be fabricated without problems. A layout versus schematic check (LVS) ensures that a circuit layout corresponds to the original schematic.

4.2 High-Level Model

The high-level model is written in Java. According to Section 4.1 it is taken to generate test data for the verification of the HDL modules. The crypto module offers the functionality of the AES and Grain algorithms. As a consequence, the high-level model is split up into two parts which reflect the two cryptographic algorithms AES (see Section 4.2.1) and Grain (see Section 4.2.2). First runtime estimates are possible (see Section 4.2.3). Since the datapath is required to be 32-bit wide and a common RAM is used, the high-level model implementation looks very much like a software solution.

4.2.1 AES

The core AES function for the encryption is the cipher function shown in Listing 4.2. It is the highest abstraction layer of the AES algorithm. The sub functions are:

1. AddRoundKey
2. SubBytes
3. MixColumns
4. Key expansion

ShiftRows are done implicitly (see Section 4.5.1). The high-level model reflects the hardware implementation fairly well. Each of these sub functions are processed in a block of four lines. Each of these lines processes 32 bits of data. This sums up to the 128 bits of the AES state. The result is stored in a 32-bit variable and there are always two 32-bit values as inputs. This corresponds to the dual output port RAM with one input which will be used in the hardware model. The implementation is very straight forward. From this high-level sequential model, the state diagram for the controlpath of the HDL model can easily be created.

The implementation of the countermeasures against power analysis attacks is not discussed here because it is described in detail in Section 4.5.5.

```
public void doCipher()
{
  int round=0;

  state[0]=addroundkey.perform(state[0], key[0]);
  state[1]=addroundkey.perform(state[1], key[1]);
  state[2]=addroundkey.perform(state[2], key[2]);
  state[3]=addroundkey.perform(state[3], key[3]);

  for(round=1;round<10;round++)
  {
    state[4]=subbytes.perform((state[0]&0xFF00FF00)| (state[1]&0x00FF00FF));
    state[5]=subbytes.perform((state[1]&0xFF00FF00)| (state[2]&0x00FF00FF));
    state[6]=subbytes.perform((state[2]&0xFF00FF00)| (state[3]&0x00FF00FF));
    state[7]=subbytes.perform((state[3]&0xFF00FF00)| (state[0]&0x00FF00FF));

    state[0]=mixcol.perform((state[4]&0xFFFF0000)|(state[6]&0x0000FFFF),0);
    state[1]=mixcol.perform((state[5]&0xFFFF0000)|(state[7]&0x0000FFFF),0);
    state[2]=mixcol.perform((state[6]&0xFFFF0000)|(state[4]&0x0000FFFF),0);
    state[3]=mixcol.perform((state[7]&0xFFFF0000)|(state[5]&0x0000FFFF),0);

    key[0]=keyexpansion.expandKey(oldkey,round,0);
    key[1]=keyexpansion.expandKey(oldkey,round,1);
    key[2]=keyexpansion.expandKey(oldkey,round,2);
    key[3]=keyexpansion.expandKey(oldkey,round,3);

    state[0]=addroundkey.perform(state[0],key[0]);
    state[1]=addroundkey.perform(state[1],key[1]);
    state[2]=addroundkey.perform(state[2],key[2]);
    state[3]=addroundkey.perform(state[3],key[3]);
  }
  round=10;

  state[4]=subbytes.perform((state[0]&0xFF00FF00)| (state[1]&0x00FF00FF));
  state[5]=subbytes.perform((state[1]&0xFF00FF00)| (state[2]&0x00FF00FF));
  state[6]=subbytes.perform((state[2]&0xFF00FF00)| (state[3]&0x00FF00FF));
  state[7]=subbytes.perform((state[3]&0xFF00FF00)| (state[0]&0x00FF00FF));

  //last shift rows need to be explicit because in the
  //last round there is no MixColumns
  state[0]=(state[4]&0xFFFF0000)|(state[6]&0x0000FFFF);
  state[1]=(state[5]&0xFFFF0000)|(state[7]&0x0000FFFF);
  state[2]=(state[6]&0xFFFF0000)|(state[4]&0x0000FFFF);
  state[3]=(state[7]&0xFFFF0000)|(state[5]&0x0000FFFF);

  key[0]=keyexpansion.expandKey(oldkey,round,0);
  key[1]=keyexpansion.expandKey(oldkey,round,1);
  key[2]=keyexpansion.expandKey(oldkey,round,2);
  key[3]=keyexpansion.expandKey(oldkey,round,3);

  state[0]=addroundkey.perform(state[0],key[0]);
  state[1]=addroundkey.perform(state[1],key[1]);
  state[2]=addroundkey.perform(state[2],key[2]);
  state[3]=addroundkey.perform(state[3],key[3]);
}
```

Listing 4.2: AES high-level model code snippet.

4.2.2 Grain

Contrary to the implementation of the AES algorithm which supports a 32-bit implementation very well, the implementation of the Grain algorithm is much more complex. The Grain algorithm is a pure hardware-oriented stream cipher algorithm consisting of two 80-bit feedback shift registers (NFSR and LFSR). Consequently, it is only expected to output one bit at a time. It is possible to calculate 16 output bits at once, but due to data dependencies the required 32 bits are impossible. Hence, the sub functions $F(x)$, $H(x)$, $G(x)$ and $Z(x)$ are also 16-bit wide.

The challenge is to provide the right data for the sub functions because it is not feasible to access every bit of the 160-bit state of the Grain algorithm at a single time. Due to the dual-port RAM, 64 bits are possible at once. Therefore, a 32-bit register and a 1-bit register are added and increase the accessible bits to a number of 97. Then every sub function can be calculated, but can not be processed simultaneously. $F(x)$ and $H(x)$ are calculated first because primarily the data from the LFSR is needed and then the data of primarily the NFSR is loaded to calculate $G(x)$ and $Z(x)$.

4.2.3 Estimates

The timing estimates of the AES algorithm are easy to determine. The assumption is that one line of the source code in which a sub function is processed takes one clock cycle. The whole AES encryption algorithm takes 164 cycles. The number increases to 861 cycles when using dummy cycles. For the decryption the number of clock cycles is 240 and 1090 with dummy cycles. For the Grain algorithm the estimates are 80 clock cycles for the initialization process and 18 clock cycles to receive 32 bits of the key stream.

4.3 Hardware Architecture

The architecture of the crypto module can be seen in Figure 4.3.

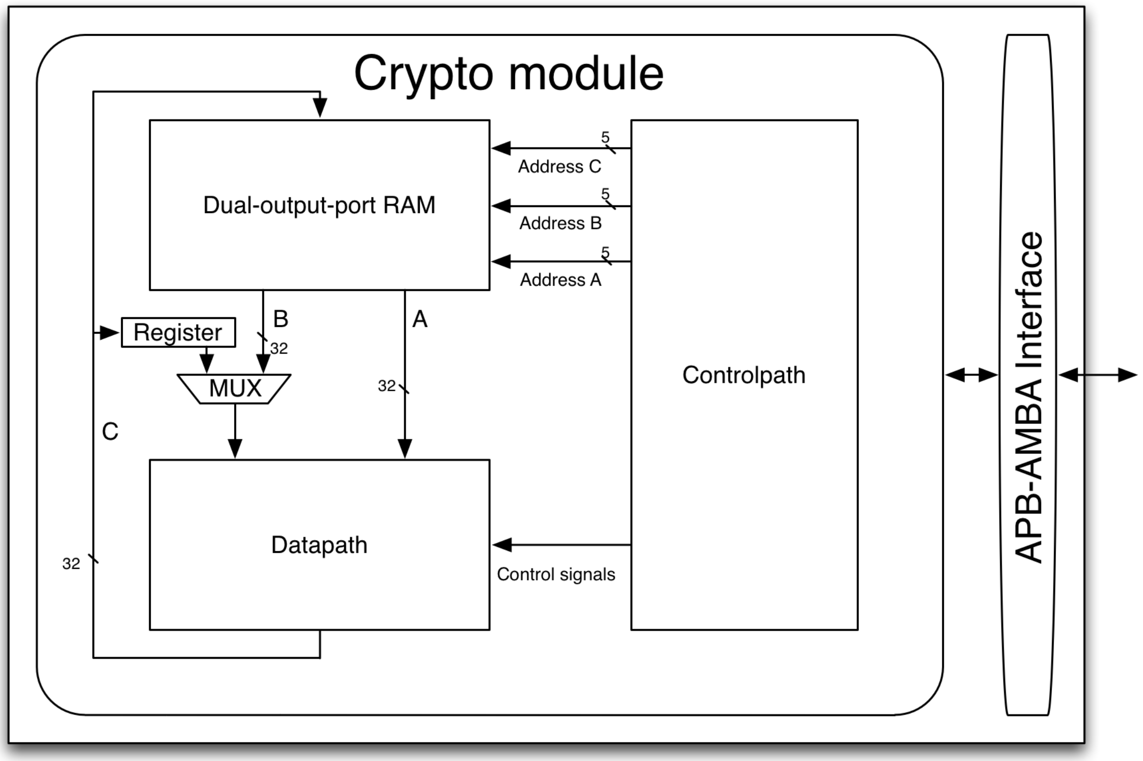


Figure 4.3: Hardware architecture of the crypto module.

The crypto module consists of three main components:

- Controlpath (see Section 4.3.1)
- Datapath (see Section 4.3.2)
- Dual output port RAM (see Section 4.7)

The crypto module can be easily accessed using an APB AMBA interface (see Section 4.4).

When writing a result from the datapath to the RAM at the time t cycles, a problem occurs if the result is needed at the input of the datapath at the time $t + 1$ cycles. The RAM needs two cycles to provide the new result. As a consequence, a 32-bit wide register is added and stores the output of the datapath of the previous cycle. This register can be selected by a multiplexer to be the input of the datapath.

4.3.1 Controlpath

The specific state machines of the different cryptographic algorithms are implemented in the controlpath. The desired state machine is chosen by setting the operation mode through the AMBA interface. Therefore, multiplexers are used. Every state creates different control signals for the datapath and the RAM.

Concerning the RAM, these control signals include one write and two read addresses in addition to the corresponding write-enable and two read-enable signals. The standard form of a finite state machine can be shown in Figure 4.4. The state machine consists of combinational logic and flip-flops. It has inputs x_i and outputs y_i. The flip-flops are used to hold the current state [55].

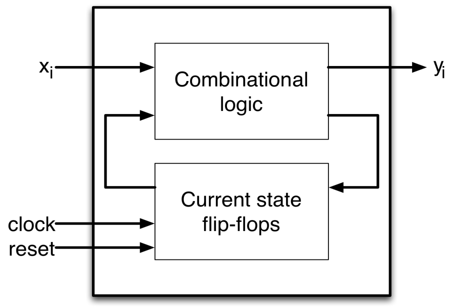


Figure 4.4: Finite state machine.

There are two approaches for the combinational logic to calculate the next state:

Moore machine: The combinational logic only uses the content of the flip-flops to determine the next state.

Mealy machine: The combinational logic uses the flip-flops and the inputs x_i to calculate the next state.

The implemented state machines of the cryptographic algorithms are Moore machines. The controlpath consists of some multiplexers to choose the desired cryptographic algorithm depending on the operating mode. Either the AES controlpath (see Section 4.5.6) or the Grain controlpath (see Section 4.6.3) can be chosen.

Several possibilities exist to encode the state of the finite state machine:

Binary encoding: The state in the flip-flops is represented in binary code such as 000 $\rightarrow$ 001 $\rightarrow$ 010.

One-hot encoding: Only one flip-flop at a time is 1, the others are 0 (001 $\rightarrow$ 010 $\rightarrow$ 100).

One-cold encoding: Only one flip-flop at a time is 0, the others are 1 (110 $\rightarrow$ 101 $\rightarrow$ 011).

Gray encoding: The state in the flip-flops is represented in Gray code such as 000 $\rightarrow$ 001 $\rightarrow$ 011.

One-hot encoding and one-cold encoding are very fast because a transition to another state is simply a bit-shifting operation and therefore no combinational logic is necessary. Binary encoding is used for the controlpath of the crypto module. Due to the large size of the finite state machine, one-hot encoding can not be used because it would result in the hardware being very large since every state requires its own flip-flop.

