Exploring the Design Space of the GPS Authentication Scheme

Georg Hofferek, Bakk.techn.
georg@hofferek.at

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

Master's Thesis

Supervisor: Dipl.-Ing. Dr.techn. Johannes Wolkerstorfer

January, 2008

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 dieser Arbeit, die auf Arbeiten anderer Autoren beruhen, sind durch Angabe der entsprechenden Referenz gekennzeichnet.

Georg Hofferek, Bakk.techn.

Acknowledgements

Numerous people deserve to be thanked for their support during the creation of this thesis. In particular, I would like to thank Johannes Wolkerstorfer, for being an excellent supervisor. He always took the time to discuss problems and difficulties and he provided valuable feedback on all my ideas and thoughts.

I would also like to thank Martin Feldhofer, for helping me with tool-related problems and providing his expertise whenever needed.

Moreover I express my gratitude to my parents and family, for giving me the opportunity to attend the university in the first place, and for providing vital moral support.

My friends and colleagues, who persistently encouraged me whenever I needed reassurance, should also be mentioned. They additionally helped me to relax, have fun, and distract my mind from temporary setbacks.

Another person I am grateful to is Gerhard Haas, my high school physics teacher. He fostered and increased my interest in science and therefore greatly contributed to my decision to study Telematics.

Last but not least, I sincerely thank Marc Girault, for taking the time to email me detailed answers to all the questions I asked him.

Georg Hofferek January, 2008

Abstract

GPS is a public-key zero-knowledge protocol, which provides unilateral authentication and offers a rather large design space of possible implementation variants compared to other authentication schemes. Due to its flexibility it has been suggested for use in passive RFID tags, challenging the doctrine that public-key cryptography is too complex for passive devices. In its smallest configuration, GPS allows to use a coupon-based approach where only simple integer operations need to be computed on the prover's side during authentication. More complex forms which require computation of number-theoretic operations, hashes, and random numbers form the opposite border of the design space. This work presents an approach where complex number-theoretic operations are (slowly) precomputed during idle time of the tag. The idea is to accept longer execution times to save chip area and decrease power consumption. To the best of the author's knowledge this is the first time that this approach has been attempted with GPS. Several (parameterizable) architectures will be presented and discussed. The smallest full-precision arithmetic unit requires approximately 50000 gate equivalents and can calculate one commitment in about 2.4 million clock cycles. When using digit-level arithmetic, the size of the arithmetic unit can be reduced significantly. Using a digit size of 8 bits, the arithmetic unit takes only about 800 gate equivalents (plus RAM space for 560 bytes), at the price of spending approximately 66.6 million clock cycles for one commitment calculation. However, due to its short critical path this unit could be operated at frequencies up to about 290 MHz, when synthesized for UMC 0.13μm CMOS technology.

Keywords: GPS (Girault, Poupard, Stern), Authentication, Zero-Knowledge Protocol, RFID, Design Space, Hardware Implementation

Kurzfassung

GPS ist ein auf öffentlichen Schlüsseln basierendes Zero-Knowledge-Protokoll, welches einseitige Authentifizierung und, im Vergleich zu anderen Authenfizierungsverfahren, eine Fülle an Implementierungsvarianten anbietet. In seiner kleinsten Ausführung erlaubt GPS die Verwendung eines Coupon-basierten Ansatzes, bei dessen Verwendung auf Seite des Authentifizierenden während der Authentifizierung nur einfache Operationen mit ganzen Zahlen notwendig sind. Komplexere Varianten, welche die Berechnung von zahlentheoretischen Operationen, Hashes und Zufallszahlen erfordern, bilden die gegenüberliegende Grenze der Designmöglicheiten. Diese Arbeit präsentiert einen Ansatz bei dem die zahlentheoretischen Operationen (langsam) im Voraus berechnet werden, während sich das Tag im Leerlauf befindet. Die Idee ist eine längere Ausführungszeit in Kauf zu nehmen, um dafür Chipfläche zu sparen und die Leistungsaufnahme zu senken. Nach bestem Wissen des Autors ist dies das erste Mal, dass dieser Ansatz mit GPS versucht wurde. Mehrere (parametrisierbare) Architekturen werden präsentiert und besprochen. Die kleinste arithmetische Einheit, die mit Werten in voller Wortbreite arbeitet, benötigt ca. 50000 Gatteräquivalente, und ist in der Lage eine Berechnung in ungefähr 2.4 Millionen Taktzyklen auszuführen. Durch die Verwendung von arithemtischen Einheiten auf Ziffernbreite kann die Schaltungsgröße signifikant reduziert werden. Eine Arithmetikeinheit auf Ziffernebene mit 8-bit breiten Ziffern benötigt nur ca. 800 Gatteräquivalente, dafür aber ungefähr 66.6 Millionen Taktzyklen pro Berechnung. Allerdings kann diese Schaltung aufgrund ihres kurzen kritischen Pfades mit bis zu 290 MHz getaktet werden, wenn sie für UMC 0.13μm CMOS Technologie synthetisiert wird.

Stichwörter: GPS (Girault, Poupard, Stern), Authentifizierung, Zero-Knowledge Protokoll, RFID, Design Space, Hardware Implementierung

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Contactless technology has been getting more and more attention from the scientific community and also from the industry in recent years. Today passive radio-frequency identification (RFID) tags exist, which are capable of operating without their own power source, by extracting power from the electro-magnetic field of a nearby reader device. The same field is used to send data (commands) from the reader to the tag. Even though tags do not have their own power source, they are still capable of sending a response to the reader device. The physical principles on which this capability is based vary with the frequency of the reader field. This work will consider high-frequency (HF) tags, which are operated at 13.56 MHz, when referring to RFID tags in examples and application scenarios.

HF tags can send a response to the reader by means of so-called *load modulation*: The tag modulates its power consumption and thereby influences the electro-magnetic field. This can be detected by the reader.

There are numerous applications for RFID systems. The basic setup of such a system is depicted in figure 1.1. Tags are usually attached to an object, for example a product. Thereby goals like tracking a product's way, automatizing storage inventories, etc. can be achieved by employing several reader devices at adequate locations, which are connected to a central back-end database system.

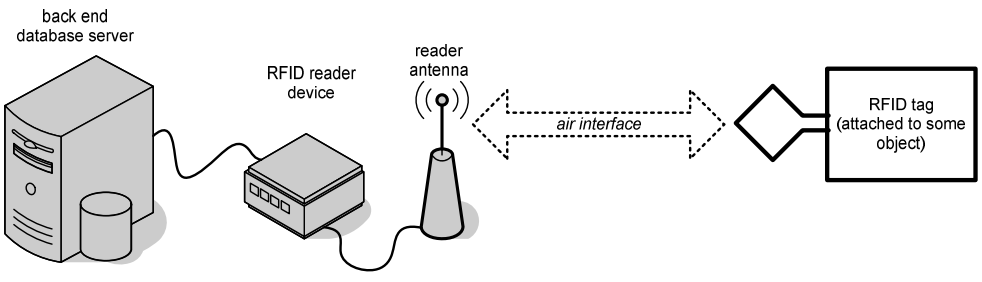

Figure 1.1: *Basic Setup of an RFID system. The reader device generates an electro-magnetic field, from which the tag draws its power. The field is also used for communication. Usually the reader device is connected to some sort of back-end database system, which stores information about the tags.*

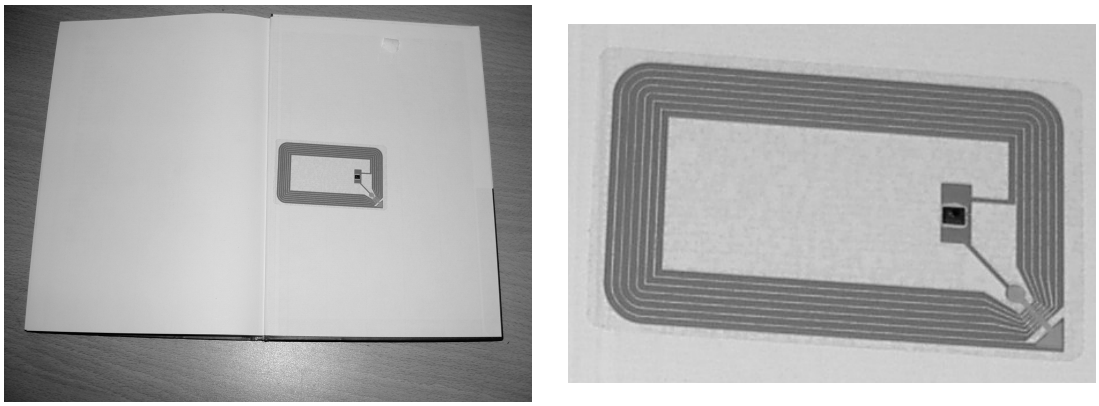

Figure 1.2: *Picture of an RFID tag in a book from the library of Graz University of Technology. The tags are used as anti-theft devices on one hand, and on the other hand for use with self-service terminals when borrowing books.*

1.1.1 Authentication in RFID Systems

One of the original goals of RFID systems was that tags should replace bar codes for product identification. To do so each product would be equipped with a tag. The tag would store the product's electronic identity code, and would reveal this code to an inquiring reader device.

However, one can easily think of applications where simple identification is not enough. If a tag was able to *prove* its identity, not just claim it, many more applications would become possible. Such an authentication would have to employ cryptographic methods in order to be secure, because the communication between the reader and the tag can be eavesdropped easily. Since tags incorporate digital circuits, this seems conceivable, at least from a theoretical point of view.

The basic components of such a cryptography-enhanced tag are shown in figure 1.3. The analog front end extracts power from the reader field. It also reconstructs the clock signal, demodulates received data and modulates data to send. The digital part performs the main job of the tag, whatever that might be. Non-volatile memory is also present, which enables the tag to store information even while it is inactive and not powered. Since cryptographic operations are quite complex in most cases, the enhanced tag also contains a dedicated cryptographic module, which is capable of performing the required operations.

1.1.2 Possible Fields of Application

As it has been said above, there are numerous scenarios where RFID tags or contactless smartcards, which are capable of *authentication* instead of identification only, can be employed. Two examples of such scenarios will be given here.

Access Control

Access control systems can be based on RFID tags. Instead of ordinary keys, persons take small objects (key rings, wrist watches, plastic cards,...) with an embedded tag with them. Reader devices are placed near points where access must be controlled, for example near a door. Therefore a person must not search for the key and put it in the lock in order

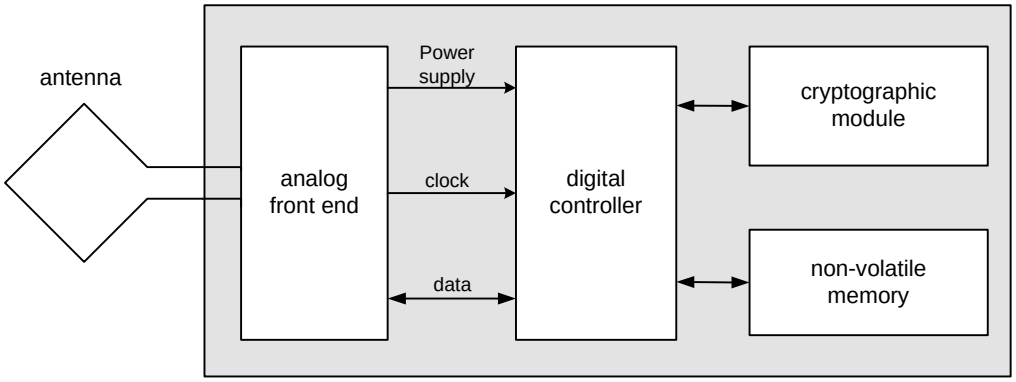

Figure 1.3: *A cryptography-enhanced RFID tag consists of several components: An analog front end which extracts power, clock signal and data from the reader field, a digital controller for the main functionality of the tag, non-volatile memory and a cryptographic module which performs the required cryptographic operations.*

to unlock the door. As soon as the tag is in range of the reader device, the door would unlock automatically.

It is obvious that an access control application requires more than identification by simply transmitting an identity code. Otherwise an eavesdropper who monitors the air interface would be able to duplicate the identity and reuse it later, to gain unauthorized access.

Genuineness of Products

Product piracy has become a mentionable issue in recent years, especially in the fields of textile industry, pharmaceutical industry, and tobacco industry, to name just a few examples. Fake products are produced at minimum costs, but are sold at high prices as original branded products. Most customers find the counterfeits indistinguishable from the originals when looking at them with the naked eye.

Now imagine cryptography-enhanced RFID tags, which are intrinsically tied to the products. Suppose further that in the near future most mobile phones and pocket computers (PDAs) will be equipped with an RFID reader (which is quite a likely assumption). Then customers would suddenly have the means of checking whether they really acquire what they are paying for.

There are even more vital applications for proofing a products genuineness. Concerning pharmaceutical products for example, telling the difference between a genuine drug and a (possibly dangerous) counterfeit could mean the difference between life and death.

1.1.3 Constraints of Passive RFID Devices

Although cryptography-enhanced tags sound very nice in theory, there are practical constraints which are difficult to meet. The fiercest constraint is power consumption. Tags intended for mass production also impose constraints on the maximum chip area (die size), but these constraints are more economical than technological.

Implementing cryptographic modules which adhere to these constraints is quite challenging. In recent years, however, progress has been made. [FDW04] and [FWR05] present

an implementation of the *Advanced Encryption Algorithm* (AES), which meets the constraints of passive RFID tags.

AES, however, is a symmetric cipher. Therefore the key which is used must be known to all parties involved, but must be kept secret to all possible adversaries. This is for example imaginable for small- or medium-scoped access control systems, like access control at a company's main entrance or its basement garage. Such closed-loop systems usually allow for the assumption of trust between all the parties.

However, when thinking of open-loop systems such as for example access control at public transportation systems, like in underground stations, this is much more difficult to achieve. Systems like the above-mentioned product genuineness system could not be realized at all, because there is no sufficient basis of trust to give all parties involved access to the secret key.

A remedy can be found by employing public-key cryptography. Public-key cryptography is based on the idea that each users has a *key pair* instead of just one single key. The key pair consists of a public key and a private key. The private key remains secret, while the public key can be made public without compromising security. The private key and the public key are linked in a mathematical way. One user performs an operation with the private key, and another user can then verify the result by using the public key only. Details about public-key cryptography will be presented in §2.3.3.