4.3.2 Datapath

Figure 4.5 shows an overview of the datapath. The datapath consists of two submodules which are the AES datapath and the Grain datapath. A multiplexer is used to choose the output of the desired datapath.

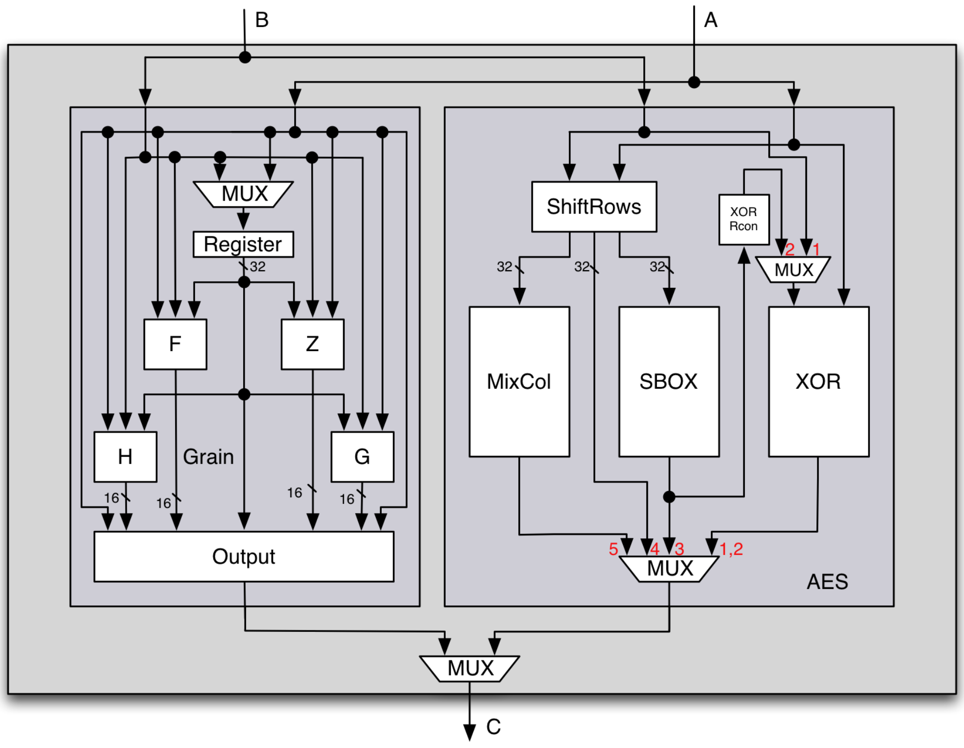


Figure 4.5: Hardware architecture of the datapath.

The AES datapath can process five operation modes:

1. AddRoundKey
2. Key expansion (see Section 4.5.3)
3. SubBytes (see Section 4.5.2)
4. Explicit ShiftRows (see Section 4.5.1)
5. MixColumns (see Section 4.5.4)

The Grain datapath has nine output modes:

1. $F(x)$ and $H(x)$
2. $G(x)$ and $Z(x)$
3. Five output modes which provide the possibility to compose the output of different half words from the two 32-bit inputs and the 32-bit register
4. Two output modes which allow the update of NFSR and LFSR

4.4 AMBA Interface

The Advanced Microcontroller Bus Architecture [3] describes three different on-chip interconnects:

Advanced High-performance Bus: AHB is an AMBA bus intended for high performance and high clock cycle designs.

Advanced System Bus: The application areas of the ASB bus are high performance 16- and 32-bit embedded microcontrollers.

AMBA APB: This interface is optimized for minimal power consumption and low interface complexity, and is therefore used in this work.

4.4.1 AMBA APB

According to [3] any signal transition is only related to the rising edge of the clock cycle. The ABP interface can easily be integrated in any design flow.

4.4.2 Interface Description

The slave interface description is shown in Figure 4.6. The signal PSELx is used to select the desired slave device. When the signal PENABLE is set to 1, the slave device has to either update the selected register from PWDATA or provide the state of the selected register to PRDATA depending on the PWRITE signal. PADDR is used to select the desired register for reading or writing. PRESETn is used to reset the interface. This signal is active low. PCLK is the clock signal. A signal transition is only allowed at the positive edge of the clock signal.

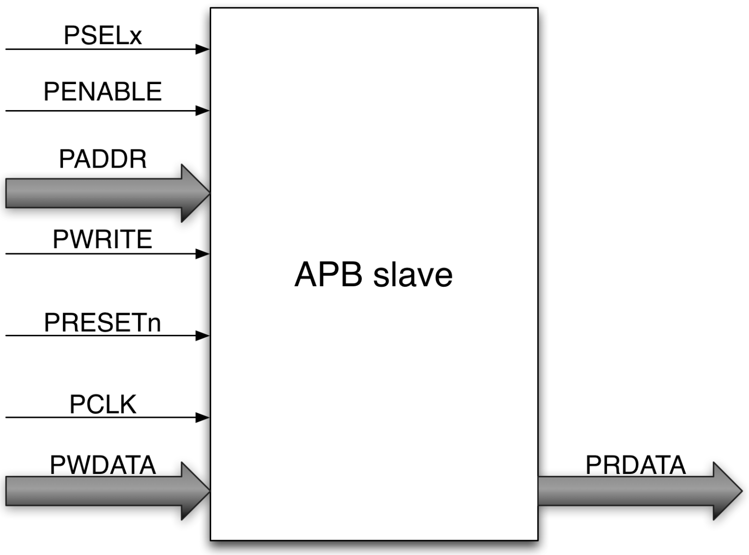


Figure 4.6: APB slave interface description.

4.4.3 State Diagram

Considering the state diagram (see Figure 4.7) the ABP-Bus can be described using three states:

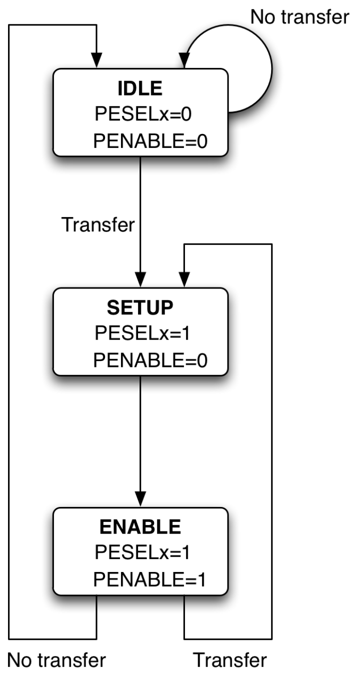


Figure 4.7: APB state diagram.

IDLE: It is the default state and no transmission takes place.
$PSELx = 0$ and $PENABLE = 0$.

SETUP: A transfer is required and $PSELx = 1$ and $PENABLE = 0$. This state always remains for one clock cycle and the following state will always be ENABLE.

ENABLE: PSELx and PENABLE are set to 1. The address, write and select signals have to be stable during the SETUP to ENABLE transition. The ENABLE state lasts also only one clock cycle and if no further transfer is needed, the next state will be IDLE or SETUP.

4.4.4 Write Transfer

The write transfer (see Figure 4.8) lasts two cycles and starts with the SETUP state. After the rising edge of the SETUP state the address, write data, write and select signal are set. The write signal PWRITE must be set to 1. At the following rising edge the PENABLE signal is set to one, indicating the beginning of the ENABLE state. The transfer is completed at the end of the cycle. A measure to reduce the power consumption is not to change the address signal as well as the write signal after the transfer until the next bus access occurs.

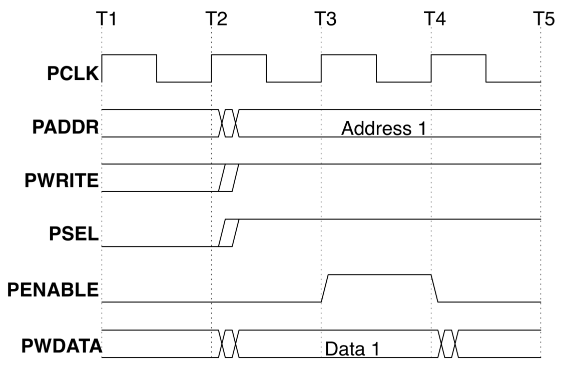


Figure 4.8: APB write transfer.

4.4.5 Read Transfer

The read transfer (see Figure 4.9) works the same way as the write transfer except that the PWRITE signal is set to 0 and the slave has to provide the requested data. The data is sampled at the rising edge of the clock at the end of the ENABLE cycle.

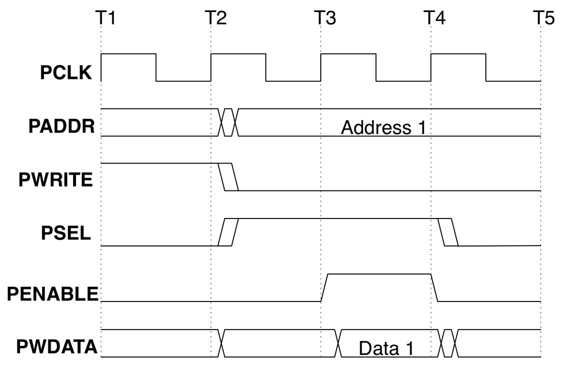


Figure 4.9: APB read transfer.

4.4.6 Implementation Details

As a consequence of the 32-bit wide datapath, PWDATA and PRDATA are both 32-bit as well. In this implementation a 6-bit register holds the PADDR value. Figure 4.10 illustrates the memory map of the AMBA interface.

The values from $0 - 23$ are mapped to the addresses of the RAM for reading or writing accesses. The value 62 is used to set the registers for the countermeasures against power analysis attacks. PWDATA[1:0] sets the shuffling register while PWDATA[5:2] sets the dummy operations register. Finally, the value 63 is used for setting the operation mode for the crypto module. Therefore, PWDATA[2:0] is used. The following operation modes are supported:

AESENC: AES algorithm ECB-mode encryption (PWDATA[2:0]=001)

AESDEC: AES algorithm ECB-mode decryption (PWDATA[2:0]=010)

AESCBCENC: AES algorithm CBC-mode encryption (PWDATA[2:0]=011)

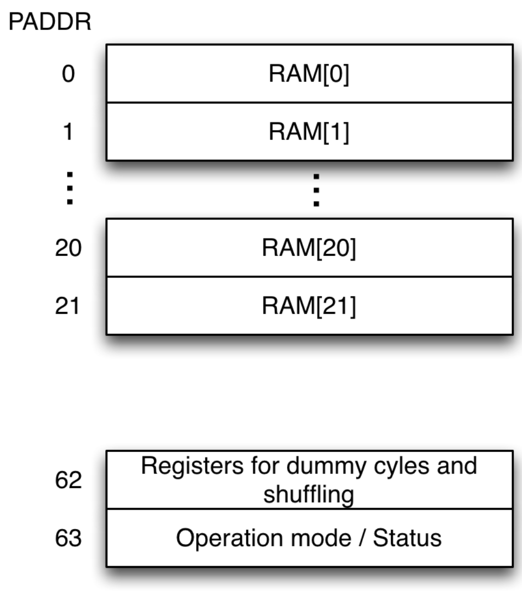


Figure 4.10: Memory map of the AMBA interface.

AESCBCDEC: AES algorithm CBC-mode decryption (PWDATA[2:0]=100)

GRAININIT: Grain algorithm initialization (PWDATA[2:0]=111)

GRAIN: Grain algorithm: get 32-bits of the key stream (PWDATA[2:0]=110)

Furthermore, PWDATA[3] sets the start signal and PWDATA[4] sets the reset signal. Table 4.1 shows the command sequence which is necessary to start the AES algorithm ECB-mode encryption. The binary representation is written in brackets. First the value 9 is written to the AMBA interface at the address 63. The three least significant bits determine the operation mode while the fourth least significant bit starts the AES algorithm. The start bit has to be set to 0 at the next AMBA access, but the mode must remain the same until the algorithm finishes.

PADDR	PWDATA
63	9 (0 1 001)
63	1 (0 0 001)

Table 4.1: Steps necessary to start the AES algorithm ECB-mode encryption.