Since the public key can also be given to untrusted parties without affecting security, public-key cryptography can be used in scenarios where symmetric cryptography would not suffice. However, public-key cryptography usually involves number-theoretic operations, which in general require much more resources than symmetric ciphers, when implemented in hardware. Therefore for many years the general belief was that it is impossible to implement public-key cryptography in passive RFID devices. However, new and flexible authentication protocols, specifically tailored towards low resource requirements, question this doctrine.

1.2 First Glimpse at GPS

GPS is a very flexible public-key zero-knowledge authentication protocol named after its creators Marc Girault, Guillaume Poupard, and Jacques Stern. Its security is based on the discrete logarithm problem. It can be slightly modified to incorporate *elliptic curve cryptography*, instead of modular arithmetic.

One of the most important assets of GPS is the possibility to precompute a significant portion of the necessary calculations already before the protocol interaction starts. The only operations which a prover must do "online" during a protocol run are one integer multiplication and one integer addition. There is not even the need for modular reduction.

These properties make GPS an ideal candidate for use in RFID systems. Tags would not have to compute complicated operations in order to authenticate against a reader. Such operations would only be executed by the reader device, which is not bound by the same limitations as the tag and therefore has much more computing power.

GPS also offers a variety of implementation options; to cryptographers and to hardware designers. These options will be analyzed in §4. The main goal of this thesis is to explore which of these variants is best suited for hardware implementations with fierce resource constraints. Different architectures and their respective resource requirements will be presented in §6. Special attention will be paid to the so-called *coupon-recalculation approach*, which will be outlined in §4.2.3. In short, this approach is based on the idea that

a tag could use idle time in between authentications to perform precomputations, which speed up the actual authentication process. Since these precomputations are not bound by specific timing constraints, very slow but area- and power-efficient implementations could be possible. §6 will present the results which have been achieved with this approach.

1.3 Further Structure of this Work

This thesis focuses on three main goals: Presenting some basic background knowledge about the topics at hand, summarizing the most important features and specialties of the GPS authentication scheme, and investigating different variants of hardware implementations of GPS.

§2 will address the issue of authentication. Different authentication techniques will be described, examples will be given. Some of these techniques rely on complex mathematics. A summary of some important mathematical aspects, which are of interest in the scope of cryptography (especially in the scope of GPS), is given in §3. This section will not only present abstract definitions and theorems, but also arithmetic algorithms, which are of interest to hardware designers.

§4 will concentrate on the GPS authentication scheme. Its properties and design options will be described and analyzed.

§5 summarizes basic knowledge of hardware design. It will shortly present an overview of the most important aspects of CMOS technology, before analyzing the power consumption of CMOS circuits and ways to decrease it.

§6 will present different high-level models and hardware description language (HDL) implementations of GPS, which have been created during the course of this thesis. Different approaches and implementation variants will be compared among each other.

§7 will summarize the achieved results and present some ideas for future improvements and further work. It will also raise some questions, which maybe can be answered by the scientific community in the future.

Concerning the background knowledge it should be noted that since there are many topics to cover it will not be possible to present all of them in detail. The theoretic parts of this thesis (Chapters §2, §3, and §5) will focus on presenting those details which are of importance and interest to the implementations which will be discussed in §6. The rest will be covered in a more qualitative form, by means of examples. References to literature containing more detailed explanations will be given throughout this work.

Chapter 2

Authentication

The term *authentication* is used in (at least) two different contexts in the field of IT security:

- *Message authentication*: Upon receipt of a message, the receiving party verifies that the content of the message has not been altered along the communication channel.

- *Entity authentication*: One party verifies the identity claimed by another party.

The GPS scheme is primarily an entity-authentication scheme. Therefore message authentication is beyond the scope of this work. The remainder of this chapter will present a brief overview of state-of-the-art methods for entity authentication.

There are numerous literature sources which present the basic principles of authentication. The summary which will be given in this chapter is mainly based on [MvOV97] and [Osw06]. Since most of what will be presented can be considered common knowledge in the field of cryptography, references to these sources will not always be given explicitly.

2.1 Entity Authentication

2.1.1 Entities

In the context of authentication, *entity* is an umbrella term for everything that is capable of claiming (and subsequently possibly proving) a certain identity. That includes persons, computers, servers, smartcards, RFID tags,.... The reasons why these entities require authentication are various and depend on the context. For instance, users must prove their identity to a computer system in order to access their private files and data. This procedure is usually called *login*. It can also be the other way round: Imagine a web server which hosts an application for online banking. It seems reasonable that such a server proves its identity to the user, before the user enters any confidential information into the system. That way the user can be sure that the information is entered into the real system and not into a faked, malicious one. Yet another example would be an RFID tag, which is attached to some object. By proving its own identity, the tag implicitly also proves the identity of the object it is attached to. There are numerous scenarios, where such a prove could be useful. Some of them have been sketched in the introduction.

2.1.2 Identification versus Authentication

Although not all authors use the terms *identification* and *authentication* consistently, this work will adhere to the following definitions:

- *Identification*: Identification is the process in which an entity claims to have a specific identity. This is usually done by communicating an individual identifier (a name, a string, a number,...). Usually identifiers are unique within their frame of reference. No proof for the validity of the claim is given.
- *Authentication*: During authentication one entity (called the *prover*) proves its claimed identity to another one (called the *verifier*) by means of corroborative evidence.

In every-day's terms an example for identification would be a person telling his or her name to another person. Whereas authentication would mean that the first person proves the claim of identity by showing an ID card (cf. figure 2.1).

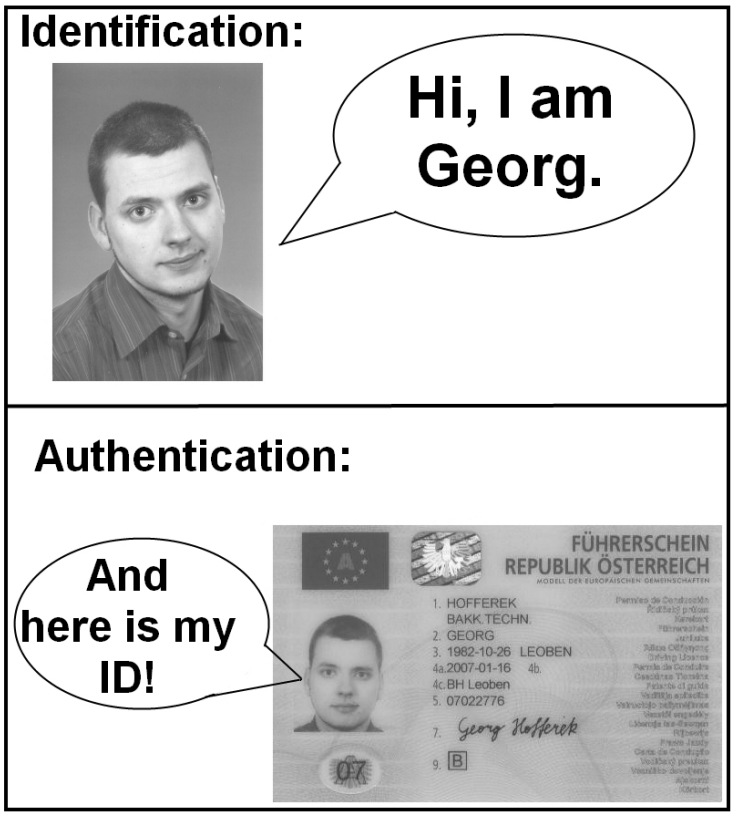

Figure 2.1: *Identification versus Authentication: By showing a valid ID card, issued by a trusted authority, a person can prove his or her identity to others.*

2.1.3 Means of Authentication

Authentication methods can be divided into three main categories, depending on which sort of evidence they are based on (cf. [MvOV97]). The three types of evidence are:

- Something the entity *inherently is*.

- Something the entity *possesses*, called a *(security) token*.
- Something the entity *knows*, called a *secret*.

The following paragraphs will briefly explain a few details on these categories.

Inherent Properties

Inherent properties of entities are mainly used for authenticating persons. Examples for properties which are suited for authentication include fingerprints, retinal patterns, voice patterns, or even the genetic DNA pattern of a person. Since authentication based on inherent properties of entities generally does not involve cryptographic techniques, it will not be considered any further.

Tokens

One of the simplest examples for a security token from every-day's life is a key to unlock doors. Applying technical terminology, a person authenticates to a door by unlocking it with the key, thus proving to be a member of the group of persons which are allowed to enter the door.

Other examples of tokens are ID cards, magnetic-stripe cards, smart cards, etc.

Secrets

A *secret* is a piece of information (for example a password, or a PIN), either shared by the prover and the verifier, or known to the prover only. During the authentication process either the secret itself, or some other information derived from the secret, is exchanged between prover and verifier. Thus the proving entity proves its identity by convincing the verifier that it knows the secret.

If the secret itself is transmitted during the authentication process, this is called *weak authentication*. Weak authentication will be discussed in greater detail in §2.2. There are also schemes where the secret itself is not transmitted, but only information derived from it by means of cryptographic operations. If it is infeasible (or even impossible) to retrieve the secret from this information, than this is called *strong authentication*. Strong authentication will be discussed in §2.3.

Combined Approaches

All means of authentication described above have their advantages and disadvantages. For example tokens and inherent properties are difficult to forge, in most cases. On the other hand tokens could be lost (and found by a malicious person), or even be stolen.

Therefore sometimes a combination of the aforementioned approaches is used. For example when drawing cash from an account at an ATM, one usually requires two things: The first is a bank card, which is either a magnetic-stripe card or a smart card. This card serves as security token. Second, one needs to know the PIN of the card. The PIN serves as a secret. Both the token and knowledge of the secret are required to successfully complete the procedure.

The advantage of such an approach is that someone trying to use a stolen token to impersonate the rightful owner cannot do any harm without knowing the secret. Thus the overall system is more secure than it would have been with just the token or just the secret.

2.1.4 Impersonation

If a malicious entity C is capable of convincing an honest entity B that it has an identity $A \neq C$, then C is said to have successfully *impersonated* A. The act of trying to do so is called an *impersonation attack*. Authentication protocols must ensure that impersonation is either completely impossible (which can hardly be achieved), or at least infeasible. Depending on the context of the application it might also be acceptable to ensure that the probability of successful impersonation is below a threshold specified by the application.

2.2 Weak Authentication

Weak authentication, also called *password authentication*, is a very simple form of authentication, based on a (time-invariant) secret called the *password*. The password can be a normal word, a number, a phrase, a sequence of characters, or any combination of those.

2.2.1 Historical Example

Passwords have been used for centuries to distinguish between allies and enemies in military proceedings. The commander of a military unit issues a password to all subordinates. When a guard encounters an unidentified person, the person would be stopped and questioned for the password. The person will be assumed to be an ally and be allowed to proceed, if and only if the correct answer is supplied. Otherwise the person will be assumed to be an enemy and will be arrested.

It should be noted that the authentication process does not prove the actual identity of the person, but only that the person belongs to a specific group. This is due to the fact that the same password is used by all persons of the group. Adapting the scheme to authenticate the identity of specific persons is possible, by increasing the number of passwords. If there are n persons in total $n \cdot (n-1) = (n^2 - n)$ passwords would be needed, because each pair of persons must share a secret password: Person A cannot use the same password for authenticating to person B and for authenticating to person C. If it were so, person B could use this password to impersonate A when authenticating to C. Therefore this system is only practical for a very small number of users.

There is another inherent problem: Anyone observing a (successful) authentication learns the password and therefore can subsequently perform impersonations. To minimize the threat posed by that, passwords are changed frequently. This, however, necessitates informing all legitimate users of the new password. That can be inconvenient and complicated if the number of users is large.

It should be noted that despite the obvious limitations and drawbacks many armies, including the Austrian Federal Army Forces, still use passwords as a first mean of authentication during missions and maneuvers.

2.2.2 Login Passwords

Login passwords are a widespread application of password-based authentication. The technique used is a little more sophisticated than the historical example in §2.2.1. Each user initially chooses a password and enters it into the computer system. However, the system does not store the password itself. Instead the password is fed to a one-way (hash) function. The ouput of this function is stored. This makes it impossible to determine a user's password from the data stored in the system.

When logging in, a user first claims an identity, usually by entering a user name. Afterwards the user proves this identity by entering the password. The system feeds the user's input to the same one-way function which was used during the password setup phase and compares the result with the stored value. If there is a match, authentication is successful and the user is logged in. Otherwise the user is rejected.

Since in this scenario the users authenticate only to the computer system, and not among each other, each user must only remember one password. The users must, however, ensure that their passwords remain secret. Malicious users could try to spy on passwords by looking over other users' shoulder while they enter their password, or, more sophisticated, by using key-logger programs to eavesdrop on the communication between keyboard and login process. Once a malicious party knows a user's password, it can impersonate this user.

Another problem with passwords is that an imposter can simply try to guess a user's password. Many users use obvious passwords, such as for example the first name of their child, the licence number of their car, the date of their birth,.... To prevent that, users are encouraged to choose secure passwords, consisting of a (random) combination of letters (in upper *and* lower case), digits, and special characters. Such passwords are, however, hard to remember, which is why many users tend to write them down and put the note somewhere near the computer. This is of course at least as bad as choosing an insecure password.

There is yet another course of action an attacker can take: A so-called *exhaustive search attack*. To mount such an attack the attacker simply tries out all possible passwords. To circumvent such an attack, a system can limit the number of (unsuccessful) authentication attempts for a user. After failing to authenticate for this number of times the system could deactivate the user's account until further notice.

2.2.3 Further Reading

Further information on password authentication, including some enhancements to the basic ideas presented above, can be found in [MvOV97].

2.3 Strong Authentication – Challenge-Response Protocols

Strong authentication protocols are those, which do not require to transmit an entity's secret itself during execution. This can be achieved in the following way: The verifier issues a *challenge* and communicates it to the prover. The prover then answers with a *response*, derived from the challenge and the prover's secret. Therefore such authentication methods are also called *challenge-response protocols*.

The way in which the response is derived usually employs cryptographic techniques to ensure that it is impossible (or at least infeasible) to determine the secret from knowing the challenge and the response only. Therefore an eavesdropper does not learn the secret.