While in reading mode PWDATA[0] contains the status of the crypto module. Two states are distinguished:

- Ready: PWDATA[0] $\rightarrow 1$
- Not ready: PWDATA[0] $\rightarrow 0$

4.5 AES

This section deals with the implementation of the AES algorithm. As a consequence of the 32-bit-data-path implementation of the AES algorithm, every sub-function must be executed four times (see Figure 4.11) to process the whole 128-bit AES state. The implementation of the sub-functions is described in this section. The ShiftRows operation is performed implicitly (see Section 4.5.1). According to the AES state diagram, there are four AddRoundKey (yellow) executions. The AddRoundKey sub-function is simply implemented using a 32-bit wide XOR gate. The inputs are 32-bit of the AES state and the current 32-bit round key.

The first round starts with four times the SubBytes operation (red) (see Section 4.5.2), followed by four times MixColumns (cyan) (see Section 4.5.4), four times the key expansion (orange) (see Section 4.5.3) and four times AddRoundKey. These four times four AES sub-functions form one round which is executed nine times. The tenth time the round is altered in a way that the four MixColumns operations are substituted by four times the ShiftRows operations (green), which are simply the second parts of the implicit ShiftRows operations 4.5.1. Because there are no MixColumns operations in the last round, the ShiftRows must be done explicitly.

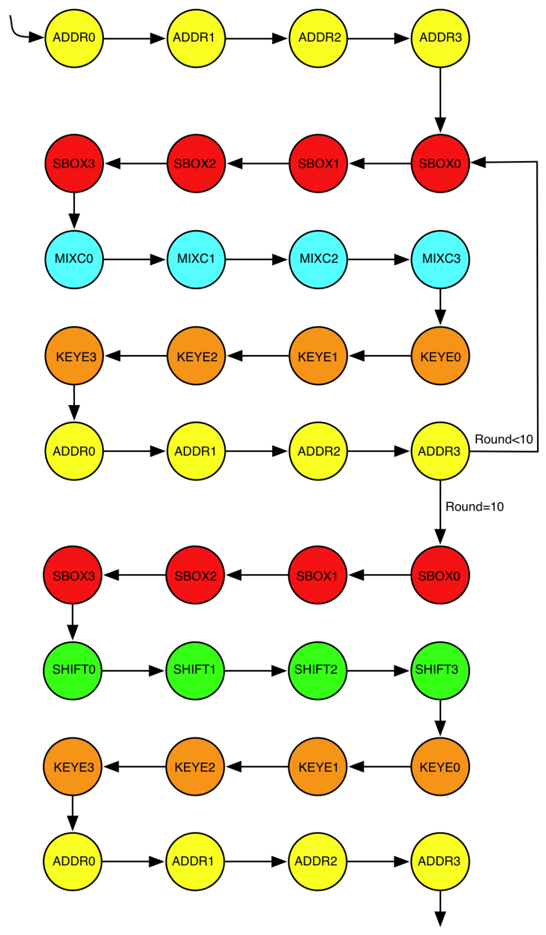


Figure 4.11: AES state diagram for encryption.

In regard to the state diagram of the AES decryption, there are some differences compared to the state diagram of the encryption (see Figure 4.12). First, the four key expansion sub-functions are executed ten times to receive the round key of the tenth round. In contrast to encryption where the round counter increases, here the round counter decreases. Instead of using the same functions of the encryption, SubBytes, MixColumns and the explicit ShiftRows operations are substituted by their inverse counterparts. To perform the inverse key expansion the order of the four key expansion sub-functions is simply reversed. An important requirement is the addition of the four inverse MixKey sub-functions after every inverse key expansion except for the last. The reason is that the inverse equivalent cipher structure is implemented (see Section 2.2.5) and therefore, the inverse MixColumns function must also be performed on the round key.

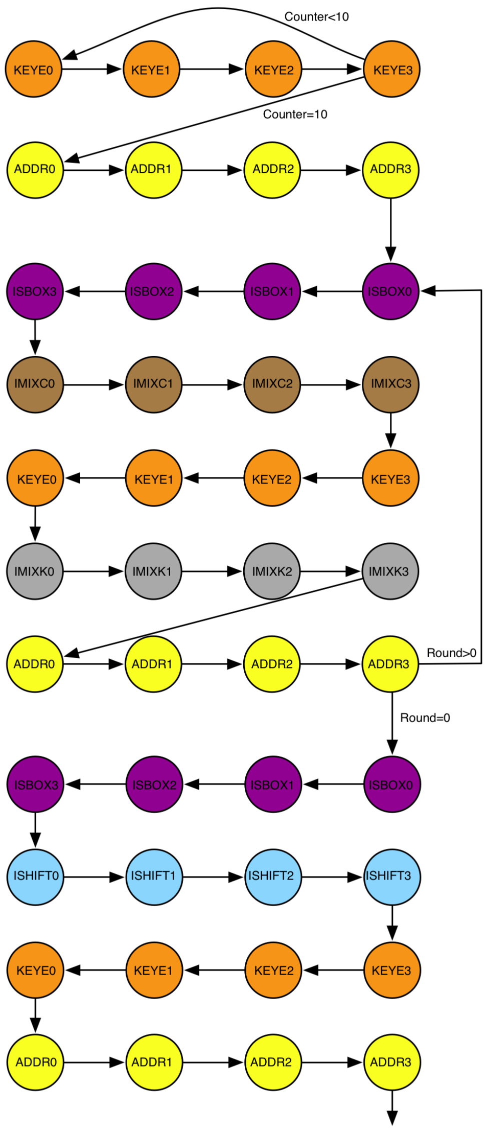


Figure 4.12: AES state diagram for decryption.

4.5.1 ShiftRows

The ShiftRows operation operates row by row (see Section 2.2.5 on page 13). Unfortunately, the AES state is stored column wise in the RAM. That is the reason the ShiftRows operation is the bottleneck because AddRoundKey, SubBytes and MixColumns need only four cycles each. ShiftRows requires about 44 cycles. Tillich [56] proposes a solution to overcome this bottleneck by using a dual read port RAM and implicit ShiftRows. Implicit ShiftRows means that no additional cycles are used for performing the ShiftRows operation. Figure 4.13 shows the implicit ShiftRows in detail.

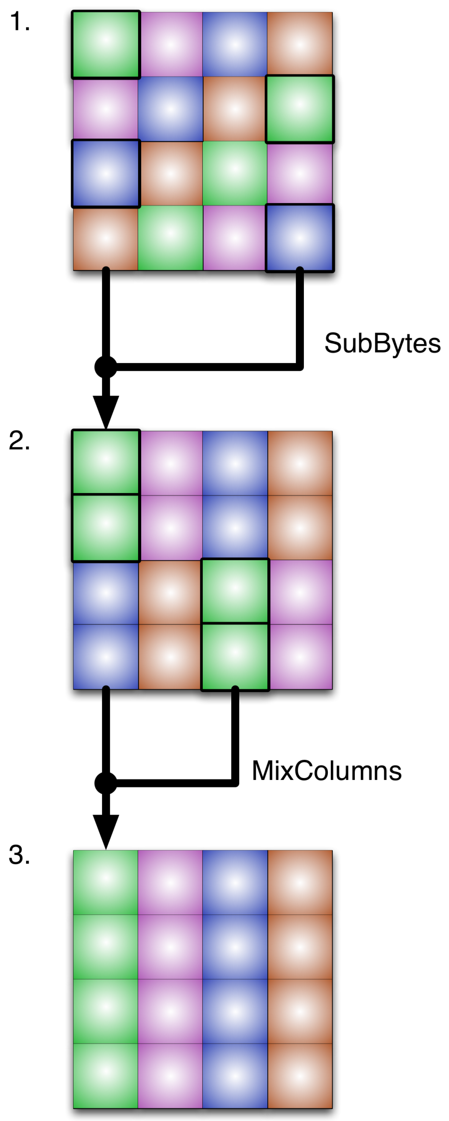


Figure 4.13: Implicit ShiftRows.

The first block symbolizes the initial state which is stored in the RAM at the start of each round. To perform the first of 4 SubBytes steps the first and the fourth column are loaded into the output registers of the RAM. The 4-byte inputs for the SubBytes operation are chosen in the following way:

- First and third byte from the first column
- Second and fourth byte from the fourth column

How the inputs for the other three SubBytes operations are chosen is presented in the figure. The second block shows the state after the four SubBytes operations. The implicit ShiftRows operation is already halfway done. As input for the first MixColumns step the following bytes are taken:

- First and second byte from the first column
- Third and fourth byte from the third column

The third block shows the finished implicit ShiftRows process. The inverse implicit ShiftRows process works exactly the same way but the equivalent inverse cipher structure is required (see Section 2.2.5). The order of the AES sub-functions must remain the same. The disadvantage is that the inverse MixColumns operation must also be performed on the round key, resulting in four additional cycles per round.

4.5.2 SubBytes

The SubBytes operation (see Section 2.2.5 on page 13) consists of two steps:

1. Multiplicative inverse in the finite field $GF(2^8)$
2. Affine transformation

While there is the possibility to solve the SubBytes operation simply by using two look-up tables (one for encryption and one for decryption), each table has 256 one-byte entries. Since the requirements are implementing a 32-bit datapath, the number of look-up tables for encryption plus decryption has to be 8. Another approach is to calculate the S-box values on the fly. The affine transformation is trivial to implement. For the implementation the only gates needed are XOR gates.

The implementation of the multiplicative inverse in the finite field $GF(2^8)$ is the difficult part. One of the best solutions is proposed by Canright [7]. It describes the direct calculation of the S-box function using sub-field arithmetic. The advantage is that the area consumption is relatively small compared to the table approach. The inverse in the finite field $GF(2^8)$ is calculated by breaking down the problem into simpler problems, first into 4-bit operations in $GF(2^4)$ and then into 2-bit operations in $GF(2^2)$, and finally into single bit operations in $GF(2)$. In order to process 32 bits in one clock cycle four Canright S-boxes are connected in parallel.

4.5.3 Key Expansion

The key expansion is needed to calculate the round key for every round (see Section 2.2.5 on page 15). There are two approaches for calculating the round key:

1. Calculating the whole key at the beginning of the AES algorithm
2. Calculating only the round key just when it is needed (on the fly)

The first approach is easier to implement but since there is an additional need for storage of 160 bytes the second approach is preferrable. The on-the-fly key expansion is done in four clock cycles. The four words of the round key are calculated the following way where N denotes the round:

$$RKey[0]_N = RKey[0]_{N-1} \oplus Sbox32(leftrotate(RKey[0]_{N-1})) \oplus rcon(N) \tag{4.1}$$

$$RKey[1]_N = RKey[1]_{N-1} \oplus RKey[0]_N \tag{4.2}$$

$$RKey[2]_N = RKey[2]_{N-1} \oplus RKey[1]_N \tag{4.3}$$

$$RKey[3]_N = RKey[3]_{N-1} \oplus RKey[2]_N \tag{4.4}$$

The four key words must be calculated exactly in the described order. For calculating $RKey[1]_N - RKey[3]_N$ the same 32-bit XOR circuit used for the AddRoundKey operation can be used.

The AES datapath provides its own key expansion mode which is used for calculating the first key word. The $Sbox32$ operation works the same as the SubBytes operation. The $leftrotate$ function means an 8-bit circular left shift. The $rcon(N)$ function consists of values given by $[x^{N-1}, \{00\}, \{00\}, \{00\}]$ where x is $\{02\}$ and is implemented using a look-up table (see Table 4.2). For the lookup table it is only necessary to store the highest byte. To perform the inverse key expansion the order of calculating the key words must be reverted.

N	$rcon(N)$
1	0x01000000
2	0x02000000
3	0x04000000
4	0x08000000
5	0x10000000
6	0x20000000
7	0x40000000
8	0x80000000
9	0x1b000000
10	0x36000000

Table 4.2: $rcon(N)$ look-up table.

4.5.4 MixColumns

For an introduction to the MixColumns operation see Section 2.2.5 on page 14 and page 15 for the inverse MixColumns operation. There are several approaches of implementing the MixColumns operation. For this work the proposed approach by Wolkerstorfer [62] was chosen. The idea is to use the same network for encryption and decryption as shown in Figure 4.14. The basic component is a polynomial multiplier which is placed in parallel four times. The input bytes $a_{0,x} - a_{3,x}$ are wired to the inputs of each of these multipliers. The figure illustrates that the order of the inputs varies depending on the multiplier.

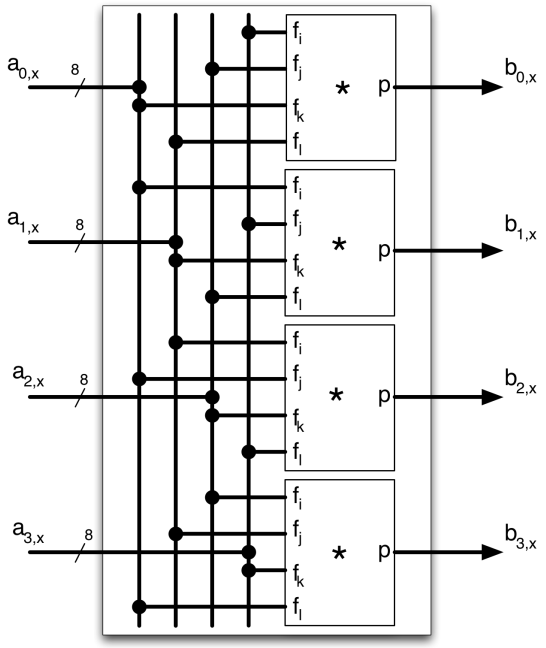


Figure 4.14: MixColumns network.

Figure 4.15 shows the architecture of the polynomial multiplier in detail. For the encryption the polynomial coefficients $\{01\}, \{02\}, \{03\}$ are needed for the multiplication. The $\overline{enc}$ signal is set to 0. As a consequence, the outputs of all AND gates are $0x00$ and do not influence the overall output p. For the decryption the polynomial coefficients $\{09\}, \{0d\}, \{0e\}, \{0b\}$ are needed. Equations 4.5-4.8 show a way to decompose these coefficients. In decryption mode the $\overline{enc}$ signal is set to 1. This causes the outputs of the AND gates to be XORed with the output of the upper branches. The advantage of this approach is that additional to the coefficients for encryption only $\{08\}, \{0c\}$ are necessary.

$$\{09\} \cdot x = \{01\} \cdot x \oplus \{08\} \cdot x \tag{4.5}$$

$$\{0d\} \cdot x = \{01\} \cdot x \oplus \{0c\} \cdot x \tag{4.6}$$

$$\{0e\} \cdot x = \{02\} \cdot x \oplus \{0c\} \cdot x \tag{4.7}$$

$$\{0b\} \cdot x = \{03\} \cdot x \oplus \{08\} \cdot x \tag{4.8}$$

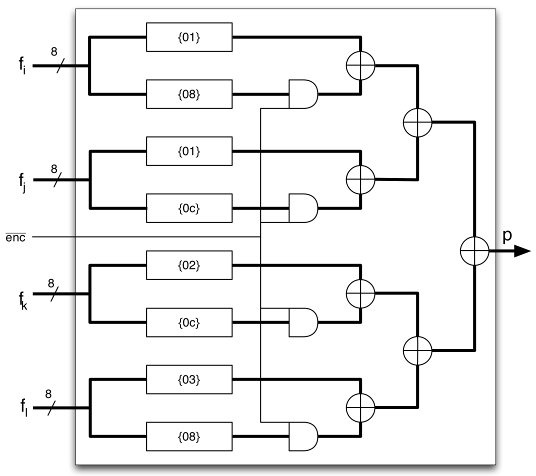


Figure 4.15: MixColumns polynomial multiplication.

4.5.5 Countermeasures against Power Analysis Attacks

The countermeasures against power analysis attacks are implemented mainly in the AES controlpath. Therefore, the original AES controlpath (see Figure 4.11) is altered. Furthermore, when dealing with shuffling and dummy operations, in this section the implementation of these concepts in decryption mode is not considered explicitly because it is performed the same way as for the encryption.

Shuffling

The method of shuffling is described in Section 3.3.2. According to the state diagram (see Figure 4.16) no additional states are necessary to implement shuffling.

Every sub-function of the AES algorithm is split up into four 32-bit executions which do not influence each other. The order of these executions is not relevant. The only exception is the key expansion, where shuffling is not possible. For the other sub-functions there are four possible orders:

- $0 \to 1 \to 2 \to 3$
- $1 \to 2 \to 3 \to 0$
- $2 \to 3 \to 0 \to 1$
- $3 \to 0 \to 1 \to 2$

These four possibilities can be encoded using two bits. Therefore a two-bit wide register is added into the AES controlpath which holds the value of the shuffling start pointer. The start pointer can be set using the AMBA interface. This happens at start-up with a random value.

For example, when the start pointer is set to 3 the execution order would be the following: $ADDR3 \to ADDR0 \to ADDR1 \to ADDR2 \to SBOX3 \to SBOX0 \to SBOX1 \to SBOX2 \to MIXC3$ ...

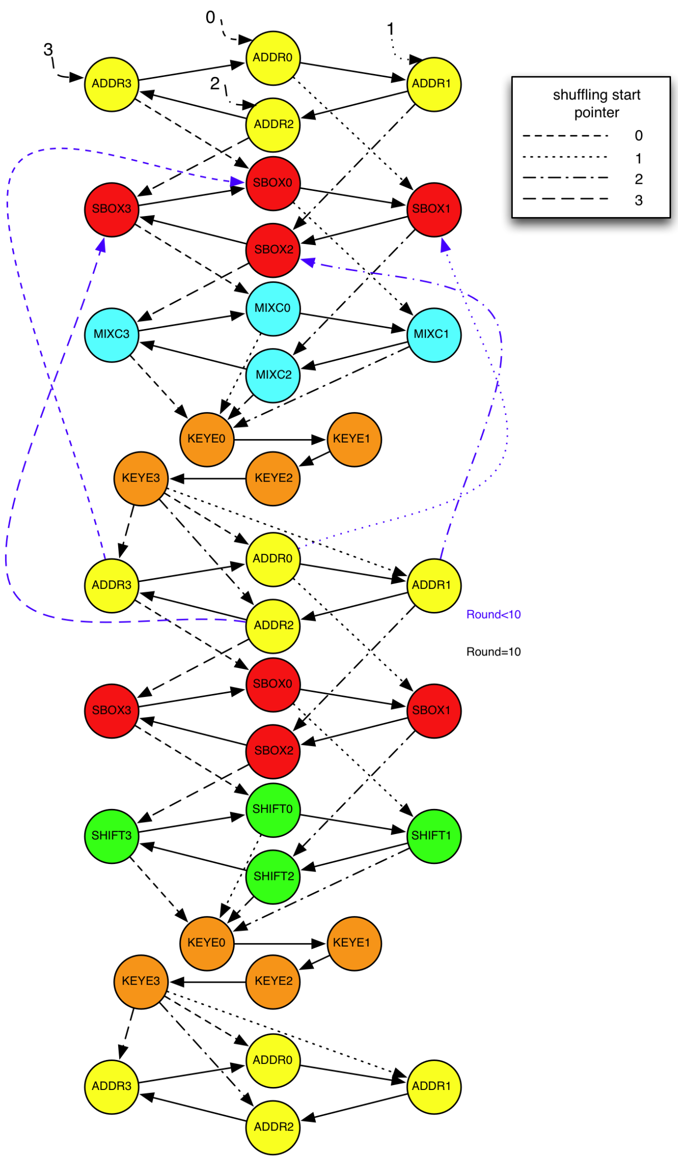


Figure 4.16: AES state diagram with shuffling.

Dummy Operations

Dummy operations (see Section 3.3.2) are inserted into the AES state diagram as shown in Figure 4.17. Before each group of four sub-functions a dummy state is inserted. The dummy state is repeated depending on a register in the AES datapath. This register is four bits wide and holds a random value which can be set using the AMBA interface. The random number N determines how often the dummy state is executed before the real processing is started. The range of N is $0 - 15$ in this implementation but can, in principle, be increased or decreased. This means the execution order is $1 + N$ times the dummy state followed by the group of four sub-functions and then $1 + (15 - N)$ times the dummy state again. After that the next dummy state follows for $1 + N$ times.

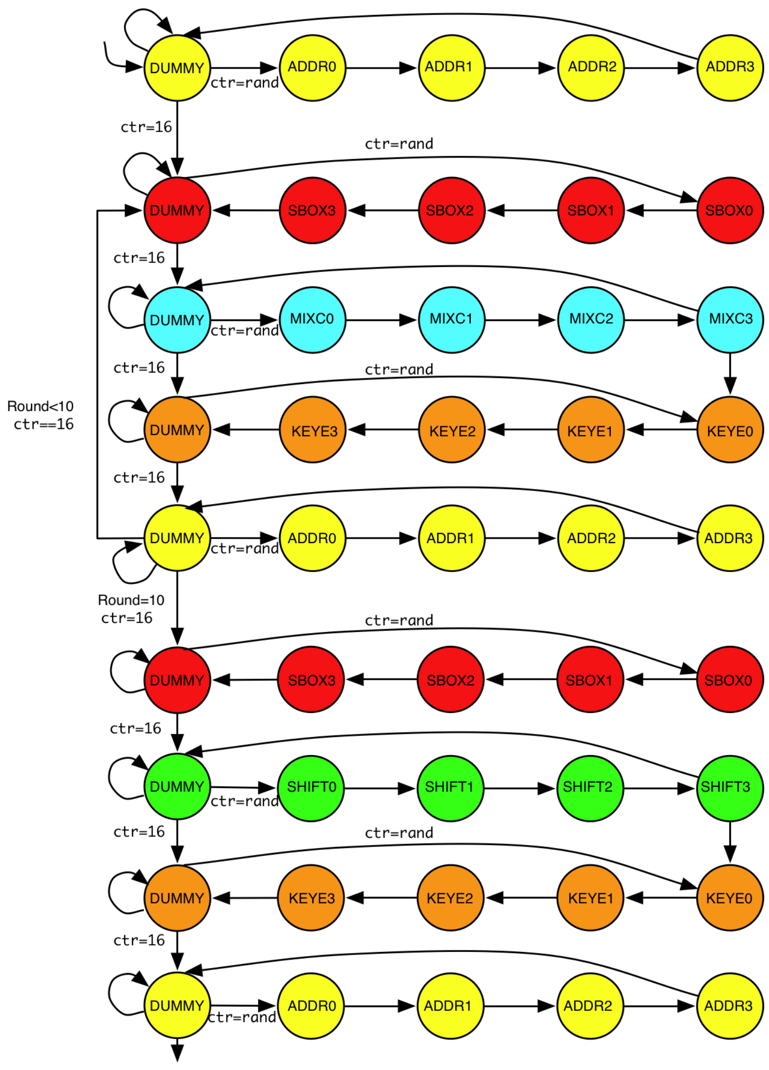


Figure 4.17: AES state diagram with dummy cycles.

Combining Shuffling and Dummy Operations

Shuffling and dummy operations can easily be implemented together. As mentioned before the dummy state is implemented before each group of four sub-functions. The order of the sub-functions is not only $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$, but also the three other possibilities described before are possible. The dummy operations are controlled by a 4-bit register, whereas the operation of the shuffling is determined by a 2-bit register. Both registers are set at start-up using the AMBA interface and random values.

4.5.6 AES Controlpath

In the AES controlpath the state diagram which combines shuffling and dummy operations is implemented. Since the structure of the AES encryption and decryption is similar, only one state diagram is needed. The differences are handled using if-statements. The implementation consists of two nested switch-case statements.

The outer statement implements the AES rounds starting from Round 0 with the four AddRoundKey operations. Round 1- Round 9 are the identical standard AES rounds and Round 10 is the last round of the AES where the MixColumns operations are substituted by the explicit ShiftRows operations. Round 11 is the idle state which is reached after the algorithm has finished. Round 12 is needed for the key expansion in decryption mode. The key expansion sub-functions must be executed ten times to receive the round key of the tenth round which is necessary before starting the decryption. A 4-bit wide register holds the value for the round.