A so-called *replay attack* is also circumvented: Suppose an eavesdropper E has monitored A authenticating against B, and now tries to impersonate A. If B chooses a different challenge each time (for example by selecting challenges at random), E cannot easily impersonate A, because the response E would have to send to B is dependent on the challenge. Therefore the response given by A, which has been observed by E before, is useless, as long as the cryptographic means by which it has been derived are secure.

2.3.1 A Simple Example

§2.2.1 presented a sort of a password-based authentication scheme frequently used by the military. This section will present another interesting curiosity from the realm of military security. According to information obtained from [FAFoA], the Austrian Federal Army Forces (and most probably also many other armies) have developed an improvement to the scheme presented in §2.2.1. Instead of issuing a normal password, the users agree on an integer number. This number is called *Losungszahl* in German, which roughly translates to "password number".

The protocol then works as follows: Suppose the *Losungszahl* is 17. Whenever a guard wants to authenticate an approaching person, the guard shouts a number between 0 and the *Losungszahl*, for example 14. This number is the challenge. The person who wants to authenticate then responds with the difference between the challenge and the *Losungszahl*, which would be 3 in this example. Thereby knowledge of the secret *Losungszahl* is proven, without actually revealing the *Losungszahl* itself.

Strictly speaking this is not a strong authentication protocol, because it is very easy to recover the secret from the challenge and the response, especially when knowing the rules of the protocol. Therefore this scheme is clearly in violation of *Kerckhoffs' principle*. But even without knowing the protocol, after observing a few protocol runs, an intelligent observer would find it quite easy to recover the *Losungszahl*. However, the correlation between challenge, *Losungszahl* and response must be a very simple one, because the calculations are done by mental arithmetics, by people who are possibly in a stressful situation.[1]

Nevertheless a simple replay attack can be prevented and according to [FAFoA], the security of this protocol is largely based on the assumption that an eavesdropping enemy has a different native language. That makes replay attacks the most likely ones, because they can be done without understanding the meaning of what has been said during protocol interaction. Furthermore, being incapable of understanding the used language makes interpreting the observed challenges and responses more difficult.

2.3.2 Authentication based on Secret-Key Cryptography

Secret-Key Cryptography

Among other things secret-key cryptography provides encryption functions E_K which can transform a plain text P into a cipher text C, using a key K:

$$C = E_K(P) \tag{2.1}$$

The inverse function E_K^{-1}, called decryption function, transforms the cipher text back into plain text:

$$P = E_K^{-1}(C) \tag{2.2}$$

These functions must satisfy certain requirements: For example it must be infeasible to determine the plain text corresponding to a specific cipher text without knowing the key K. Moreover it must be infeasible to recover the key, when knowing a plain text and its

[1] Therefore [FAFoA] imposes very narrow constraints on the choice of the *Losungszahl* and the challenge: The *Losungszahl* must be a positive integer less than 10 and the challenge must be a positive integer, less then or equal to the *Losungszahl*. Obviously Austrian military commanders do not think very highly of the arithmetic capabilities of their subordinates.

corresponding cipher text. A more formal definition of the requirements for such functions can be found in [MvOV97]. Since the same key is used for encryption and decryption, secret-key cryptography is often also referred to as *symmetric cryptography*. Today many such encryption functions (also called *ciphers*) exist. One which is widely used is the *Advanced Encryption Standard* (AES) (cf. [NIS02]).

Usage in Authentication Protocols

Ciphers like AES can be used to construct challenge-response protocols in a straightforward way: Suppose a prover and a verifier share a secret key K. When the prover requests to be authenticated, the verifier sends a (randomly chosen) plain text P to the prover. This plain text is the challenge. The prover uses the secret key K to encrypt P, obtaining a cipher text C, which is sent back to the verifier. Using the same secret key K, the verifier can then either decrypt C and check, whether the result equals P, or also encrypt P and compare the result with the response of the prover. If the response matches the verifiers expectations, the verifier can safely assume that the prover knows the secret key K. Without it, the prover would have been unable to answer the challenge correctly. Moreover an eavesdropper who observed the challenge and the response is unable to easily derive the secret key.

There are numerous variants of the basic idea sketched above. Some offer additional features, such as mutual authentication, key update or concurrent session-key exchange. Several of these variants are described in [MvOV97] or [Osw06].

Problems

Although challenge-response protocols based on secret-key cryptography offer many advantages, such as for example a high level of security, there are also some disadvantages. One of the most serious drawbacks is the key management problem. Since the key must remain secret a system consisting of n entities would need $(n^2 - n)$ keys, and each user would be required storing a subset of n of these keys. The reason for that is the same that has already been outlined in §2.2.1.

Again, if users only authenticate against one central verifier, and not among each other, the number of keys can be reduced to n (= one per user). Still, this can be a limiting issue, if the number of users is very large.

2.3.3 Authentication based on Public-Key Cryptography

Public Key Cryptography

Public-key cryptography has been developed in the late 1970's to overcome some of the shortcomings of secret-key cryptography. It involves key pairs (e, d), consisting of a so-called *public key* e and a so-called *private key* d. As the names suggest the public key is intended to be made public, for example by posting it in a directory on the Internet, whereas the private key must remain secret. The basic idea of public-key cryptography is that encryption is done by using the public key e, whereas for decryption the private key d is needed. Let P be a plain text and C the corresponding cipher text. Then the following equations describes the principle of public-key cryptography:

$$C = E_e(P) \tag{2.3}$$

$$P = E_d^{-1}(C) \tag{2.4}$$

Since a different key is used for encryption and decryption, public-key cryptography is often also referred to as *asymmetric cryptography*. Due to this asymmetry everyone is capable of encrypting a plain text P, because the *encryption key* e is publicly known. However only the holder of the corresponding *decryption key* d is able to decrypt the resulting cipher text C.

The public key and the private key are linked in a mathematical way, so that it is infeasible to compute the private key when given the public key only. This can be achieved by the use of mathematical functions, which are relatively easy to compute but which possess an inverse function which is very hard to compute. Number theory provides a few such problems, for example the *integer factorization problem* (IFP): It is very easy to choose two large prime numbers p, q at random and multiply them to obtain the product $n = p \cdot q$. However, no efficient way is known to determine the factors p, q when given n only.

Problems like the IFP are called *hard mathematical problems*. Another hard problem on which many public-key crypto-systems are based is the so-called *discrete logarithm problem* (DLP). The DLP and its mathematic basics will be discussed in §3.1.

One of the first public-key crypto-systems was the RSA system, which is still very popular. It is named after its creators Ronald L. Rivest, Adi Shamir and Leonard Adleman (cf. [RSA78]). Its security is related to the IFP: If solving the IFP was possible, RSA would not be secure anymore. For the sake of completeness it should be noted that the opposite statement has not yet been proven: It is unknown whether breaking the RSA scheme necessitates solving the IFP, or whether there are also other possibilities.

Further details about public-key crypto-systems are beyond the scope of this thesis. Formal definitions and requirements are summarized in [MvOV97]. Another good introduction to the topic, including a survey of possible attacks, can be found in [Osw06].

Usage in Authentication Protocols

The basic concept how public-key cryptography can be used in authentication protocols is the same as with secret-key cryptography: The verifier chooses a plain text P at random. Then M is encrypted with the public key of the prover, which is known to the verifier. The resulting cipher text C is then sent to the prover as challenge. The prover is able to decrypt C by using the private key. The decrypted plain text is sent back to the verifier, who checks whether it equals the original plain text. If so, authentication was successful.

Several variant of this basic idea exist. More information about them can be found in [MvOV97].

Advantages and Drawbacks

The main advantage compared to protocols based on secret key cryptography is that, although each user is able to authenticate against each other user, only one key pair per user is necessary. However, this flexibility comes at a price: Ensuring the authenticity of public keys and their correct mapping to user identities is a complex problem. So-called *public key infrastructures* (PKI) are required to take care of the distribution, authentication, and revocation of public keys. The exact requirements of PKIs depend on their field of application and will not be discussed here any further. Further information on this issue is available for example in [Osw06].

Another drawback of public-key cryptography is that it usually requires number-theoretic operations. Performing such calculations is in most cases more costly (in terms of computing time, hardware resources, power and energy consumption) than performing operations for secret-key cryptography.

2.4 Zero-Knowledge Protocols

Challenge-response protocols, as they have been outlined in §2.3, have yet one small disadvantage: They convey more information than would be necessary during their execution. For example with each protocol run an eavesdropper learns a valid plain-text cipher-text pair. Given enough such pairs the eavesdropper might be able to deduce some information about the prover's secret. So in a sense some information about the secret is leaking.

Zero-knowledge techniques address this problem. The basic idea is that during execution of a zero-knowledge protocol no information about the secret is leaking. Such protocols aim at conveying one single bit of information only: Whether the prover knows the secret, or not.

2.4.1 The Principle of Zero-Knowledge Protocols

The best way to understand zero-knowledge protocols is an illustrative example. The story which will be told in this section is an adaption of similar stories in [QGB90] and [Wik07b].

The Story

Suppose there are two persons, for some reasons[2] named Alice and Bob, who live nearby a very special cave (cf. figure 2.2): This cave splits into two paths shortly after the entrance. The paths form nearly a circle. However the passage from one path to the other is blocked by a magic door. To open this door, one needs to say the *magic word*.

Now suppose that Alice knows the magic word, but Bob does not. Since Bob is a curious person, he offers to pay Alice a huge amount of money, if she tells him the magic word. Alice agrees, but she insists that Bob gives her the money *before* she tells him the secret, because she does not fully trust Bob, and fears that he will betray her and keep the money, once he knows the word. Bob on the other hand requests that Alice proves that she knows the secret magic word before he pays her, because he fears that Alice just pretends to know the word. So what they need, besides a lesson in interpersonal trust, is a way for Alice to prove her knowledge, without revealing the knowledge itself.

The Solution

After careful consideration Alice and Bob agree on the procedure depicted in figure 2.2:

1. Alice enters the cave and chooses one of the two paths. She walks along the chosen path until she reaches the magic door. While she is doing so, Bob waits outside the cave, unable to see which path Alice is choosing. This step is called *commitment*, because Alice commits herself to the path she chooses.

[2] These reasons are outlined in [Wik07a].

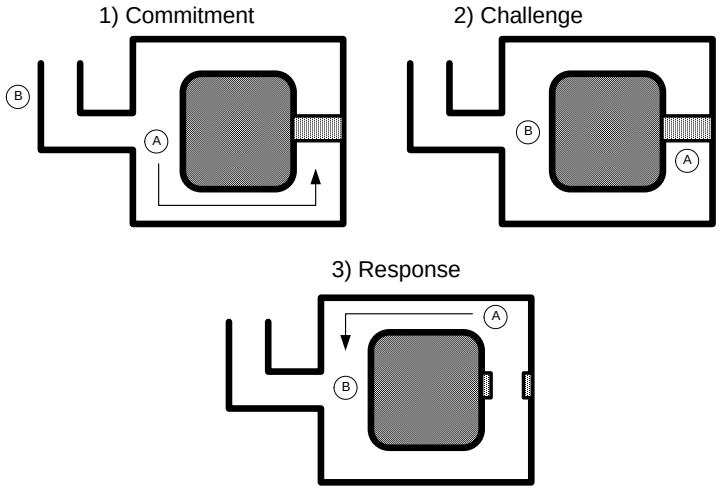

Figure 2.2: *Illustrative example of a zero-knowledge proof: Alice's knowledge of the* magic word *to open the secret door is proven to Bob, without Alice having to reveal the secret. (Adapted from [Wik07b].)*

2. Bob enters the cave and walks to the place where the two paths separate. He then chooses one of them and shouts his choice to Alice. After that he waits for Alice to return to him along the path he requested. This step is the *challenge*.

3. Now there are two possibilities: Either Alice is already in the path Bob requested. Then she simply returns along this path. Or she is in the other path. Then she must use the magic word to open the magic door and slip through. Since Bob is waiting on the other side of the cave, he does not learn the magic word. He does not even know whether Alice has actually used it or not. The important fact is that no matter which path Alice has chosen initially, she is able to respond properly to Bob's challenge, if she knows the magic word.

If Alice is unable to return via the path requested by Bob, he assumes that she does not know the magic word. However, even if she returns to Bob as requested, Bob is still not convinced that Alice knows the magic word. Obviously there is a 50% chance that Alice was lucky and initially chose the path Bob later requested.

So they repeat the procedure, over and over again. One single failure of Alice would suffice to make Bob believe that she is a liar. However, if she can satisfy Bob's challenge in all of k tries, the probability that she was just lucky is 2^{-k}. So after successfully completing for example 100 rounds of the protocol, Bob is convinced that Alice is either a visionary, capable of looking into the future and foreseeing his challenges, or that she really knows the magic word.

Remarks

The procedure outlined above is really a zero-knowledge proof, because Bob does not learn anything except that Alice knows the magic word. To prove this statement suppose there is a third person, named Charly, who accompanies Bob during the execution of the protocol. Although Charly witnesses everything, he cannot be sure whether Alice really

knows the magic word. From his perspective Alice and Bob could be conspiring against him, by having agreed on a sequence of challenges beforehand. That means that Bob does not get any information which he could not also have created himself, from a third person's (Charly's) point of view.

It also means that a zero-knowledge proof conveys information only to someone who has been interactively involved in the protocol execution, for example by being allowed to choose challenges. Someone just watching does not learn anything. Therefore the correct term for such a proof would be *interactive zero-knowledge proof*. For the sake of brevity it is often simply referred to as *zero-knowledge proof*.

2.4.2 Properties of Zero-Knowledge Schemes

A zero-knowledge protocol must fulfil three properties in order to be suited for use in authentication. These are:

1. *Completeness* property
2. *Soundness* property
3. *Zero-Knowledge* property

The subsequent sections will define and explain the meaning of these properties. For the sake of clarity the definitions will be slightly simplified to avoid too many references to complexity theory. More precise definitions can be found in [MvOV97].

Completeness Property

A protocol is said to be *complete*, if an honest prover is able to successfully complete the protocol with an honest verifier, with overwhelming probability. The level of the acceptable failure probability is application- and context-dependent.

In other words, completeness ensures that the protocol can be used by honest parties in a meaningful way. A protocol which does not have the completeness property is useless for practical applications, since honest provers could be rejected.

Soundness Property

If an imposter tries to impersonate an honest prover and any honest verifier will reject this imposter during protocol execution with overwhelming probability, then the protocol is said to be *sound*. Again, the definition of "overwhelming" depends on the application.

[MvOV97] gives a slightly more complicated definition for soundness, which has an interesting facet. In short [MvOV97] defines a protocol to be sound if the following statement holds: *If an imposter is accepted by a verifier with non-negligible probability, then this imposter can be used to deduce information equivalent to the impersonated prover's secret.*

On first sight it is hard to see how this definition is related to the one given above. The correlation reveals itself by considering the following *proof by contradiction*: Suppose the prover's secret is the factorization of a large number n. Since the integer factorization problem is considered to be a hard problem, deriving the secret (the factorization) is intractable. Suppose further that the protocol is sound. Then the definition of [MvOV97] implies that imposters cannot complete the protocol with non-negligible probability, because the existence of such an imposter would lead to a contradiction.

Assume such an imposter existed. Then according to the definition given by [MvOV97] this imposter can be used to deduce the impersonated user's secret, or equivalent information. That means that this imposter would provide a mean of determining the factorization of n. That is a contradiction to the assumption that the IFP is intractable. Therefore such an imposter cannot exist.

Zero-Knowledge Property

If it is possible to produce (simulated) records of protocol executions without knowing the secret information, and if a third person cannot distinguish these simulated records from records of real protocol runs, then the protocol is said to have the *zero-knowledge* property.

In other words that means that all the information which is exchanged during protocol execution could also have been produced without knowing the secret. However, that does *not* imply that it is possible to successfully complete the protocol without knowing the secret. As it has already been explained in the remarks of §2.4.1, a simulator could choose the challenges *before* being forced to choose a commitment. By doing so, simulated records can be created, which are indistinguishable from real records during later inspections. However, during protocol execution one is forced to choose a commitment before receiving a challenge. Therefore knowledge of the secret is required in order to be able to answer the challenge with overwhelming probability.

There are three notions of the zero-knowledge property, which are distinguished by the exact definition in which way the distribution of simulated records differs from the distribution of real records (cf. [Wik07b]).

1. *Perfect zero-knowledge*: The distributions of simulated and real records are the same.
2. *Statistical zero-knowledge*: The statistical difference of the distributions is negligible.
3. *Computational zero-knowledge*: No efficient algorithm exists to distinguish the distributions.

From an applications point of view, the difference between these three notions is, however, not of notable interest.

2.4.3 Probability of Impersonation

It should be noted that in all zero-knowledge protocols, as well as all challenge-response-based authentication protocols (cf. §2.3) an imposter has always a non-zero probability of success. This is due to the fact that the number of possible responses is finite. Let m be the number of possible responses. Then an imposter has a probability of success of $\frac{1}{m}$, by simply guessing a response at random. Challenge-response protocols usually have such a large m that this probability is negligible. Some zero-knowledge protocols, however, inherently have only a small number of possible responses. For example in the protocol outlined by the story in §2.4.1, Alice has only two possibilities to "respond": Returning via path 1 or returning via path 2. That would lead to an unacceptably high probability of success of $\frac{1}{2} = 0.5$ for an imposter.

Fortunately the probability of success of an imposter can be lowered from $\frac{1}{m}$ to $\left(\frac{1}{m}\right)^k = \frac{1}{m^k}$ by simply repeating the protocol k times. This might be undesirable with challenge-response protocols, because each protocol execution leaks a certain amount of information about the secret. Zero-knowledge protocols, however, do not have the same problem.

2.4.4 List of (Some) Zero-Knowledge Protocols

There are many different zero-knowledge protocols. For example the Fiat-Shamir protocol, the Feige-Fiat-Shamir protocol, the Guillou-Quisquater (GQ) protocol, the Schnorr protocol,.... Details about these protocols can be found in [MvOV97].

Details about the GPS protocol will be presented in §4.

2.5 Further Reading

The overview of authentication techniques presented in this chapter barely scratched the surface of the topic. Authentication is an important topic, with many applications and therefore receives a lot of attention from the researchers in the field of cryptography. [MvOV97] provides many more detailed insights on authentication mechanisms and protocols, as well as the cryptographic techniques employed for authentication.

A good introduction and overview of the topic is also given by [Osw06]. Mathematic basics can be found in [FT03].

Chapter 3

Mathematical Background

This chapter will present some basic mathematic principles and definitions, which are of particular importance in the context of cryptography. The focus will lie on establishing a common notation and vocabulary. Therefore some details, such as for example proofs or well-known algorithms and theorems, will be omitted. Furthermore, this survey will only include topics which will be of further interest in the remaining part of this thesis.

A more detailed overview of mathematical basics in cryptography can be found in [MvOV97]. Further references to literature on specific topics will be given throughout this chapter.

3.1 Groups and the Discrete Logarithm Problem

3.1.1 Definition and Axioms

A set G, together with a (binary) operation $\otimes$ is called a *group*, if the following axioms are satisfied:

1. *Closure:* $\forall a, b \in G : (a \otimes b) \in G$
2. *Associativity:* $\forall a, b, c \in G : (a \otimes b) \otimes c = a \otimes (b \otimes c)$
3. *Neutral element:* $\exists e \in G : \forall a \in G : a \otimes e = e \otimes a = a$, where e is called the "identity element" or "neutral element" (sometimes also referred to as $\mathbf{0}$).
4. *Inverse element:* $\forall a \in G : \exists a' \in G : a \otimes a' = a' \otimes a = e$, where e is the identity element, defined in axiom 3. a' is called the *inverse* of a in G.

If the group operation is also commutative, that is if the condition

$$\forall a, b \in G : a \otimes b = b \otimes a \tag{3.1}$$

holds, then the group $(G, \otimes)$ is called an *Abelian* group.

Example

The set of integers $\mathbb{Z}$ and the addition operation $+$ form a group $(\mathbb{Z}, +)$, which has the neutral element 0. Since the addition of integers is a commutative operation, this group is actually an Abelian group. Note that the underlying set, the set of integers $\mathbb{Z}$, has an infinite number of elements.

$(\mathbb{Z}, \cdot)$ (where $\cdot$ is the integer multiplication operation) is *not* a group, since obviously not all of the axioms are fulfilled. For example multiplicative inverses cannot be found for all elements of the set.

3.1.2 Finite Groups

Additive Groups

When the underlying set of a group has only a finite number of elements, the group is called a *finite group*. Let $\mathbb{Z}_n = \{x \in \mathbb{Z} : 0 \leq x < n\}$ be the set of integers from 0 (inclusive) to n (exclusive). Furthermore, let $+_n$ be a modular addition operator, such that $a +_n b = (a + b) \bmod n$. Then it can easily be shown that $(\mathbb{Z}_n, +_n)$ is an (Abelian) group, with 0 being the neutral element. Since its group operation is based on addition, such a group is often referred to as an *additive group*.

Multiplicative Groups

For all integers $m \in \mathbb{Z}_n$ which are coprime to n, there exists an integer $m^{-1} \in \mathbb{Z}_n$, which satisfies $m \cdot m^{-1} = 1 \bmod n$. This integer m^{-1} can be computed with the extended Euclidean algorithm and is called the *multiplicative inverse* of m with respect to n.

Now let $\mathbb{Z}_n^* \subset \mathbb{Z}_n$ be the set of all integers from $\mathbb{Z}_n$, for which a multiplicative inverse exists. That is

$$\mathbb{Z}_n^* = \{m \in \mathbb{Z}_n : \exists\, m^{-1} \in \mathbb{Z}_n : m \cdot m^{-1} \equiv 1 \pmod{n}\} \tag{3.2}$$

Then it can easily be shown that this set of invertible integers modulo n, together with modular multiplication $\cdot_n$ forms an (Abelian) group $(\mathbb{Z}_n^*, \cdot_n)$, with 1 being the neutral element. $\cdot_n$ is defined analogue to modular addition: $a \cdot_n b = (a \cdot b) \bmod n$

Note that if n is a prime number, all elements of $\mathbb{Z}_n$ except for 0 have a multiplicative inverse, since $\gcd(x, n) = 1$ is sufficient for x being invertible.

Since the group operation of such a group is based on multiplication, it is often referred to as *multiplicative group*.

Order of Groups and Elements

The order of a group, denoted by $|G|$ is the number of its elements, or ∞, if the group has an infinite number of elements.
The order of an element a is defined as the least positive integer k so that

$$\underbrace{a \otimes a \otimes a \otimes \ldots \otimes a}_{k \text{ times}} = e \tag{3.3}$$

where e denotes the group's neutral element and $\otimes$ denotes the group operation. If no such integer exists, the element is said to have infinite order.

Cyclic Groups and Generators

When all elements of a group G can be generated by successively applying the group operation to an element a, then the group is called a *cyclic group* and the element a is

called a *generator*. More formally: The group $(G, \otimes)$ is cyclic, and the element a is a generator if the following condition holds:

$$\forall x \in G : \exists\, k \in \mathbb{N} : x = \underbrace{a \otimes a \otimes a \otimes \ldots \otimes a}_{k \text{ times}} \tag{3.4}$$

If the condition only holds for a subset $G' \subset G$, then a generates a so-called *subgroup* $(G', \otimes)$.

3.1.3 The Discrete Logarithm Problem

Let a, x be elements of the group $(G, \otimes)$. Then an integer k satisfying

$$x = \underbrace{a \otimes a \otimes a \otimes \ldots \otimes a}_{k \text{ times}} \tag{3.5}$$

is called *discrete logarithm* of x with respect to base a. The term "discrete logarithm" originates from the fact that when considering a multiplicative group $(\mathbb{Z}_n^*, \cdot_n)$

$$x = \underbrace{a \otimes a \otimes a \otimes \ldots \otimes a}_{k \text{ times}} = \underbrace{a \cdot a \cdot a \cdot \ldots \cdot a}_{k \text{ times}} = a^k \tag{3.6}$$

In that case k is the logarithm of x in base a. When considering finite groups the term "discrete logarithm" is used. Note that if the order of the basis a is finite, the discrete logarithm k is unique only modulo the order of a.

Finding the discrete logarithm k of a given element x with respect to a given element a is called the *discrete logarithm problem* (DLP). For some groups, the DLP is easy to solve. Considering for example an additive group $(\mathbb{Z}_n, +_n)$, discrete logarithms can be computed efficiently by means of (integer) division: Suppose $x = 15, a = 3$, with $x, a \in (\mathbb{Z}_{17}, +_{17})$. Then, adhering to the abstract definition given by equation 3.5, the discrete logarithm of x with respect to basis a is the number of times a must be added to itself to obtain x. This number can easily be calculated by dividing x by a: $k = x/a = 15/3 = 5$. Even if a is added to itself so many times that modular reduction occurs, it is still easy to find discrete logarithms in additive groups.

For other groups, such as for example for multiplicative groups $(\mathbb{Z}_n, \cdot_n)$, the DLP is considered to be intractable, provided that the group has enough elements to thwart exhaustive search. In such groups the DLP is a *hard mathematical problem*. Therefore it can be the basis for crypto systems: It is easy to find an element x by applying the group operation k times to an element a. It is, however, computationally infeasible to do the opposite. Namely determining k, when given a and x and knowing that x has been obtained by applying the group operation k times to a.

3.1.4 Notation

For the sake of brevity, $+_n$ and $\cdot_n$ will hereafter simply be written as $+$ and $\cdot$, whenever it is clear from the context that modular (group) operations are meant. Furthermore the remainder of this thesis will focus on multiplicative groups over invertible integers modulo an integer n. Therefore the set $\mathbb{Z}_n^*$ will be identified with the corresponding multiplicative group and will be used as a shorthand for $(\mathbb{Z}_n^*, \cdot_n)$. The multiplicative inverse of an element x will be denoted x^{-1}.

3.1.5 Further Reading

This section only gave a very short overview of the basic terminology and concepts of group theory. A more detailed and complete introduction can be found in [FT03]. A summary of group-related definitions and theorems which play an important role in cryptography can also be found in [MvOV97].

3.2 Groups over Points of Elliptic Curves

§3.1 concentrated on groups over subsets of the set of integers $\mathbb{Z}$. However, it is also possible to construct groups with other basic elements than integer numbers. An example for such groups are groups over points of elliptic curves. Elliptic curves play in increasingly important role in cryptography. This section will give a very brief introduction on the most important mathematic basics of elliptic curves in cryptography.

3.2.1 Elliptic Curves

The following equation, a so-called *Weierstrass equation*, defines an elliptic curve E:

$$E: \; y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{3.7}$$

where coefficients the $a_1,\ a_2,\ a_3,\ a_4,\ a_6$ are all elements of a (finite) field[1] $\mathbb{F}$. Strictly speaking, the coefficients a_i must satisfy some additional requirements, which are addressed (for example) in [HMV03]. For finite fields $\mathbb{F}$ with a characteristic not equal to 2 or 3, equation 3.7 can be shortened to

$$E: \; y^2 = x^3 + ax + b \tag{3.8}$$

with $a, b \in \mathbb{F}$, without loss of generality.

For the remainder of this thesis only curves over prime fields (with prime numbers greater than 3) will be considered. Such fields will be denoted $\mathbb{F}_p$ or $GF(p)$, where the prime number p is the number of elements of the field. Therefore equation 3.8 can be used instead of equation 3.7 hereafter.

3.2.2 Points on Elliptic Curves

Affine Representation

A tuple $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying equation 3.8 is called a *point* of the elliptic curve E. The coordinates x and y are also referred to as the *affine* representation of the curve point.

Point Operations

Given an elliptic curve E (as defined by equation 3.8 over a finite field $\mathbb{F}_p$) and two points $P_1,\ P_2$ on the curve represented by their affine coordinates (x_1, y_1) and (x_2, y_2), one can

[1] A *field* $\mathbb{F}$ is a mathematical structure, similar to a group, but with two binary operations instead of just one, and with more axioms which must be fulfilled. Essentially fields allow all commonly known arithmetic operations, such as addition, subtraction, multiplication, "division" (by multiplicative inversion), ... More details about (finite) fields can be found in [HMV03], [MvOV97] and [FT03].

define an operation called *point addition*:

$$\begin{aligned} P_3 = (x_3, y_3) &= P_1 + P_2 = (x_1, y_1) + (x_2, y_2) \\ x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \\ \lambda &= \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P_1 = P_2 \end{cases} \end{aligned} \tag{3.9}$$

The formulas shown in equation 3.9 are derived from a geometric rule, called the *chord-and-tangent rule*, which clearly depicts the addition of two curve points when considering curves over non-finite fields, such as for example the real numbers $\mathbb{R}$. (cf. [Wer02] for details.)