The inner switch-case statement determines the state inside a round. This state changes every clock cycle unless the algorithm is finished and in the Idle state. The execution order of the AES sub-functions is controlled by the state. If-statements provide the necessary branches to implement the functionality of encryption and decryption in both ECB and CBC mode. A 6-bit wide register holds the value of the state. The right control signals to the datapath and the RAM are set in every state.

4.6 Grain

The stream cipher Grain (see Section 2.2.2) has to be implemented on the 32-bit datapath. The representation of the 160-bit state in the RAM is implemented as shown in Figure 4.18.

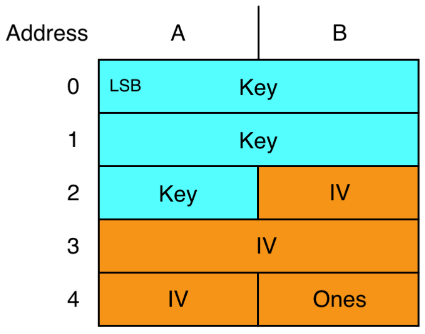


Figure 4.18: Representation of the Grain state at the beginning of the algorithm.

The blue color in the figure denotes the bits of the NFSR, whereas the orange color denotes the bits of the LFSR. The least significant bit is the leftmost bit on address 0. The drawback regarding the implementation of the Grain algorithm is that it is impossible to process 32-bit in parallel. The reason is the structure of the stream cipher (see Figure 2.1). The most significant bit used for the function $h(x)$ is s_{i+62}. Since the most significant bit is s_{i+79} this function can only process up to 18 bits in parallel. The $g(x)$ function allows 17 bits and the $h(x)$ function allows 16 bits. As a result the Grain algorithm is limited to a number of 16 bits processing in parallel.

4.6.1 Implementing the Bit-Shifting Functionality

The Grain algorithm consists of two shift registers which are connected in serial order. The implementation of the bit-shifting functionality which is necessary for this algorithm works with an address pointer to the least-significant bit of the NFSR. The first start-address is $0A$ followed by $0B$ (see Figure 4.19) and so on. The next address after $4B$ is $0A$ again. The 16 most-significant bits of the NFSR (denoted as $G(x)$) and LFSR (denoted as $F(x)$) are updated after every bit-shift operation. The other values remain the same.

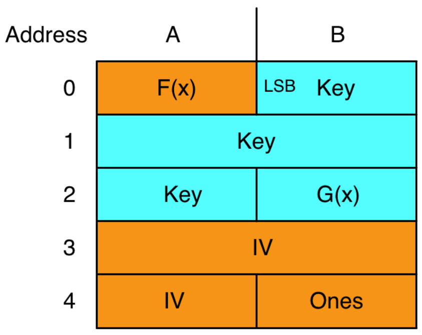


Figure 4.19: Representation of the Grain state after 16-bit left-shift.

4.6.2 Implementation of the Sub-Functions

The implementation of the update functions of the NFSR $G(x)$ and $H(x)$ require the use of an additional register. The reason for that is that these functions consist of XOR as well as AND gates. The AND gates are problematic. The interval which is needed to calculate $H(x)$ is b_{i+63}-s_{i+79} and $G(x)$ is b_{i+9}-s_{i+15}. The dual output memory only provides 64 bits. An additional 33-bit register is needed because for $H(x)$ 97 bits are needed simultaneously. The 33-bit register is split up into a 32-bit register and a one-bit register.

4.6.3 Grain Controlpath

The implementation of the Grain controlpath consists of two nested switch-case statements. The outer statement implements three states:

PRE-STATE: This state has no sub-states and is used to select either INIT or ROUND as the next state depending on the selected operation mode.

INIT: The initialization of the Grain algorithm is performed in this state. It consists of 13 sub-states. It is executed ten times to complete the initialization. The number of repetitions is controlled by a counter.

ROUND: The Grain algorithm is performed to receive 32 bits of the key stream. It is composed of 14 sub-states and must be executed two times.

The start-address can either be xA or xB, which leads to different control signals to the datapath and RAM. This problem is solved using If-statements.

4.7 Memory

Regarding cryptographic devices, a common approach is to place the needed registers in the datapath selected by multiplexers. As a consequence every cryptographic algorithm has its own tailored registers. To avoid this overhead there is the idea of reusing registers and organizing these registers in a RAM. The advantages are that the device needs less area and power.

In this specific approach the memory access is limited to one input port and two output ports. The RAM offers the functionality of doing one write and two read operations simultaneously. This interface has the benefit that the custom RAM module can be replaced by a RAM module of a manufacturer.

The basic component of this RAM module is a 32-bit register. There are two different approaches of implementing these registers, either flip-flop based (see Section 4.7.3) or latch-based (see Section 4.7.4).

4.7.1 RAM Size

The RAM consists of 22 data registers and is easily upgradeable. The size of 22 is determined by the requirements for the AES algorithm which are illustrated in Table 4.3 in detail. Here the AES algorithm operating in CBC decryption mode is shown. It has the largest memory requirements.

	Size [bit]	# registers
State	128	4
Key	128	4
Temporary state	128	4
IV	128	4
Initial state (only decryption)	128	4
Random data (for dummy cycles)	64	2
Sum	704	22

Table 4.3: AES memory requirements.

Figure 4.20 demonstrates that 22 registers are the minimum number needed for the AES CBC decryption algorithm. The ciphertext, which is denoted as initial state in the table, has to be stored because it is needed for the decryption of the next 128-bit block. In addition to the key, four registers are also needed to hold the state and the temporary state. Apart from this, four registers are also needed for holding the IV. It is impossible to use the same four registers for the temporary state and the IV, because the IV is used at the end of the decryption and the temporary state is needed prior to this. This would be possible if the order could be reversed.

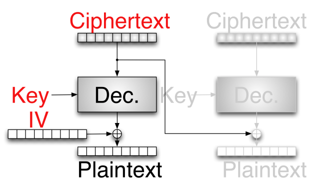


Figure 4.20: AES CBC decryption mode.

4.7.2 Architecture

Figure 4.21 shows the architecture of the RAM.

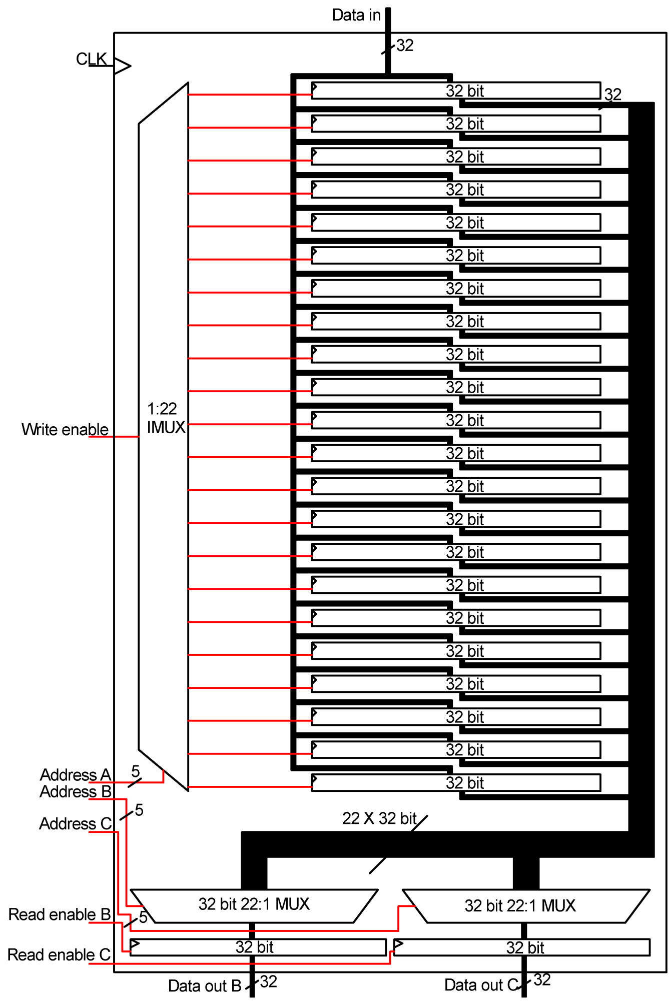


Figure 4.21: Dual output port RAM.

Write Access

There is a 32-bit data input which is also the input of every register. The write enable signal is used as an input for the inverse multiplexer. Depending on the address the write signal is wired to the enable input port of the appropriate register. As a consequence a register only stores the data input value when the write enable signal is high and the address is set to this register.

Read Access

Every register has a 32-bit output. These 22 32-bit outputs form a bus which is the input of two independent multiplexers. By setting the address the associated data register value is stored in the output register, provided that the read enable signal is set. The other output port works the same way.

4.7.3 Flip-Flop Based Register

The basic component of the register shown in Figure 4.22 is a flip-flop. (For further details concerning flip-flops see Section 3.4.2). 32 flip-flops are connected in parallel. Every flip-flop has a common clock wire and shares the same enable wire. The input wires of the flip-flops are combined to a 32-bit input as well as the output wires form a 32-bit output.

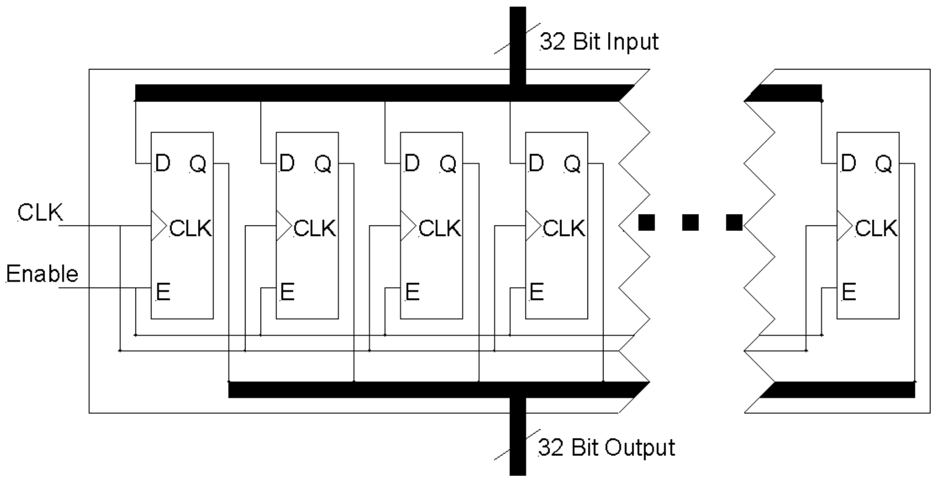


Figure 4.22: 32-bit flip-flop based register.

4.7.4 Latch-Based Register

The basic component of the register shown in Figure 4.23 is a latch. (For further details concerning latches see Section 3.4.2). 32 latches are connected in parallel. Every latch shares the same enable wire. The input wires of the latches are combined to a 32-bit input as well as the output wires form a 32-bit output. An additional latch is needed to provide the enable signal for the 32 data latches only when the clock signal is high.

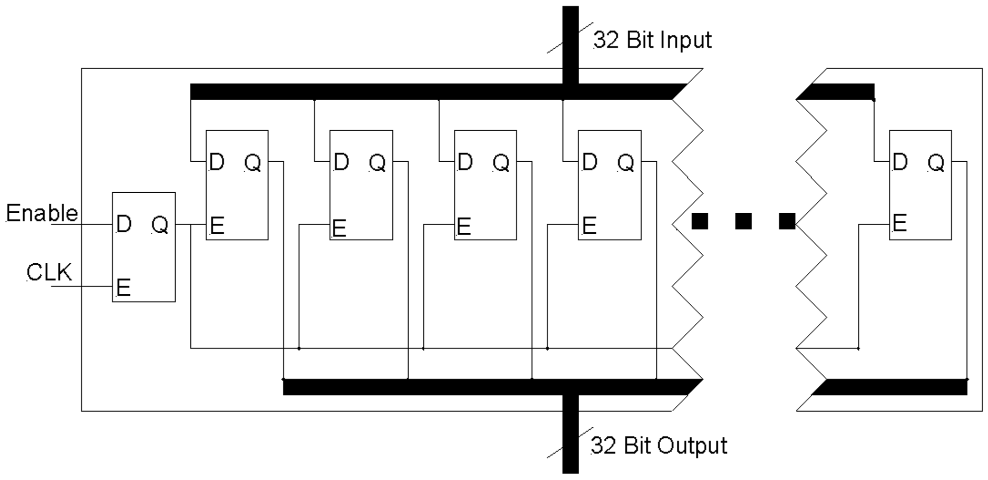


Figure 4.23: 32-bit latch-based register.