From equation 3.9 it can also be seen that two different cases are distinguished when adding points: The addition of two distinct points and the addition of a point to itself. The latter is also referred to as *point doubling*.

Since the (affine) coordinates of the curve points (x_i, y_i), as well as the curve parameters a, b are all elements of the underlying (finite) field $\mathbb{F}$, all operations of equation 3.9 (such as for example addition, subtraction, squaring, ...) are operations within this (finite) field $\mathbb{F}$. Note that division is realized by first finding the multiplicative inverse of the divisor and then multiplying it with the dividend.

Scalar Multiplication

A curve point P can be multiplied by a scalar k, by repeatedly adding the point to itself k times:

$$k \cdot P = \underbrace{P + P + \ldots + P}_{k \text{ times}} \tag{3.10}$$

Details how the scalar multiple of a point can computed will be presented in §3.4.2.

Projective Representation

In addition to affine representation, there is also a so-called *projective* representation of curve points, which involves three instead of just two coordinates. Projective coordinates (x, y, z) are derived from affine coordinates by substituting (x, y) with $(\frac{x}{z^c}, \frac{y}{z^d})$, with c, d being small natural numbers. The rationale behind using projective coordinates, which obviously require more storage space, is that point operations with projective coordinates do not require division. Since division is realized by multiplicative inversion, which is considered to be a costly operation, avoiding it is a fair trade-off for the extra storage costs. Formulas for point addition using projective coordinates can be obtained by performing the substitution $(x, y, z) \rightarrow (\frac{x}{z^c}, \frac{y}{z^d})$ in equation 3.9 and clearing denominators.

It should be noted that, in contrast to affine representation, projective coordinates are not unique. That is, there is more than one triple (x, y, z) representing the same curve point P. When given a point $P = (x, y, z)$, each triple $(\alpha^c x, \alpha^d y, \alpha z)$, with $\alpha \in \mathbb{F},\ \alpha \neq 0$, is a valid representation of the same point.

To transform an affine representation (x, y) into a projective representation, one simply sets $z = 1$ and obtains $(x, y, 1)$ as a valid representation. The inverse transformation, from projective coordinates to affine coordinates, requires the inversion of the z-coordinate: $(x, y, z) \rightarrow (xz^{-c}, yz^{-d})$

In practice, multiple point operations (such as for example multiple additions during scalar multiplication) are usually performed with points in projective representation, before converting the final result to affine representation. Therefore the number of (costly) inversion operations is greatly reduced.

3.2.3 Group Law

After defining points on elliptic curves and an addition operation for such points, one question arises: Do the set of points on an elliptic curve and the addition operation (as defined by equation 3.9) form a group, as defined by the axioms in §3.1.1? Unfortunately the answer is "no". Fortunately, however, this can be changed with a few additional definitions.

According to §3.1.1 a group needs a neutral element. By defining the *point at infinity*, denoted ∞, as neutral element, this requirement can be fulfilled. The point at infinity does not have an affine representation. In projective coordinates, however, it can be represented by $(x, y, 0)$, where x and y must be chosen to fulfil the curve equation.

There is one other requirement from §3.1.1 which must be satisfied: Each element of the group must have an inverse, with respect to the group operation. This requirement can be fulfilled by defining the inverse of a point P

$$-P = -(x, y) = (x, -y) \tag{3.11}$$

where $-y$ denotes the additive inverse of y in the underlying (finite) field $\mathbb{F}_p$.

Given all these definitions, it can be shown that the set of points on an elliptic curve E defined over the field $\mathbb{F}_p$, together with the point at infinity, and the point addition operation form an Abelian group.

3.2.4 Elliptic Curve Discrete Logarithm Problem

After defining groups over points of elliptic curves, it is reasonable to try finding an equivalent of the discrete logarithm problem, as defined in §3.1.3, which can be used in cryptographic algorithms. By re-examining the definition in §3.1.3, it becomes clear how to do so: Given an elliptic curve E defined over $\mathbb{F}_p$, and two points P and Q on E (where P is called the *base point*), finding the scalar k satisfying

$$Q = k \cdot P \tag{3.12}$$

is called the *elliptic curve discrete logarithm problem* (ECDLP). If the order of the base point is sufficiently high, the ECDLP is considered to be computationally infeasible. So far no algorithms with sub-exponential runtime are known to solve the ECDLP. That makes the ECDLP a good basis for cryptographic algorithms. Furthermore due to the complexity of the ECDLP, elliptic-curve cryptography (ECC) achieves the same level of (computational) security as ordinary cryptographic algorithms (based for example on the DLP in groups $\mathbb{Z}_n^*$), with much smaller key sizes and smaller underlying fields. In many applications this significant reduction in size pays off for the more complex operations that must be performed during scalar multiplication.

3.2.5 Further Reading

By today elliptic curves and their application in cryptography have been thoroughly researched. This section only gave an overview of some important concepts and terms.

There are, however, numerous books, papers, lecture notes and theses, which provide a detailed analysis of the topic.

One of the most important text books on elliptic curve cryptography is [HMV03]. It presents mathematic basics, different algorithms for performing point operations (and comparisons between them), applications and protocols, etc.

Readers more interested in pure mathematics should refer to [Wer02] or [FT03]. Whereas readers interested in hardware implementations of ECC should refer to [Wol04], which contains interesting aspects of ECC hardware implementations, as well as a good introduction to the topic, from a hardware designer's point of view.

3.3 Numbers and Digits

When implementing arithmetic operations in hardware, one must find a way to represent the numbers, which form the operands and the results, in a convenient way. Sometimes numbers are too large to fit a particular hardware and must be broken into smaller parts. Since cryptographic operations often involve large numbers, this is of particular importance in this context.

Numbers (and their representations) are needed in almost every system and application, no matter from which domain. General-purpose processors, signal-processors, cryptographic applications, network components, mobile phones, mp3 players,... all use numbers to perform their tasks. Since the use of numbers and their representations is so manifold, the terminology is sometimes ambiguous. For example some refer to a number as "word", and divide it into "digits", while others use the term "word" to refer to a part of a number and call the whole number a "multi-precision value". The following sections are intended to establish a consistent notation and terminology which will then be used throughout the remainder of this work.

Cryptographic operations are (usually) performed on integers. Real numbers and floating-point numbers are not needed. Therefore the considerations will be limited to integers. When it comes to GPS, the scope of the considerations can be narrowed further, because all basic operations can be performed using positive (unsigned) numbers only. However, since point operations on elliptic curves require subtraction and additive inversion (cf. §3.2.2), some basics about negative numbers will be presented as well.

3.3.1 Binary Representation

Unsigned Numbers

Every non-negative integer α, with $0 \leq \alpha < 2^k$ (with $k \in \mathbb{N}$) can be uniquely represented as a sequence of k bits α_i in the following way:

$$\alpha = (\alpha_{k-1}\alpha_{k-2}\alpha_{k-3}\ldots\alpha_2\alpha_1\alpha_0)_2 = \sum_{i=0}^{k-1} \left(2^i \cdot \alpha_i\right), \qquad \alpha_i \in \{0,1\} \tag{3.13}$$

The leftmost bit α_{k-1} is referred to as the *most significant bit* (MSB), and the rightmost bit α_0 is referred to as the *least significant bit* (LSB). The parameter k will hereafter be referred to as the *(bit-)size* of the number α.

Two's Complement

There are many ways of representing negative integers. One which is very widespread is the use of the so-called *two's complement*. The two's complement of a number can be obtained by first inverting all the bits of its binary representation (0 becomes 1, 1 becomes 0), and then adding 1 to the result of the bitwise inversion. With a two's complement system using k bits, numbers α with $-2^{k-1} \leq \alpha < 2^{k-1}$ can be represented. Positive numbers in the range $\left[0, 2^{k-1} - 1\right]$ are represented in usual binary representation, using $(k-1)$ bits. The k-th bit (the leftmost bit) is set to 0. Negative numbers (in the range $\left[-2^{k-1}, -1\right]$) are represented by the two's complement of their absolute value. It is obvious that the leftmost bit of such numbers will always be 1. Therefore, the leftmost bit of such a system is often referred to as *sign bit*, since it determines whether the represented number is zero or positive (sign bit = 0), or negative (sign bit = 1).

Using the two's complement has many advantages: Additive inversion is simple, there is no redundant representation for zero, addition must not distinguish positive and negative numbers, and subtraction can be performed by addition of the additive inverse, to name the most important ones.

Other Representations

For the sake of completeness it should be noted that binary representation is not the only way of encoding numbers. Many other encoding schemes exist. Most of them have been developed for a particular purpose, along with an algorithm to convert back and forth between binary representation and other representations. Since no such representations will be used in the implementations described in this thesis, they will not be discussed any further.

3.3.2 Digits

When numbers become too large they must sometimes be divided into smaller parts, called *digits*. Again, there are many ways how this can be done. Digit sets can for example be redundant or non-redundant. This work will focus on a straightforward, non-redundant approach.

A *radix* 2^d *digit* (with $d \in \mathbb{N}$) is defined as a number in the range $\left[0, 2^d - 1\right]$. Such a digit z can easily be written in binary representation, using d bits: $z = (z_{d-1} z_{d-2} \ldots z_2 z_1 z_0)_2$. Hereafter we will also refer to d as the *digit size*.[2]

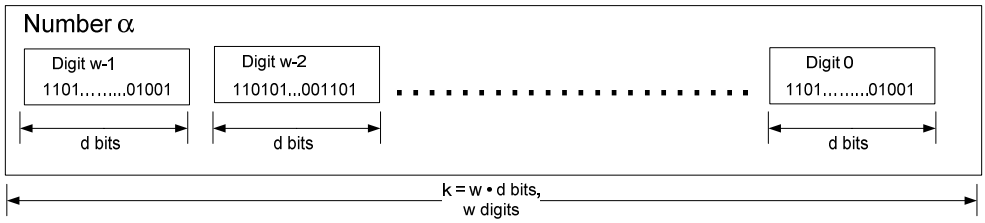

Figure 3.1: *A k-bit number α can be divided into w digits of size d. ($k = w \cdot d$)*

[2]Note that some authors refer to numbers in the range $\left[0, 2^d - 1\right]$ as "radix d" digit, instead of "radix 2^d digit".

Given this definition of a digit, it is straightforward to represent a number as a sequence of digits. First the number is zero-padded on the left-hand side, until the bit-size of the number is a multiple of the digit size. Then the bits of the number are grouped into groups of d bits each. Each d-bit group is then treated as one digit (cf. figure 3.1). The number can then be written as a sequence of digits in the following way:

$$\begin{aligned} &\alpha \in \left[0, 2^k - 1\right], \qquad k, d, w \in \mathbb{N}, \qquad k = w \cdot d \\ &\alpha = \langle \alpha[w-1], \alpha[w-2], \ldots, \alpha[2], \alpha[1], \alpha[0] \rangle = \sum_{i=0}^{w} \left(\alpha[i] \cdot \left(2^d\right)^i \right) \\ &\text{where each digit } \alpha[i] \in \left[0, 2^d - 1\right] \end{aligned} \tag{3.14}$$

The leftmost digit $\alpha[w-1]$ will be denoted *most significant digit* (MSD). The rightmost digit $\alpha[0]$ will be denoted *least significant digit* (LSD). Note that the digits of α are addressed with square brackets ($\alpha[i]$), while the bits of α are addressed with an index (α_i).

Nomenclature

When referring to a number, composed of digits, the terms "word", "full-precision value", "full-precision number", "multi-precision value", and "multi-precision number" will be used synonymously in the remainder of this work.

3.3.3 Multi-Precision Operations

When operating with multi-precision values on digit-level, most arithmetic operations (such as for example addition or multiplication) cannot be performed in one single step. Instead, algorithms performing the operation digit by digit are necessary. To show an example, algorithms for digit-level multi-precision addition and multiplication will be presented. It is important to note that when adding or multiplying single digits, the result may be too large to fit into one digit. Adding two single digits might generate a carry-bit, which must be propagated to the next higher digit. Multiplication of two single digits produces results which consist of two digits. Therefore an accumulator capable of storing two digits is sufficient for performing multiplication and addition. The two digits of the accumulator will be denoted H (most significant) and L (least significant).

Multi-Precision Addition

Algorithm 1: Multi-Precision Addition

Input: Two multi-precision values a, b, each consisting of w digits
Output: A multi-precision value $c = a + b$, consisting of $w + 1$ digits

```
1 H = 0
2 for i = 0 to w − 1 do
3     ⟨H, L⟩ = a[i] + b[i] + H
4     c[i] = L
5 end
6 c[w] = H
7 return c
```

Multi-precision addition can be performed with algorithm 1. First, the most significant digit of the accumulator is set to 0. This digit serves as a "carry-register" and must therefore be cleared in the beginning. Then, within the loop, the corresponding digits of the two multi-precision operands are added. The current value of the carry (H) is also added. The result is stored to the accumulator. The accumulator-digit L will then contain the sum of the two digits of a and b, whereas the accumulator digit H will contain the carry-out of the current addition, which will serve as carry-in for the next iteration. After the loop has terminated the final carry-out is still stored in H. It can either be stored to the w-th digit of the result c (thus making the result one digit longer than the operands a, b), as depicted by algorithm 1, or it can simply be ignored, if the addition is supposed to be performed modulo $2^{w \cdot d}$.

Multi-Precision Multiplication

Algorithm 2: Multi-Precision Multiplication

Input: Two multi-precision values a, b, each consisting of w digits
Output: A multi-precision value $c = a \cdot b$, consisting of $2w$ digits, initially set to 0

```
1 for i = 0 to w − 1 do
2     H = 0
3     for j = 0 to w − 1 do
4         ⟨H, L⟩ = c[i + j] + a[i] · b[j] + H
5         c[i + j] = L
6     end
7     c[i + w] = H
8 end
9 return c
```

Multi-precision multiplication can be done in more than one way. A very straightforward approach is described by algorithm 2.

By means of two nested loops all combinations of digit-products $a[i] \cdot b[j]$ are calculated and accumulated at the correct position of the final result.

3.4 Arithmetic Algorithms

Hardware for performing arithmetic operations is usually only capable of a few primitive operations, such as for example addition, multiplication and shifting. More complex operations, like multiplicative inversion, exponentiation, or modular reduction must be implemented via algorithms composed of primitive operations and control structures (looping, conditional execution,...). This section will present those algorithms, which will be used in later chapters of this work.

3.4.1 Fast Exponentiation – Square & Multiply

Exponentiation, or raising a base b to the power of an exponent e, is defined as repeatedly multiplying the base b with itself, e times (cf. equation 3.15).

$$b^e = \underbrace{b \cdot b \cdot \cdot b \cdot \ldots \cdot b \cdot b}_{e \text{ times}} \tag{3.15}$$

Equation 3.15 is also the simplest solution to calculating an exponentiation: By repeated multiplication, as depicted by algorithm 3.

Algorithm 3: Calculating an Exponentiation by Repeated Multiplication

Input: A base b and an exponent e
Output: A number $r = b^e$

```
1 r = 1
2 for i = 0 to e do
3     r = r · b
4 end
5 return r
```

However, this simple algorithm has a serious drawback: The number of multiplications is equal to the exponent. This linear correspondence prohibits the algorithm from being used with large exponents. Fortunately an improvement exists, which utilizes the binary representation of the exponent e (equation 3.16).

$$b^e = b^{\sum_{i=0}^{k-1} e_i \cdot 2^i} = \prod_{i=0}^{k-1} \left(b^{2^i}\right)^{e_i} \tag{3.16}$$

As it can be seen from equation 3.16, the result of b^e can be obtained by multiplying those b^{2^i}, which have a corresponding $e_i = 1$. The sequence of the b^{2^i} can be obtained by squaring. That leads to the so-called *square-and-multiply algorithm* (algorithm 4). There are two versions of this algorithm. One, which processes the bits of the exponent from LSB to MSB, and one which does it the other way round. The MSB-to-LSB variant has the advantage that it does not need an auxiliary variable. Therefore this variant will be described here: First, squaring is performed in each step. Then the intermediate result is multiplied by the base, or not, depending on the current bit of the exponent.

Algorithm 4: Square & Multiply Algorithm for Calculating an Exponentiation

Input: A base b and a k-bit exponent e
Output: A number $r = b^e$

```
1 r = 1
2 for i = k − 1 downto 0 do
3     r = r · r
4     if e_i = 1 then
5         r = r · b
6     end
7 end
8 return r
```

With this algorithm, the number of multiplications is $2k$ at most, when the exponent consists of k bits. In the average case (assuming an equal distribution of 0-bits and 1-bits in the exponent), the number of multiplications is $(1.5 \cdot k)$. In special cases, where the exponent has only very few 1-bits, the number of multiplications becomes close to k. The latter is frequently the case when performing RSA encryption, where the encryption exponent can be chosen to have very few 1-bits, to speed up the computation. For GPS,

however, the exponent is chosen at random, so the square and multiply algorithm will need about $(1.5 \cdot k)$ multiplications.

It is notable that in any case the number of multiplications is bound by $2k$. And since $k = \lceil ld(e) \rceil$, the number of multiplications grows only logarithmically with e, instead of linearly, as it did with algorithm 3.

Another (slight) improvement is possible: When $e = 0$ is prohibited by preconditions (as it is the case in most cryptographic applications), it is possible to initialize the intermediate result with b instead of 1 (line 1 of algorithm 4). Then the loop counter i is initialized with the index of the highest 1-bit of the exponent e. The precondition $e \neq 0$ ensures that such a bit exists. If the m most significant bits of the exponent are all zero, this improvement saves m unnecessary squarings. This improved version of algorithm 4 is used in all implementations described in §6 of this work.

3.4.2 Scalar Multiplication – Double & Add

Scalar multiplication is the elliptic curve equivalent of exponentiation of numbers in multiplicative groups. Since scalar multiplication is defined as repeated addition (cf. §3.2.2, equation 3.10), the ideas of §3.4.1 can be applied to scalar multiplication as well. This results in the so-called *double-and-add algorithm* to calculate scalar multiplication. A version, incorporating the improvement discussed at the end of §3.4.1, is depicted by algorithm 5.

Algorithm 5: Double & Add Algorithm for Calculating a Scalar Multiplication

Input: A base point P and a k-bit scalar e, with $e \neq 0$
Output: A curve point $Q = e \cdot P$

```
1  Q = R
2  j = k − 1
3  while e_j ≠ 1 do
4      j = j − 1
5  end
6  for i = j − 1 downto 0 do
7      Q = Q + Q
8      if e_i = 1 then
9          Q = Q + P
10     end
11 end
12 return Q
```

3.4.3 Multiplicative Inversion

Finding the multiplicative inverse m^{-1} of a number m (modulo n), satisfying $m \cdot m^{-1} \equiv 1 \pmod{n}$, can be done in multiple ways. A common approach is using the *extended Euclidean algorithm*. The Euclidean algorithm is, however, not very suited for hardware implementation, especially in low-resource devices. A simpler, but probably less efficient

solution can be obtained by utilizing the *theorem of Euler*[3]:

$$a^{\varphi(n)} \equiv 1 \pmod{n}, \qquad \forall a, n \in \mathbb{N} \tag{3.17}$$

where $\varphi(n)$ denotes *Euler's totient function*, also called Euler's φ-function. It is defined as the number of positive integers less than n, which are coprime to n. If n is a prime number p, $\varphi(p) = p - 1$, since all numbers less than p are coprime to p. By rewriting equation 3.17 one can obtain a formula for calculating multiplicative inverses:

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad \Leftrightarrow \quad a \cdot a^{\varphi(n)-1} \equiv 1 \pmod{n} \quad \Rightarrow \quad a^{\varphi(n)-1} = a^{-1} \tag{3.18}$$

As it can be seen from equation 3.18, the multiplicative inverse of a number (modulo n) can be calculated by raising it to the power of $(\varphi(n) - 1)$. For doing so, the square&multiply algorithm, presented in §3.4.1 can be used.

3.4.4 Modular Reduction

When performing arithmetic modulo n, the result of any operation (such as for example addition or multiplication) is not necessarily in the interval $[0, n-1]$. Therefore a modular reduction step is necessary to get a fully reduced result. There are several ways how this can be achieved. Which one of them is preferable depends on the context of the modular reduction and possible preconditions.

Subtraction of Modulus

The easiest, but (in general) also the least efficient way to get a fully reduced result, is to subtract the modulus repeatedly, until a result in the range $[0, n-1]$ is obtained (algorithm 6).

Algorithm 6: Modular Reduction by Repeated Subtraction of Modulus

Input: A number m, a modulus n
Output: A number m mod n in the range $[0, n-1]$

```
1 while m ≥ n do
2     m = m − n
3 end
4 return m
```

This algorithm is obviously suitable only, if $m < k \cdot n$, with k being a small positive integer. Otherwise the number of iterations would be too high. A possible application of this reduction method is reduction after addition. When adding two values which are in the range $[0, n-1]$, the result is in the range $[0, 2n-2]$. Therefore, a fully reduced result is obtained after zero or one subtractions.

In contrast to this, the result of a multiplication is in the range $\left[0, n^2 - 2n + 1\right]$. To obtain a fully reduced result with algorithm 6 up to $n-2$ subtractions could be necessary.

[3]The theorem of Euler is sometimes also referred to as the theorem of Euler-Fermat or the theorem of Fermat-Euler, respectively.

Division

Another way of obtaining fully reduced results is to do a division of the intermediate result by the modulus. Then either the division algorithm is capable of outputting the remainder of the division (which would be the desired fully reduced result), or the quotient is multiplied by the modulus and then subtracted from the non-reduced intermediate result to obtain a fully reduced result. Since division is a very complex and costly operation, this method is not very practical.

Quotient Estimation

Sometimes arithmetic algorithms impose certain conditions on intermediate results that allow to do an estimation of the quotient, without actually performing a division. The corresponding multiple of the modulus can then be subtracted from the intermediate result. Depending on the exact nature of the preconditions, this result may still not be fully reduced, but is guaranteed to be in a range of $[0, k \cdot n - 1]$, where k is a very small positive integer. A fully reduced result can then be obtained by applying algorithm 6.

Special Moduli

For moduli of a special form, reduction might be very easy or even become trivial. If the modulus is a power of 2, for example 2^k, then modular reduction can be performed by ignoring all but the least significant k bits of the intermediate result. This trick can be generalized to moduli which are the sum or difference of only a few different powers of 2. [HMV03] presents detailed algorithms which perform such reductions.

These algorithms are widely used in elliptic-curve cryptography, because the FIPS 186-2 (cf. [NIS00]) standard recommends using curves over fields with so-called *NIST primes* as moduli. These primes satisfy the requirements described in the previous paragraph.

Barret Reduction

Barret reduction is a reduction method that does not impose any restrictions on the modulus. Its details are, however, beyond the scope of this work. It is cited for the sake of completeness only. Interested readers should refer to [HMV03].

Interleaved Reduction

Some arithmetic algorithms are designed to perform interleaved reduction of intermediate results. One such method which is very commonly known is *Montgomery multiplication*. Since Montgomery multiplication is widely used in the implementations described in this work, it will be presented in more detail in the next section.

3.4.5 Montgomery Multiplication

Montgomery multiplication has been introduced by Peter L. Montgomery in [Mon85]. The basic idea is to use the function $MonMul(a, b) = a \cdot b/R \bmod n$, where R is the so-called Montgomery constant, instead of normal multiplication. In order to be able to do so in a meaningful way, a transformation to the so-called *Montgomery domain* is necessary.

The Montgomery Domain

From a mathematical point of view, the Montgomery domain is isomorphic to the integer domain. An element a of the integer domain is mapped to the Montgomery domain by multiplying it with the Montgomery constant R:

$$a' = a \cdot R \pmod{n} \tag{3.19}$$

The Montgomery constant R needs to be larger than the modulus n. Furthermore n and R must be coprime. Usually $R = 2^k$ is chosen for two main reasons: First, the modulus n is usually either a prime number, or a product of a few (large) prime numbers. So choosing R to be a power of 2 ensures that $\gcd(n, R) = 1$. Second, multiplication by R and reduction modulo R are both trivial operations in hardware using binary number representation, for $R = 2^k$.

Equation 3.19 showed how to convert from the integer domain to the Montgomery domain. The inverse transformation is shown by equation 3.20:

$$a = a' \cdot R^{-1} = a'/R \pmod{n} \tag{3.20}$$

Both this transformations can be done utilizing the Montgomery multiplication function $MonMul(a, b) = a \cdot b/R \bmod n$:

$$a' = MonMul(a, R^2) = a \cdot R^2/R = a \cdot R \pmod{n} \tag{3.21}$$

$$a = MonMul(a', 1) = MonMul(a \cdot R, 1) = a \cdot R \cdot 1/R = a \pmod{n} \tag{3.22}$$

This is very convenient, because it means that a system capable of performing $MonMul(a, b)$ does not need any additional resources or capabilities for performing transformations between the two domains.

Arithmetics in the Montgomery Domain

Let a, b, c be numbers of the integer domain, and a', b', c' their corresponding counterparts in the Montgomery domain. Suppose further that $c = a + b$. Then this implies $c' = a' + b'$. The proof is simple:

$$c' = a' + b' = (a \cdot R) + (b \cdot R) = (a + b) \cdot R = c \cdot R \pmod{n} \tag{3.23}$$

The same could be proven for subtraction. So addition and subtraction can be performed in the integer domain or in the Montgomery domain, with the same outcome.

Multiplication, however, is not the same in the integer domain and the Montgomery domain. Let a, b, c again be from the integer domain, but this time $c = a \cdot b$. This time, $c' \neq a' \cdot b'$, when a', b', c' are again the counterparts of a, b, c in the Montgomery domain. This is proved by equation 3.24:

$$c' = c \cdot R = (a \cdot b) \cdot R \neq a \cdot R \cdot b \cdot R = a' \cdot b' \pmod{n} \tag{3.24}$$

The corresponding counterpart of integer multiplication in the Montgomery domain is the Montgomery multiplication function $MonMul(a, b)$. This can be seen from equation 3.25. Let a, b, c be from the integer domain, with $c = a \cdot b$ and a', b', c' from the Montgomery domain, with $c' = MonMul(a', b')$:

$$c' = MonMul(a', b') = a' \cdot b'/R = (a \cdot R) \cdot (b \cdot R)/R = a \cdot b \cdot R^2/R = c \cdot R \pmod{n} \tag{3.25}$$

Usage of Montgomery Multiplication

This section will demonstrate the usefulness of Montgomery multiplication by showing how it can be utilized to calculate $x = g^r \bmod n$, which is one of the basic calculations performed during execution of the GPS authentication scheme. First the base g must be converted to the Montgomery domain:

$$g' = MonMul(g, R^2) = g \cdot R \bmod n \tag{3.26}$$

Then a slightly modified version of algorithm 4, where all integer multiplications have been replaced by calls to $MonMul$, is applied. Finally the intermediate result x' returned from algorithm 4 is converted back to the integer domain:

$$x = MonMul(x', 1) \tag{3.27}$$

The advantage of this approach is obvious: Modular reduction of intermediate results of algorithm 4 is done automatically, because the $MonMul$ function inherently performs (interleaved) reduction. This outweighs the disadvantage of having to convert operands and results between the domains, whenever many multiplications are performed.

Since addition and subtraction can be done in the Montgomery domain without special considerations, it could also be used for ECC point operations. For example, when performing a scalar multiplication (using algorithm 5), the coordinates representing the base point could be converted to the Montgomery domain. Then all field operations necessary for the point operations would be performed in the Montgomery domain, before finally the results obtained for the coordinates of the resulting curve point are converted back to the integer domain.

It should, however, be noted that the role of Montgomery multiplication in ECC is not so crucial, since many ECC applications use NIST primes as moduli, which allow for other fast reduction algorithms (cf. 3.4.4).

Bit-serial Implementation of Montgomery Multiplication

A simple way of implementing Montgomery multiplication in hardware is a bit-serial implementation: One operand is scheduled at full precision, the other one bit by bit. Algorithm 7 depicts such an implementation:

In each iteration operand a is multiplied by one bit of operand b. This so-called *partial product* is then added to the accumulator. After that the reduction step follows: If the intermediate result in the accumulator is odd, the modulus (which is an odd number too) is added. After this conditional addition the accumulator always holds an even intermediate result, which can be divided by 2, by simply shifting it one bit to the right. It is interesting to note that the question, whether the intermediate result is even or odd, can be answered by examining its LSB only. Therefore instead of a conditional addition it is possible to unconditionally add the product of the intermediate result's LSB and the modulus. In this "multiplication" is a simple masking operation. (Further details concerning hardware multipliers and partial product generation will be presented in §5.3.2.)