Chapter 5

Results

This chapter deals with the results of the synthesized crypto module. Section 5.1 analyzes the area consumption, Section 5.2 focuses on the power consumption and Section 5.3 concentrates on timing issues of the crypto module.

The data used for the workflow is a clock frequency of 1 MHz and a supplying voltage of 3.3 V, unless other values are explicitly mentioned. The target technology is the 0.35 μm-CMOS process from austriamicrosystems which uses digital standard cells.

The functional verification is done with the data of the high-level Java model. The correct functionality is ensured after simulating the behavioral, synthesized and post-layout model.

Figure 5.1 shows the layout of the crypto module after place and route with the estimated location of the modules. The tool used is Encounter from Cadence.

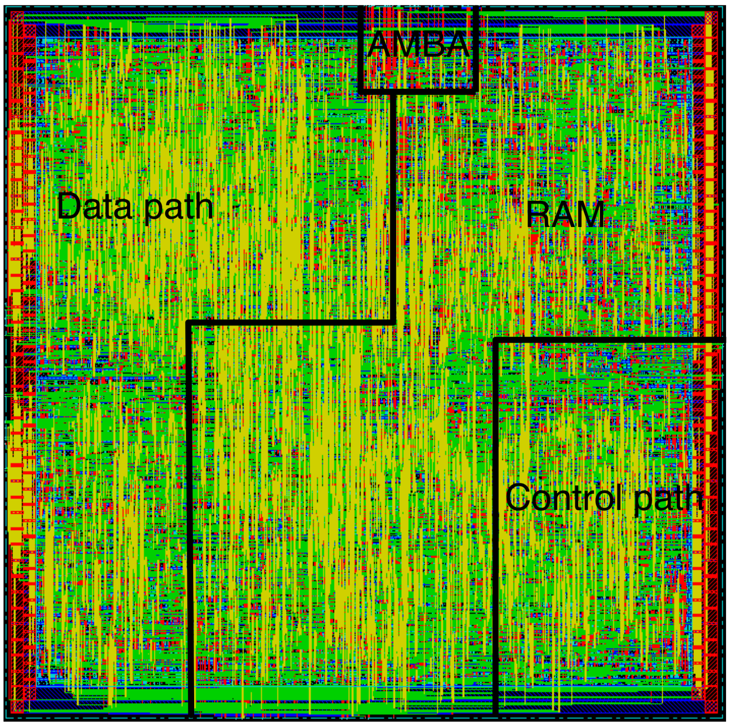


Figure 5.1: Layout of the crypto module after place and route with the estimated location of the modules.

5.1 Area

The area consumption after the place and route process varies because it significantly depends on the used target technology and settings. That is the reason the values for the area consumption of each of the modules are taken after the synthesis step. In addition to calculate the area of the top modules, the synthesizer also determines the size of all submodules independent of their hierarchical level.

The module size is expressed in μm^2. This value is completely influenced by the used target technology. Gate equivalents (GE) are used to correct the module size in a way that it can also be compared to other target technologies. It is a basic unit for describing relative digital circuit complexity. The number of gate equivalents of a circuit is the number of individual logic gates that would have to be interconnected to perform the same function [19]. To receive the number of gate equivalents the area of the circuit is divided by the area of a single NAND gate with two inputs of drive strength one of the used target technology. The 0.35 μm austriamicrosystems technology NAND gate requires 55 μm^2.

The maximum clock frequency used for the synthesis was 1 MHz. Table 5.1 reflects the results of the top-level modules.

Module	Cell area		
	$[\mu m^2]$	$[GE]$	[%]
Datapath	291873.40	5307	36.34
Controlpath	156611.00	2846	19.50
22x32-bit RAM	309418.20	5626	38.53
AMBA interface	28009.80	510	3.49
Glue logic	17180.80	312	2.14
Total area	803093.20	14601	100.00

Table 5.1: Cell area of the top-level modules @ 1 MHz.

When analyzing the area consumption of the top level modules, there is no surprise. As expected, the largest module is the RAM module (38.53 %). This number is the result of using a latch-based RAM architecture. For a comparison of the results of flip-flop based RAM versus latch-based RAM see Section 5.1.4. The second largest module is the datapath (36.34 %) followed by the controlpath (19.50 %). The sizes of the AMBA interface (3.49 %) and the glue logic (2.14 %) are comparatively small. Diagram 5.2 illustrates the distribution of the area of the top-level modules at a glance.

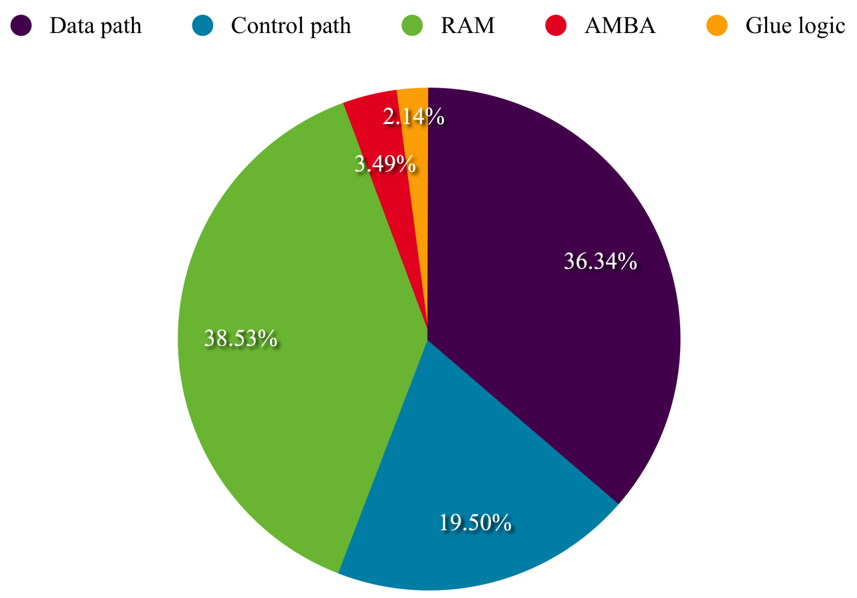


Figure 5.2: Distribution of the area of the top-level modules.

The following subsections deal with the area consumption of the AES (see Section 5.1.1) and the Grain (see Section 5.1.3) algorithms.

5.1.1 Cell Area of AES

Table 5.2 reflects the fact that the datapath of the AES algorithm is almost three times the size of the controlpath. Considering that the countermeasures against power analysis attacks are included in the controlpath, the bare controlpath is actually even smaller (see Section 5.1.2).

Module	Cell area		
	$[\mu m^2]$	$[GE]$	[%]
AES controlpath	51451.40	935	26.24
AES datapath	144635.40	2630	73.76
AES	196086.80	3565	100.00

Table 5.2: Cell area of AES.

The datapath of the AES algorithm is analyzed in Table 5.3. The largest parts of the datapath are the S-box (42.65 %) and the MixColumns function (39.46 %). Due to its 32-bit wide XOR function, the AddRoundkey submodule is as expected, very small.

The 15.07 % labeled as "Other" include the multiplexers needed to select the right inputs as well as outputs of the submodules in addition to the functionality needed for the key expansion.

Module	Cell area		
	$[\mu m^2]$	$[GE]$	[%]
AddRoundkey	4076.80	74	2.82
MixColumns	57075.20	1038	39.46
S-box	61679.80	1121	42.65
Other	21803.6	396	15.07
AES datapath	144580.80	2629	100.00

Table 5.3: Cell area of AES datapath.

5.1.2 Countermeasures against Power Analysis Attacks

The additional gates, which are needed to implement the countermeasures against power analysis attacks are placed mainly in the AES controlpath. Table 5.4 shows that the countermeasures need 27.35 % of the controlpath. This number can be split up to 17.76 % for dummy cycles and 9.59 % for shuffling.

Module	Cell area		
	$[\mu m^2]$	$[GE]$	[%]
AES controlpath	37382.80	679.69	72.66
Shuffling	4932.20	89.68	9.59
Dummy cycles	9136.40	166.12	17.76
Total area	51451.40	935.48	100.00

Table 5.4: Cell area of the AES controlpath.

5.1.3 Cell Area of Grain

According to Table 5.5, the datapath of the Grain algorithm is 58.85 % and the controlpath is 41.15 % of the whole area consumption of 4434 GE, which is remarkably high considering that the whole AES algorithm is only 3565 GE. The large size of the implementation presented in this work is caused by the 16-bit parallel connected architecture of the logic functions and the complexity of the controlpath described in Section 4.6.3.

Module	Cell area		
	$[\mu m^2]$	$[GE]$	[%]
Grain controlpath	100354.80	1825	41.15
Grain datapath	143525.20	2610	58.85
Grain	243880.00	4435	100.00

Table 5.5: Cell area of Grain.

Table 5.6 illustrates the area allocation of the sub functions. The biggest part is denoted as "Other" and consists of the multiplexers to select the desired input bits and output bits as well as the update functions for the NFSR and LFSR.

Module	Cell area $[\mu m^2]$	$[GE]$	[%]
F(x)	7571.20	138	5.28
G(x)	47174.40	858	32.87
H(x)	22131.20	401	15.42
Z(x)	8736.00	159	6.09
Other	57912.40	1053	40.35
Grain datapath	143525.20	2609	100.00

Table 5.6: Cell area of Grain datapath.

5.1.4 Flip-Flop Based RAM vs. Latch-Based RAM

Table 5.7 emphasizes the idea of finding the best architecture for the 22x32-bit RAM. The first approach of using flip-flop based RAM uses 6586 GE. That can be reduced by 14.58 % to 5626 GE when using the latch-based approach. 132 GE are necessary to store 32 bits considering the latch-based approach whereas, 191 GE are needed for the flip-flop based approach. The expectation has been that the latch-based approach would save nearly half of the area consumption because 33 latches are needed in comparison to 32 D-flip-flops which consist of 64 latches. The prospects are not fulfilled because:

1. The area consumption of a D-flip-flop is optimized. (The size of a D-flip-flop is 1.8 times the area consumption of a latch.)
2. Additional gates are used for the latch-based approach (symmetrical buffers for clock tree synthesis and symmetrical inverters for clock tree synthesis).

Module	Cell area $[\mu m^2]$	$[GE]$	[%]
Flip-flop based RAM	362216.41	6586	100.00
Latch-based RAM	309418.20	5626	85.42
Difference	52798.21	960	14.58

Table 5.7: Comparison of flip-flop based RAM vs. latch-based RAM.

5.2 Power Consumption

The CMOS power consumption consists of four different components which contribute to it [58]:

$$P_{total} = P_{dyn} + P_{stat} + P_{short} + P_{leak}$$

P_{dyn}**:** Dynamic power consumption is the result of charging and discharging nodes. The formula for dynamic power consumption is $P = C_L \cdot V_{DD}^2 \cdot f \cdot p_{switch}$ where C_L is the load capacitance, V_{DD} is the supply voltage, f is the clock frequency and p_{switch} is the switching activity.

P_{stat}**:** Static power consumption is caused by static currents.

P_{short}**:** Short-circuit dissipation is the result of short-circuit currents between supply and ground during transients.

P_{leak}**:** Leakage power consumption is caused by substrate currents and sub-threshold currents.

The dynamic power consumption dominates the power consumption and can easily be decreased by altering the frequency and the supply voltage, whereas decreasing the voltage influences the power consumption quadratical. That is the reason the supply voltage should be lowered, if possible.