When the loop is finished, k right-shifts have been performed, which altogether correspond to a division by 2^k. So the accumulator holds the value $a \cdot b/2^k = a \cdot b/R$, which is almost exactly what had been desired. Unfortunately the value of the accumulator upon exit from the loop is not necessarily in the range $[0, n-1]$. When examining the restrictions imposed on the input values it can be deduced that the value is, however, smaller

Algorithm 7: Bit-Serial Implementation of Montgomery Multiplication

Input: $a, b \in [0, n-1]$, odd k-bit modulus n, $R = 2^k$
Output: $c = MonMul(a, b) = a \cdot b/R \bmod n$

```
1  c = 0
2  for i = 0 to k − 1 do
3      c = c + a · b_i
4      c = c + n · c_0
5      c = c/2          // Realized by shifting 1 bit to the right
6  end
7  if c ≥ n then
8      c = c − n
9  end
10 return c
```

than $2n$. Therefore, one conditional subtraction of the modulus suffices to fully reduce the result to the range $[0, n-1]$.

It should be noted that this reduction step is crucial. If it is omitted the resulting value might not be a legal input value for the next $MonMul$ operation. The conditional subtraction, however, complicates hardware implementations and also makes the algorithm vulnerable to side-channel analysis. This problem has been described by S. Gueron in [Gue03], along with an improvement to algorithm 7, which solves the problem: Gueron defines a function called *Non Reduced Montgomery Multiplication of order s* ($NRMM^s$). The difference to classical Montgomery multiplication is that in contrast to the classical $MonMul$, the $NRMM^s$ allows the input values a, b to be greater than then modulus n. Therefore the conditional final subtraction of the modulus can be omitted. The price for this convenience is that the Montgomery constant R must be greater, which also means increasing the size of the underlying hardware.

According to [Gue03] R should be 2^{k+2} instead of 2^k, when the modulus n has k bits. That means that the underlying hardware must be enlarged by two bits. That is, however, a small price to pay, when compared to circuitry for conditional subtraction of the modulus. Another interesting fact pointed out by [Gue03] is that when $NRMM^s$ is used to convert a number from the Montgomery domain back to the integer domain, the result is guaranteed to be in the range $[0, n]$, where n can only occur as result if the original number was a multiple of n. Since this case should never occur in cryptographic applications it is safe to assume that the result of the conversion will be fully reduced to the range $[0, n-1]$. Therefore additional reduction circuitry is not necessary when using $NRMM^s$.

Since $NRMM^s$ is clearly preferable over algorithm 7, the terms "Montgomery multiplication" and "$MonMul$" shall refer to $NRMM^s$ hereafter.

Digit-Level Implementation of Montgomery Multiplication

There are several ways to implement Montgomery multiplication based on digit operations. Five different approaches have been analyzed by Koç et al. in [KAK96]. They differ in how much primitive operations (digit additions, digit multiplications, memory read/write) and how much temporary memory they require. The one which seems most suitable within the scope of this thesis, will be presented in this section. Koç et al. refer to it as *Coarsely*

integrated operand scanning (CIOS). They present two versions of it: A basic version, and a slightly improved version. The latter is the one that is depicted in algorithm 8.

Algorithm 8: Digit-Level Implementation of Montgomery Multiplication — Coarsely Integrated Operand Scanning, from [KAK96]

Input: Operands a, b, consisting of w d-bit digits each; a modulus n, consisting of w d-bit digits

Output: A number $t = MonMul(a, b)$, consisting of $(w + 1)$ d-bit digits

```
1  for i = 0 to w − 1 do
2      H = 0
3      for j = 0 to w − 1 do
4          ⟨H, L⟩ = t[j] + a[j] · b[i] + H
5          t[j] = L
6      end
7      ⟨H, L⟩ = t[w] + H
8      t[w] = L
9      t[w + 1] = H
10     H = 0
11     m = t[0] · n′[0] mod 2^d
12     ⟨H, L⟩ = t[0] + m · n[0]
13     for j = 1 to w − 1 do
14         ⟨H, L⟩ = t[j] + m · n[j] + H
15         t[j − 1] = L
16     end
17     ⟨H, L⟩ = t[w] + H
18     t[w − 1] = L
19     t[w] = t[w + 1] + H
20 end
21 return t
```

The digit $n'[0]$ in line 11 of algorithm 8 is a modulus dependant constant. [KAK96] defines it "as the inverse of the least significant word[4] of n modulo 2^w. That is, $n'[0] = -n[0]^{-1} \pmod{2^w}$." This constant is either precomputed and stored, or it can be calculated by means of a simple algorithm presented in [DK90]. When looking at the GPS authentication scheme, the modulus of a specific prover does not change. Therefore it makes sense to precompute and store $n'[0]$.

A few words about the working principle of algorithm 8: Lines 2 – 9 accumulate a partial product, where all digits of one operand are multiplied with one digit of the other operand. In lines 11 – 19 an interleaved reduction step takes place. A multiple of the modulus is added, and the whole intermediate result is shifted one digit to the right. For further details refer to [KAK96].

It should be noted that algorithm 8 does not incorporate the improvements suggested by [Gue03]. This can, however, be changed very easily, by simply choosing the number of digits w in a way that the actual operands only occupy $(w - 1)$ digits. By doing so, algorithm 8 acquires all the advantages of $NRMM^s$, as described in [Gue03].

[4]Note that [KAK96] uses the term "word" for what has been defined as a "digit" in the context of this thesis.

Chapter 4

The GPS Authentication Scheme

Now that the basic concepts of authentication and the mathematics behind them have been revisited, this chapter will present details about the GPS authentication scheme[1] and different variants of it.

To avoid any misunderstandings, it should be stated explicitly that the GPS authentication scheme has nothing to do with the well-known *global positioning system*, used for satellite-based navigation. The GPS authentication scheme owes its name to Marc Girault, Guillaume Poupard, and Jacques Stern. Whether the acronym was deliberately chosen to equal the one of the global positioning system, or whether this is just a funny coincidence remains unknown.

4.1 Basic Algorithm

GPS is an entity authentication scheme, which is of particular interest in the scope of application scenarios where prover devices only have limited computational power, but the corresponding verifier devices are not bound by the same limitations. RFID systems are an excellent example of such a scenario. Tags only have very limited resources, however reader devices have computational power which is several orders of magnitude larger. Therefore RFID systems will be used as illustrating examples throughout this section.

Hereafter it will implicitly be assumed that prover devices only have limited computational power, while verifier devices are not constricted in the same way. Recall that the physical definition of power is "work per time". Translating this to computer science implies that "limited computational power" means that only a certain amount of computations can be performed within a given time interval. Therefore a device with limited computational power is not necessarily completely incapable of performing complex operations. It would just need (much) more time to complete such an operation.

4.1.1 History

The exact creation history of the GPS scheme is somewhat difficult to track. At the EUROCRYPT'90 conference Marc Girault presented a version of the Schnorr scheme, using a composite (RSA-like) modulus instead of a prime one (cf. [Gir91]). One year later at EUROCRYPT'91 Girault presented further details, including the utilization of what he called *self-certified public keys* (cf. [Gir92]).

[1] Most papers and literature sources use the term "GPS identification protocol". Nevertheless adhering to the distinction made in §2.1.2 the term "authentication" will be used throughout this work.

Several years later Guillaume Poupard and Jacques Stern proved the security and analyzed the properties of the scheme proposed by Girault. Their findings are summarized in [PS98], presented at EUROCRYPT'98.

The name GPS seems to have appeared first in the final report of the NESSIE project (cf. [NES04]) in 2004. In the same year GPS has been standardized by ISO/IEC 9798-5 (cf. [ISO04]).

4.1.2 The Protocol and its Parameters

GPS is a zero-knowledge authentication scheme. An entity convinces a verifier of its identity by interactively proving knowledge of the discrete logarithm of a public key, which is associated with the entity. Therefore the security of the scheme is based on the discrete logarithm problem. Figure 4.1 depicts the basic GPS protocol and its parameters.

Prover		**Verifier**
	Domain Parameters	
Modulus $n = p \cdot q$ (factorization possibly known)		Modulus $n = p \cdot q$ (factorization unknown)
Base g		Base g
	Keys	
Private Key $s:\ 0 \leq s < 2^{\sigma}$ Public Key $v = g^s \bmod n$		Public Key $v = g^s \bmod n$
	Protocol	
Choose r at random, $0 \leq r < 2^{\rho}$ Calculate $x = \texttt{hash}\,(g^r \bmod n)$	$\xrightarrow{\quad x \quad}$	
	$\xleftarrow{\quad c \quad}$	Choose challenge c $0 \leq c < 2^{\delta}$
Calculate $y = r - (s \cdot c)$	$\xrightarrow{\quad y \quad}$	
		Check $x \stackrel{?}{=} \texttt{hash}\,((g^y \cdot v^c) \bmod n)$, if *yes*, accept prover; if *no*, reject prover.

Figure 4.1: *Sketch of the basic GPS protocol, as standardized by [ISO04]. The hash function is optional and the subtraction in the last calculation of the prover can be substituted by addition, if the public key is changed to $v = g^{-s} \bmod n$.*

Domain Parameters

There are two domain parameters: A base g and a modulus n, which is the product of two large prime numbers p and q. The base g might be shared by different users without further considerations. It generates a cyclic multiplicative subgroup of $\mathbb{Z}_n^*$, which has an unknown order. For the sake of simplicity $g = 2$ is suggested frequently in literature. However, g could also be chosen at random.

The modulus should be chosen consistently with all guidelines and recommendations for RSA moduli. The bit-size of the modulus should be chosen according to the security requirements. A value which is used extensively in related literature is 1024 bits.

It should be noted that the factorization of the modulus must remain secret. According to [PS98] knowing the factorization of n facilitates using the *Chinese remainder theorem* (CRT) to calculate discrete logarithms by calculating them modulo the prime factors p, q instead of modulo n. That would significantly decrease the computational power which would be required to break the scheme and hence significantly decrease the (computational) security. Therefore the factorization of n must not be known to anyone except the prover and/or a trusted authority.

An interesting observation is that unlike the RSA encryption/signature scheme, GPS allows several users to use the same modulus n. This is due to the fact that GPS does not require the prover to know the factorization of n, neither for protocol execution, nor for determining the public key associated with a (randomly chosen) secret key. So a modulus n could be generated by a trusted authority for an entire domain of users. Each user would be capable of choosing its own private key and calculating the corresponding public key without knowing the factorization of n. The trusted authority would never have to reveal the factors of n to anyone.

This scenario, however, suffers from the drawback that provers would not be able to use the Chinese remainder theorem (CRT) to speed up their computations, because for doing so knowledge of the factors p, q of n is necessary. If each prover generated its own modulus, with factorization only known to that particular user, then this would enable this user to utilize the CRT to speed up the computation of commitments.

There is yet a third scenario. Recall the idea of fighting product piracy by incorporating cryptography-enhanced RFID tags into products, which was presented in §1.1.2. Now suppose a company manufactures many different products and incorporates GPS-capable RFID tags into these products to prove their genuineness. Then this company could use the same modulus n for all products, although each product would use its own key pair. Since it is reasonable to assume that one product of that company will never try to impersonate another one, it would be possible to store the factorization of n in the tags, so that they could use the CRT to increase their performance. Of course this reasoning only holds if it is impossible to extract the factorization from the tags. Usually this can be assumed, because if extracting information from tags was possible, for example by means of some sort of physical intrusion and examination, then it would also be possible to extract secret keys and thereby break *any* scheme. But still, extracting one private key would enable an attacker to impersonate one entity; the one the key belonged to. However extracting the factorization of n and subsequently being able to calculate discrete logarithms modulo n would allow the attacker to impersonate any entity which uses the same modulus n. That would make costly extraction attempts more likely to pay off for the attacker. Therefore depending on the context of the application it should be investigated whether computation speed-ups resulting from utilizing the CRT really pay off for this additional threat.

The Keys

The private key s of an entity is a number with $0 \leq s < 2^\sigma$, which should be chosen at random. The corresponding public key is $v = g^s \bmod n$. Therefore the private key is the discrete logarithm of the public key with respect to base g. As long as the discrete logarithm problem is a hard problem, it is infeasible to determine the private key from knowing the public key only.

The choice of the parameter σ depends on the desired security. A common value, often suggested by literature, is $\sigma = 160$.

Other Parameters

There are two more parameters: The number of bits of the random value which is used for calculating the commitment and the number of bits of the challenge. The first is denoted ρ, the latter is denoted δ. Literature suggests $\delta = 20$ and $\rho = \delta + \sigma + 80 = 260$ as suitable values.

Protocol Run

Given all these parameters and keys a protocol run works as follows (cf. also figure 4.1): First the prover selects a value r at random, calculates the commitment $x = \texttt{hash}\,(g^r \bmod n)$, and sends x to the verifier. After the verifier has received the prover's commitment, the verifier selects a challenge c at random and sends it to the prover. Then the prover calculates the response $y = r - (s \cdot c)$ and sends it to the verifier. The verifier then accepts the prover if and only if $\texttt{hash}\,((g^y \cdot v^c) \bmod n) = x$.

Remarks

It should be noted that according to the specification in [ISO04] the use of the hash function is optional. However, when used, there are four different modes how to compute the hash of the number $g^r \bmod n$ together with an (optional) text field. The text field can be filled with some string or message, which is used for application-specific protocol extensions. Some other interesting details concerning the hash function will be presented in §4.4.

Another thing worth mentioning is the calculation of the response in the last step. [ISO04] specifies $y = r-(s\cdot c)$, whereas [NES04] presents a slightly different form, replacing the subtraction with addition: $y = r+(s\cdot c)$. In the latter case the public key $v = g^s \bmod n$ must be replaced with $v = g^{-s} \bmod n$. Since integer addition is easier to achieve in hardware than subtraction, all implementations discussed in §6 will employ the version with addition, as outlined by [NES04].