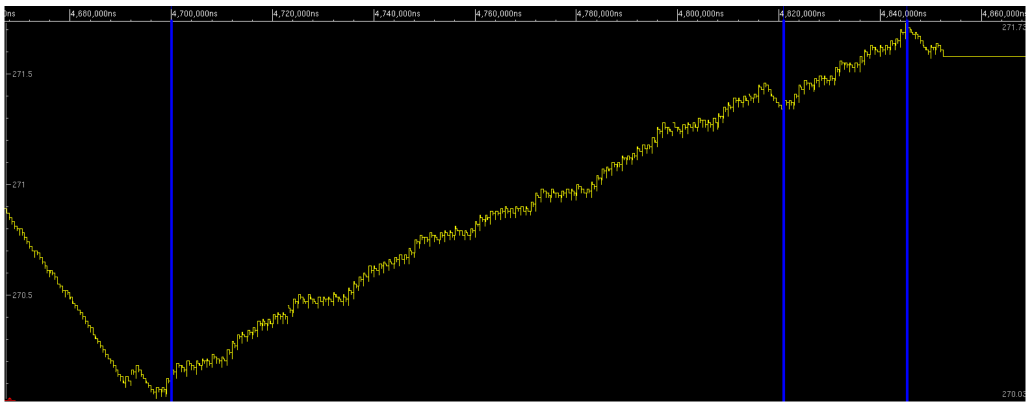
Synopsys NanoSim is used to do the power analysis of the circuit. The results are more accurate than the power analysis of the synthesis tool because it is a transistor-level circuit simulator. In addition to a *Spectre* netlist the simulator needs a VCD file to do the simulation. The VCD file is the stimulus file which is created by the behavioral model and contains a series of time ordered value changes for the signals for a given simulation run. Therefore, it is possible to analyze the power consumption of the circuit while it is calculating a real Grain or AES algorithm. The drawback of using *NanoSim* is that the analysis is computationally intensive and therefore very time consuming.

The power consumption is calculated using the formula $P = U \cdot I$ where U is the supply voltage V_{DD} of $3.3\,V$. I is substituted by the average current which is calculated by *NanoSim*. Table 5.8 shows the power consumption of different operating modes.

Operation	Clock frequency $[MHz]$	Voltage $[V]$	Average current $[\mu A]$	Power $[\mu W]$
Grain	1.00	3.3	288	950
AES ECB enc.	1.00	3.3	259	855
AES ECB dec.	1.00	3.3	279	921
AES CBC enc.	1.00	3.3	251	828
AES CBC dec.	1.00	3.3	260	858

Table 5.8: Power consumption of different operation modes.

Figure 5.3 shows the supply current trace of the Grain algorithm. The core voltage is $3.3\,V$ and the clock frequency is $1\,MHz$. The simulation time $[ns]$ is plotted on the x-axis while the supply current $[\mu A]$ is plotted on the y-axis. The first blue marker at $4700000\,ns$ represents the start of the Grain initialization. The second marker at $4821000\,ns$ displays the end of the Grain initialization and the start of the first Grain execution to receive 32 bits of the key stream. The third blue marker at $4845000\,ns$ indicates the end of the Grain algorithm.



,

Figure 5.3: Supply current trace of the Grain algorithm.

The supply current trace of the AES ECB encryption with dummy cycles is presented in Figure 5.4. The simulation parameters have remained the same. The first blue marker at $3780000\,ns$ shows the start of the first round of the AES in ECB encryption mode. The other blue markers also indicate the starting points of the other individual rounds of the algorithm. One round takes about $85000\,ns$. The last round ends at $4640000\,ns$.

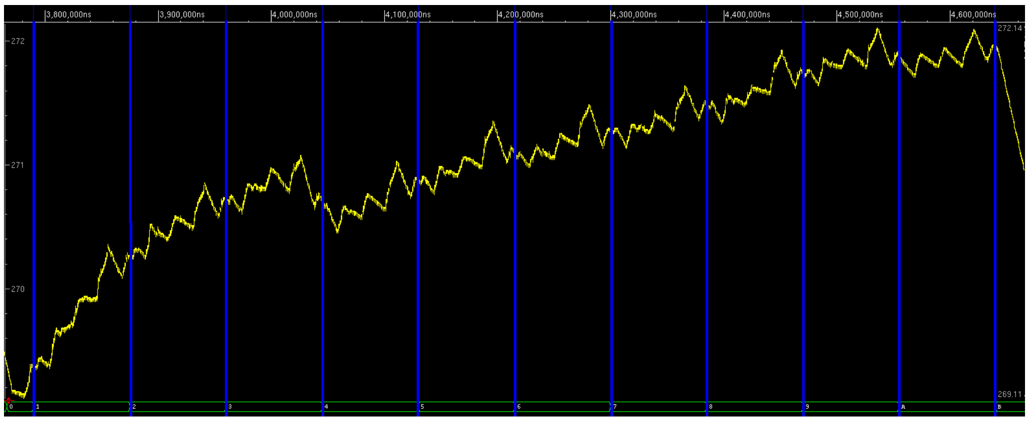


Figure 5.4: Supply current trace of the AES encryption.

5.3 Timing Analysis

This section deals with the timing analysis of the crypto module. First, the cycle count of the different modes is presented followed by the critical path which determines the maximum clock frequency.

5.3.1 Number of Cycles

The number of cycles of the several operating modes is measured from the starting time of the algorithm to the time the result is written to the RAM. Table 5.9 shows the results of the AES algorithm.

Algorithm	Cycles
AES ECB encrypt	164
AES ECB encrypt with dummy cycles	871
AES ECB decrypt	240
AES ECB decrypt with dummy cycles	1101
AES CBC encrypt	169
AES CBC encrypt with dummy cycles	875
AES CBC decrypt	244
AES CBC decrypt with dummy cycles	1105

Table 5.9: Cycle count of AES.

The number of dummy cycles is 17 for each of the subfunctions. Adding the 4 cycles where the real data is processed 21 cycles are needed. The predictions of the high-level model are very similar to the results of the AES algorithm. Table 5.10 presents the timing results of the Grain algorithm.

Algorithm	Cycles
Grain init	121
Grain 32 bits	26

Table 5.10: Cycle count of Grain.

5.3.2 Maximal Clock Frequency

The maximal clock frequency is determined by the critical path of the circuit. Figure 5.5 shows the critical path which is located in the AES datapath. It has a delay of $53.57\,ns$ and is the implementation of the formula $SubWord(RotWord(temp))\,xor\,Rcon[i/Nk]$ which is used for the key expansion (see Section 2.2.5). The delay of $53.57\,ns$ results in a maximum clock frequency of $18.66\,MHz$. A worst case scenario was taken to receive the maximum clock frequency. Therefore, the frequency could be increased beyond $18.66\,MHz$ under normal operating conditions.

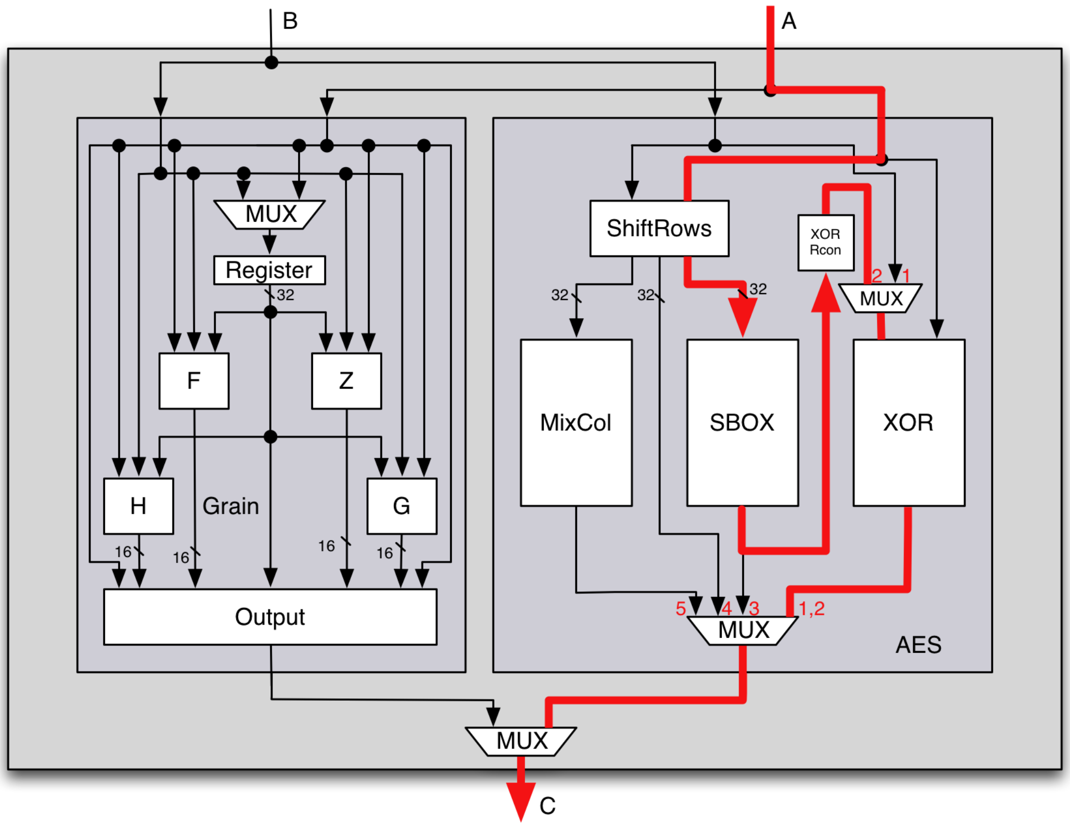


Figure 5.5: Critical path.

5.4 Comparison to Related Work

This section presents a comparison between the results of this work and others. Since the crypto module is unique no comparisons of the whole module are possible and consequently, only parts can be analyzed.

There are various works about the hardware implementation of the AES algorithm but only a few fulfill the requirements which are needed for the comparison:

- 32-bit datapath
- Standard-cell implementation (not FPGA)
- Compact size and low-power consumption

Hua Li et al. [35] propose a dual-core architecture for AES which can process encryption and decryption simultaneously with on-the-fly key expansion and is most suitable for applications such as real-time dual-duplex wireless communication.

A highly scalable architecture is presented by Pramstaller et al. [46] and can be used for a wide range of applications. There are three different versions of the architecture (high performance, standard, minimum), where the version with minimum area is most interesting in relation to this work.

Satoh et al. [50] implement a very compact high-speed architecture for AES. Encryption and decryption datapaths are efficiently combined.

Table 5.11 shows a comparison of related AES implementations.

Work	Datapath $[bit]$	Technology $[\mu m]$	Area $[GE]$	Runtime $[cycles]$	f_{max} $[MHz]$
Hua Li [35]	32	0.35	16629	44	115
Pramstaller [46]	32	0.60	6675	92	50
Satoh [50]	32	0.11	5398	54	131
This work	32	0.35	8935	164	19

Table 5.11: Comparison of related AES implementations.

The results used for this work do not include the countermeasures against power analysis attacks. The largest part of the AES implementation is the RAM. The area consumption is without the RAM only 3309 GE. However, this fact underlines the key idea of the crypto module, which is to use a shared RAM for different cryptographic algorithms.

This work has the maximum results for the cycle count for the AES encryption and the maximum clock frequency. Since the algorithm has to be implemented on contactless smartcards the main target is to reduce area and power consumption.

The problem of the Grain is that there are only very few published works. Feldhofer [13] proposes a 16-bit low-power implementation for the algorithm.

Table 5.12 shows a comparison of related Grain implementations. The number of clock cycles which is required for the two works is nearly the same, only the number for the initialization varies slightly. The area consumption of this work is quite large when keeping in mind that it is missing the RAM. The average current of this work is about 28 μA@0.1 MHz, which is about 35 times the current of the other approach.

Work	Datapath $[bit]$	Technology $[\mu m]$	Area $[GE]$	Runtime $[cycles]$	I_{avg} $[\mu A]$
Feldhofer [13]	16	0.35	3360	(130)+26	0.80@0.1 MHz
This work	32	0.35	4434	(121)+26	288@1 MHZ

Table 5.12: Comparison of related Grain implementations.

Chapter 6

Conclusions

This work describes the architecture and the implementation of a low-power crypto module. It should be possible to use the crypto module on a contactless smartcard. Therefore, the main goals are to reduce power and area consumption.

The crypto module offers the functionality of the AES and the Grain algorithm. Due to the modular architecture, it is easily upgradeable with other cryptographic algorithms. It is composed of datapath, controlpath, RAM and AMBA interface. The datapath is 32 bits wide and consists of two submodules, the AES and Grain datapath. The desired datapath is selected with multiplexers. The same principle applies to the controlpath, which also consists of AES and Grain controlpath.

The RAM used in this work has two output ports and one input port that are each 32 bits wide. It is used as common RAM for all implemented algorithms. It consists of 22 32-bit registers but it is possible to increase its size. Two different approaches are analyzed to implement the RAM (flip-flop based vs. latch-based). The latch-based approach results in smaller area and power consumption.

A 32-bit APB AMBA is used as the interface for the crypto module. The RAM can be accessed through the interface as well as some registers for selecting the operation mode.

The AES module provides encryption and decryption in ECB and CBC operating mode. Only 128-bit keys are supported and the key expansion is calculated on the fly. In every clock cycle 32 bits are processed. As a consequence every sub function must be executed four times to process the whole 128-bit AES state. The ShiftRows sub function does not need to be executed explicitly because it is done implicitly when the SubBytes and MixColumns operations are performed. The implementation of dummy cycles and shuffling make it more difficult for attackers to retrieve the secret key through power analysis attacks.

The Grain algorithm offers poor support for a 32-bit implementation. The maximum size is 16 bits. Another drawback is due to the fact that the state of the Grain is stored in the RAM, not all bits are accessible at the same time. This would be necessary to do a reasonable implementation.

The crypto module is implemented in $0.35\,\mu m$ AMS standard cell technology. The area consumption is $14601\,GE$ and the maximum clock frequency is $19\,MHz$. The crypto module requires an average current of $259\,\mu A$ when operating in AES ECB encryption mode at a clock frequency of $1\,MHz$.

The results of the crypto module are very good except the drawback of the implementation of the Grain. It is very cumbersome and it is even bigger than the AES. Finding another algorithm which can generate pseudo-random numbers and supports a

32-bit implementation is encouraged. The AES algorithm could also be used to generate pseudo-random numbers. If desired it should be possible to turn off the dummy cycles to shorten the execution time.

The next step would be to upgrade the crypto module with the ECC and SHA algorithm to provide the functionality of ECDSA. The RAM would also be used in these additional algorithms to save area and energy. This would result in a crypto module capable of both symmetric and asymmetric cryptography qualified for the use on contactless smartcards.

Appendix A

Definitions

A.1 Abbreviations

AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
ANSI	American National Standards Institute
APB	Advanced Peripheral Bus
APDU	Application Data Unit
ARM	Advanced RISC Machine
ASK	Amplitude Shift Keying
ASB	Advanced System Bus
ATS	Answer To Select
BPSK	Binary Phase Shift Keying
CA	Certification Authority
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CLB	Configurable Logic Block
CMOS	Complementary MetalOxideSemiconductor
CPU	Central Processing Unit
CTR	Counter
DES	Data Encryption Standard
DPA	Differential Power Analysis
DRAM	Dynamic Random Access Memory
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Algorithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EM	Electromagnetic
FPGA	Field Programmable Gate Array
GE	Gate Equivalent
GF	Galois Field
GSM	Global System for Mobile
HDL	Hardware Description Language
IAIK	Institute for Applied Information Processing and Communications

IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IPsec	Internet Protocol security
IP	Intellectual Property
ISO	International Organization for Standardization
IV	Initialization Vector
LFSR	Linear Feedback Shift Register
LVS	Layout Versus Schematic
MIPS	Microprocessor without Interlocked Pipeline Stages
NAND	Not AND
NFSR	Nonlinear Feedback Shift Register
NIST	National Institute of Standards and Technology
NRZ	Non Return to Zero
NSA	National Security Agency
CPU	Central Processing Unit
OFB	Output Feedback
PGP	Pretty Good Privacy
PIN	Personal Identification Number
PPS	Protocol and Parameter Selection
PRNG	Pseudo Random Number Generator
PSE	Personal Security Environment
RAM	Random Access Memory
RATS	Request for Answer To Select
RC4	Rivest Cipher 4
RFID	Radio Frequency Identification
RMS	Root Mean Square
RNG	Random Number Generator
RSA	Rivest, Shamir, Adleman
RTL	Register Transfer Level
SHA	Secure Hash Algorithm
S/MIME	Secure/Multipurpose Internet Mail Extensions
SPA	Simple Power Analysis
SRAM	Static Random Access Memory
SSH	Secure Shell
SSL	Secure Sockets Layer
TCL	Tool Command Language
TLS	Transport Layer Security
TRNG	True Random Number Generator
VHDL	VHSIC hardware description language
VLSI	Very-Large-Scale Integration
XOR	Exclusive OR

Bibliography

[1] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. The EM Side-Channel(s). Lecture notes in computer science, pages 29–45, 2003.

[2] S. Ahson and M. Ilyas, editors. RFID Handbook Applications, Technology, Security, and Privacy. CRC Press, 2008.

[3] ARM Limited. AMBA Specification, 1999.

[4] E. Barker and J. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, NIST, March 2007.

[5] H. Bidgoli. Handbook of Information Security. Wiley, 2006.

[6] J. Buchmann. Einführung in die Kryptographie. Springer, Berlin, 2008.

[7] D. Canright. A Very Compact S-Box for AES. Cryptographic Hardware and Embedded Systems – CHES 2005, 2005.

[8] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, pages 398–412. Springer-Verlag, London, UK, 1999.

[9] C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems, pages 252–263. Springer-Verlag, London, UK, 2000.

[10] M. Cochran. Notes on the Wang et al. 2^{63} SHA-1 Differential Path. Cryptology ePrint Archive, Report 2007/474, 2007.

[11] J. Daemen and V. Rijmen. The Block Cipher Rijndael. In CARDIS '98: Proceedings of the The International Conference on Smart Card Research and Applications, pages 277–284. Springer-Verlag, London, UK, 2000.

[12] J. Daemen and V. Rijmen. The Design of Rijndael. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[13] M. Feldhofer. Comparison of Low-Power Implementations of Trivium and Grain, 2007. Presentation.

[14] K. Finkenzeller. RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[15] FIPS. Advanced Encryption Standard (AES). NIST, 2001.

[16] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In RC4", Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography, pages 1–24. 2001.

[17] T. E. Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Proceedings of CRYPTO 84 on Advances in cryptology, pages 10–18. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[18] S. Garfinkel and B. Rosenberg. RFID : Applications, Security, and Privacy. Addison-Wesley, 2006.

[19] R. Graf. Modern Dictionary of Electronics. Newnes, 1999.

[20] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp. Cryptanalysis with COPACOBANA. Computers, IEEE Transactions on, 57(11):1498–1513, Nov. 2008.

[21] G. Hancke. A Practical Relay Attack on ISO 14443 Proximity Cards. Technical report, University of Cambridge, 2005.

[22] D. Hankerson, A. Menezes, and S. Vanstone. Guide to Elliptic Curve Cryptography. Springer, New York, 2004.

[23] M. Hell, T. Johansson, and W. Meier. Grain - A Stream Cipher for Constrained Environments. Int. J. Wire. Mob. Comput., 2(1):86–93, 2007.

[24] M. Hendry. Smart Card Security and Applications, Second Edition. Artech House, Inc., Norwood, MA, USA, 2001.

[25] Smart Card Sales to Reach 3 Billion by 2008, 2004.

[26] E. Hwang. Digital Logic and Microprocessor Design with VHDL. CL-Engineering, 2005.

[27] International Organization for Standardization. ISO-14443: Identification Cards – Contactless Integrated Circuit Cards – Proximity Cards, 2000.

[28] D. Johnson and A. Menezes. The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, Certicom Research, 1999.

[29] S. Kang and Y. Leblebici. CMOS Digital Integrated Circuits Analysis & Design. McGraw-Hill Science/Engineering/Math, 2002.

[30] Z. Kfir and A. Wool. Picking virtual pockets using relay attacks on contactless smartcard systems, 2005.

[31] A. Klein. Visuelle Kryptographie. Springer, 2007.

[32] N. Koblitz. Elliptic Curve Cryptosystems. Mathematics of Computation, 48(177):203–209, 1987.

[33] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In Advances in Cryptology — CRYPTO' 99, pages 388–397. Springer-Verlag, 1999.

[34] G. Langholz, J. L. Mott, and A. Kandel. Foundations of Digital Logic Design. World Scientific Publishing Company, 1998.

[35] H. Li and J. Li. A New Compact Dual-Core Architecture for AES Encryption and Decryption. Canadian Journal of Electrical and Computer Engineering, 33(3):209–213, 2008.

[36] Y. Lu, W. Meier, and S. Vaudenay. The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. In Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, volume 3621 of *Lecture Notes in Computer Science*, pages 97–117. 2005.

[37] S. Mangard, E. Oswald, and T. Popp. Power Analysis Attacks- Revealing the Secrets of Smart Card. Springer, 2007.

[38] K. Mayes and K. Markantonakis. Smart Cards, Tokens, Security and Applications. Springer, 2008.

[39] A. Menezes, P. Van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC press, 1997.

[40] V. S. Miller. Use of Elliptic Curves in Cryptography. In CRYPTO '85: Advances in Cryptology, pages 417–426. Springer-Verlag, London, UK, 1986.

[41] Y. Naito, K. Ohta, and N. Kunihiro. Improved Collision Search for Hash Functions: New Advanced Message Modification. IEICE Trans. Fundam. Electron. Commun. Comput. Sci., E91-A(1):46–54, 2008.

[42] NIST. Recommendation for Block Cipher Modes of Operation - Methods and Techniques, Dec 2001.

[43] NIST. FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2. Technical report, Department of Commerce, August 2002.

[44] V. Oklobdzija. Digital system clocking: high-performance and low-power aspects. Wiley-IEEE Press, 2003.

[45] A. Pavlov and M. Sachdev. CMOS SRAM Circuit Design and Parametric Test in Nano-Scaled Technologies: Process-Aware SRAM Design and Test. Springer Publishing Company, Incorporated, 2008.

[46] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer. Efficient AES implementations on ASICs and FPGAs. Lecture Notes in Computer Science, 3373:98, 2005.

[47] W. Rankl and K. Cox. Smart Card Applications: Design Models for Using and Programming Smart Cards. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[48] RNCOS. Smart Card Market Forecast to 2012. Industry Research Report, 2008.

[49] K. Roy and S. Prasad. Low-Power CMOS VLSI Circuit Design. Wiley India Pvt. Ltd., 2009.

[50] A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-box Optimization. Lecture notes in computer science, pages 239–254, 2002.

[51] B. Schneier. Applied Cryptography. Wiley New York, 1996.

[52] C. Shannon. Communication Theory of Secrecy Systems. Bell Systems Techn. Journal, 28:656–715, 1949.

[53] W. Stallings. Network Security Essentials: Applications and Standards. Prentice Hall, 2007.

[54] J. Swoboda, S. Spitz, and M. Pramateftakis. Kryptographie und IT-Sicherheit. Vieweg+Teubner Verlag, 2008.

[55] D. Thomas and P. Moorby. The Verilog Hardware Description Language. Springer, 2008.

[56] S. Tillich. Instruction Set Extensions for Support of Cryptography on Embedded Systems. Ph.D. thesis, Graz University of Technology, 2008.

[57] H. van Tilborg. Encyclopedia of Cryptography and Security. Springer, 2005.

[58] H. Veendrick. Deep-Submicron CMOS ICs: From Basics to ASICs. Kluwer academic publishers, 2000.

[59] J. Viega. Practical Random Number Generation in Software. In ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference, page 129. IEEE Computer Society, Washington, DC, USA, 2003.

[60] N. Weste and D. Money. CMOS VLSI Design. Pearson/Addison Wesley, 2005.

[61] W. Wolf. Modern VLSI design: System-on-Chip Design. Prentice Hall, 2002.

[62] J. Wolkerstorfer. An ASIC Implementation of the AES-MixColumn Operation. In Proceedings of Austrochip 2001, pages 129 – 132. 2001.