Also note that the multiplication and the addition in the last step are *not* done modulo n. These two operations are ordinary integer operations without modular reduction. However, if all the parameters are chosen as outlined in this section, then $y \ll n$ and the result would be the same, no matter whether modular or non-modular arithmetic was used.

4.1.3 Notation

When referring to the GPS algorithm or its parameters the symbols shown in table 4.1 will be used throughout the remainder of this thesis. Table 4.1 also presents typical values,

frequently suggested by various literature sources (cf. for example [PS98], [MR07b], or [MR07a]), for the parameters.

Meaning	**Symbol**	**Typical**
modulus	n	1024 bit
factors of modulus	p, q	512 bit each
base	g	2
private key	s	—
Number of bits of the private key	σ	160
public key	$v = g^s \bmod n$, or $v = g^{-s} \bmod n$	—
random value	r	—
Number of bits of the random value r	ρ	260
hash function	`hash`()	—
commitment	$x = g^r \bmod n$, or $x = \texttt{hash}(g^r \bmod n)$	—
challenge	c	—
Number of bits of the challenge	δ	20
response	$y = r - (s \cdot c)$, or $y = r + (s \cdot c)$	—

Table 4.1: *Symbols used to describe parameters and variables of the GPS protocol, and typical values for the parameters, as suggested by literature.*

4.2 Coupon Approaches

As it has been said before, in many applications it must be assumed that the computational power of a prover is limited. For example it seems unlikely that a passive RFID tag would be capable of performing number-theoretic operations such as modular exponentiations of quite large operands; at least within a short time. Short transaction times are, however, important for most applications. Imagine an access control system for public transportation. Each person would carry some object with a tag which must authenticate against a reader when the person wants to pass an access gate. When the person approaches the gate, there is very little time for the authentication process to complete. If it takes too long, the gate cannot be opened in time and the person would have to stop and wait for the process to complete. This is very annoying, and could lead to significant congestion at the access gates during peak times. Therefore this should be prevented to ensure the usability and acceptance of the system.

Zero-knowledge protocols provide an interesting option of decreasing the time needed to perform authentication. The commitment does not depend on any input from the verifier. Therefore there is no reason why a prover should wait until the start of the authentication process to choose a commitment. Commitments can be precomputed and stored by the prover in terms of so-called *coupons*. Three different approaches how coupons can be integrated into a GPS-based authentication system will be described and analyzed

in this section.

4.2.1 Complete Coupons

GPS allows two computations to be done in advance: Choosing a random value r and calculating the corresponding commitment $x = g^r \bmod n$. A set of coupons consisting of value pairs (r_i, x_i) could be precomputed by a (trusted) third party and stored to the prover device at manufacturing time. Since such coupons (r_i, x_i) utilize the maximum amount of precomputation, they will be referred to as *complete coupons*.

Advantages

Complete coupons do away with the necessity to implement a (pseudo-)random number generator (PRNG) on the prover device, and also avoid the complex operation of modular exponentiation. The only operations left to the prover are one integer multiplication and one integer addition which are necessary to calculate the response $y = r + (s \cdot c)$. These are non-modular integer operations, so there is not even the need to store the modulus on the prover's side, if it can be assumed that the verifier will know it (for example because it is a domain-wide parameter). Since the modulus usually is much larger than the other operands (r, s, c), this saves a significant amount of ROM or NVRAM.

To further decrease the necessary storage space, a hash function can be used (cf. figure 4.1) to decrease the size of the x_i. Without hashing, the x_i have the same number of bits as the modulus has, for example 1024. If a hash function was used, they would only need as many bits as the hash function outputs, for example 256 or even less. Since the commitments are computed on a computationally powerful device anyway, tags do not even need to know anything about the hash function and the on-tag circuit would not be complicated by the use of a hash function.

Application Scenario

Coupon-based systems could be employed in applications where the total number of authentications which a prover must perform during its lifetime is inherently limited. For example RFID tags used to track a product's way from the manufacturer, via (several) logistics companies, to a shop, and finally the end customer. The number of times the tag would be queried by readers before the corresponding product is sold to an end customer is limited. After the product has been sold there is no need for the tag to be operable any more. In fact many product-related tags are "killed" at the point of sale anyway, due to privacy concerns.

In such a scenario it would be possible to estimate the number of necessary authentications in advance, maybe adding a safety margin, and store an according number of coupons to the tag at manufacturing time.

Disadvantages

To build such a system in an economic way it must be possible to predict the number of authentications which a tag must be able to perform very precisely. If the estimation was too low the tag would stop being operable too early. However, if the estimation was too high, memory resources (and thereby money) would be wasted.

Another serious disadvantage of such a system is the fact that it enables *denial-of-service attacks* (DoS attacks). A tag cannot determine whether an authentication request from a reader is warrantable or not. Therefore if a tag contained k coupons an attacker could disable the tag by commanding it to authenticate k times.

Thinking of a supermarket in which every single product was equipped with such a tag, and assuming widespread deployment of RFID reader devices in mobile phones and pocket computers, it is very likely that someone would try to mount such an attack. Maybe even just because it is "fun". Therefore this attack scenario imposes narrow constraints on the possible fields of application for coupon-based systems: They are only applicable, if the aforementioned DoS attack scenario can be neglected due to application-specific circumstances and assumptions.

4.2.2 Partial Coupons

If an efficient pseudo-random number generator (PRNG) implementation is used, it might be preferable to store only the commitments x_i and the seed for the PRNG, instead of complete pairs (r_i, x_i). Such coupons will be referred to as *partial coupons*. They might be of special interest when memory costs are relatively high. Each partial coupons saves ρ bits of memory, compared to a complete coupon. This approach is particularly interesting when ρ is very large: For example, the low-Hamming-weight variant which will be discussed in §4.5 utilizes random values r with several hundreds of bits. When employing the partial-coupon approach to this variant, a significant amount of memory can be saved at the cost of adding a PRNG.

Advantages and Disadvantages

The main advantages and disadvantages, including the DoS-attack scenario, are the same as with complete coupons, outlined in §4.2.1. Apart from that authentication now needs a little more time, because the random number r must be generated on the fly before the response can be calculated.

4.2.3 Recalculating Coupons

As it has been said before, "limited computing power" does not mean that complex operations are completely impossible; it just means that such operations require much more time. A passive tag might not be capable of doing a modular exponentiation within the short time that an authentication protocol run may take. However, this might not be necessary.

Consider the following scenario: Tags are equipped with k precomputed (complete) coupons at production time. When a tag needs to authenticate it can use one of these coupons to perform fast and efficient authentication. However, in many applications tags are in the range of a reader field even when they are not required to authenticate. They might just happen to pass by; or they remain in the field longer than would be necessary for the authentication. Whenever a tag is in the range of a reader field, it is powered. The outside world does not care whether the tag uses this power to do something meaningful, or whether it just stays powered but inactive. Therefore it stands to reason that tags use this time to compute new coupons, when they have already used up some.

These computations would not be part of an active authentication process and therefore would not be bound by fierce timing constraints. In fact, depending on the application,

the time necessary to complete the operation is of little to no interest at all.

Two things should be investigated for such systems: The estimated number of authentications (per time) a tag must perform, and the estimated total time the tag is powered by a reader field in between authentications. These two factors determine to what extent the time to complete one coupon-recalculation is essential. The system can only work properly if (in average) in a given time interval at least as many coupons can be recalculated as are used up. Furthermore if the time during which tags are powered is rather short, it would be beneficial if the recalculation operations would store intermediate results to non-volatile memory, to resume calculation from that point at a later time, if power supply is lost during the recalculation process. Otherwise a tag might never be able to complete a recalculation if it is never powered continuously long enough.

Advantages and Disadvantages

The coupon-recalculation approach has all the advantages of the complete-coupon approach, but not all its disadvantages. As long as coupons are available authentication is as fast and efficient as in the complete-coupon approach. However, the DoS-attack scenario outlined above does not apply here. Tags cannot be completely disabled by (useless) authentication requests. The worst thing that could happen is that no more coupons are available, and that an authentication has to be delayed until the next coupon has been recalculated. This makes the DoS attack less "attractive" in many scenarios. Granted, it is not completely eliminated yet. Systems where the delay of one or a few single authentications causes "damage" are still vulnerable to the DoS attack, however systems where occasional delays are acceptable cannot be harmed so easily any more.

There is yet another interesting observation: In many application scenarios multiple tags are queried "simultaneously"[2]. For example when putting a pallet of goods to stock, a reader at the entrance of the stockroom would interrogate all tags attached to the goods. Since all tags are powered simultaneously they are all able to recalculate coupons simultaneously. Therefore the maximum delay due to coupon recalculation is constant and not depending on the number of tags which are currently out of coupons.

Implementing the ability to recalculate coupons certainly requires some additional hardware resources. However, since execution time is not an important factor for that part of the circuit, it is much more likely that it can be implemented in an area- and power-efficient manner, as if it was bound by timing constraints.

Further Remarks

There is yet another possibility to implement the coupon-recalculation approach in practice. Suppose a smartcard, with both a contactless and a contact-based interface. The contactless interface could be used for coupon-based authentication while coupons are available. After some (or all) coupons have been used up, the card could be put into a contact-based smartcard reader, to supply it with much more power than would be possible via the contactless interface. The card could use this power to perform a rather quick recalculation of coupons. One possible application for such a system is using such cards as public transportation tickets. The card could be "charged" with coupons in the morning, by means of the contact-based interface and then used for contactless authentication

[2] To be exact, the tags are queried one after another, but within a very short overall time. Furthermore all tags are powered during the entire process.

throughout the entire day. If contact-based card slots were made available at access gates, the worst thing that could happen if the card ran out of coupons is that the card holder would have to grab it and put it in the slot, instead of just carrying it and relying on contactless authentication.

Since the coupon-recalculation approach seems to be a very suitable compromise, from an application's point of view, a significant part of the design-space exploration presented in §6 will focus on finding a suitable implementation of coupon recalculation. To the best of the author's knowledge this is the first time this approach is applied to the GPS authentication scheme.

4.3 ECC-based Variant

The security of the GPS scheme is based on the discrete logarithm problem. A security proof (and proof for some other properties, such as the zero-knowledge property) can be found in [PS98]. However, the scheme is not limited to utilizing the multiplicative group $\mathbb{Z}_n^*$. Any group where the discrete logarithm problem is a hard problem can be used. The first such group that comes to a cryptographer's mind is an elliptic-curve-based group.

When pursuing a complete or partial coupon approach, virtually no changes to the prover are necessary. The ECC operations are all done during precomputation. The calculation of the response in the last step of the protocol remains unchanged. Figure 4.2 shows how elliptic-curve cryptography can be used in the GPS authentication scheme.

The advantage of the ECC-based variant over the $\mathbb{Z}_n^*$-based variant is that ECC achieves the same level of security with much lower bit-sizes. Therefore when using complete or partial coupons a considerable amount of memory can be saved, even without the need for a hash function: In the $\mathbb{Z}_n^*$-based variant the commitment x is the base g, raised to the power r, reduced modulo n: $x = g^r \bmod n$. Therefore x can have as many bits as n (= 1024 bits, to use typical values). In the ECC-based variant the commitment x is a scalar multiple of the base point P. It can be represented by two (affine) or three (projective) coordinates (cf. §3.2.2). Each coordinate is an element of the underlying finite field. Suppose the underlying field's characteristic has 160 bits (which is considered adequate security by [GJR07]). Then each commitment x_i would only consist of $2 \cdot 160 = 320$ bits (or $3 \cdot 160 = 480$ bits, when using projective coordinates) instead of 1024 bits, as it would in the $\mathbb{Z}_n^*$-based variant.

When using a coupon-recalculation approach ECC operations might be preferable over $\mathbb{Z}_n^*$ operations because they have smaller operands. Therefore fewer bits would have to be manipulated at a time, which could lead to an even more power-efficient implementation, if execution time is again left out of consideration. However, it stands to reason that an ECC-based implementation would require more area resources since more different (field) operations are required for ECC than just for modular exponentiation in $\mathbb{Z}_n^*$. However, this thesis will not do such estimations. Some further notes about this issue will be presented in §7.2.1, but apart from that investigating ECC variants will be left to future research.

4.4 Aspects of the Hash Function

As it has already been said, the hash function in the GPS protocol, as it is shown in figure 4.1, is optional according to [ISO04]. From that one can conclude that its presence is

Prover		**Verifier**
	Domain Parameters	
Elliptic Curve E		Elliptic Curve E
Base Point P		Base Point P
	Keys	
Private Key $s:\ 0 \leq s < 2^{\sigma}$ Public Key $V = s \cdot P$		Public Key $V = s \cdot P$
	Precomputation	
Repeat for all i with $0 \leq i < \tau$: Choose r_i at random, $0 \leq r_i < 2^{\rho}$ Calculate $x_i = r_i \cdot P$ Store coupon (r_i, x_i)		
	Protocol	
Select coupon (r_i, x_i)	$x_i \longrightarrow$	
		Choose challenge c $0 \leq c < 2^{\delta}$
	$\longleftarrow c$	
Calculate $y = r_i - (s \cdot c)$	$y \longrightarrow$	
		Check $x_i \stackrel{?}{=} (y \cdot P + c \cdot V)$, if *yes*, accept prover; if *no*, reject prover.

Figure 4.2: *Sketch of an ECC-based GPS variant, which uses τ precomputed coupons and no hash function.*

obviously not crucial to the security of the scheme, as it is the case in some other schemes. So the obvious question is: What is the actual purpose of the hash function?

4.4.1 Purpose of the Hash Function

[ISO04] suggests that an additional text field might be included in the input of the hash function. This text field could be used for application-specific protocol extensions. Certainly, that would be one useful purpose, if such extensions were required.

According to information obtained from [Gir07] the main purpose of the hash function is to significantly decrease the storage requirements of coupons and to minimize the number of bits which must be transmitted between prover and verifier during authentication. That raises the question whether the hash function must really satisfy all the usual requirements for cryptographic hash functions, or whether something "weaker" would also suffice.

The reason why this question is interesting is that most standardized hash functions

require considerable resources, when implemented in hardware (cf. [FR06]). On the other hand, hash functions which are specifically tailored towards low-resource hardware implementations might not fulfil all usual cryptographic requirements. Therefore the next section will investigate whether or not such "weaker" hash functions could be (securely) employed in the GPS authentication scheme, in order to save hardware resources.

4.4.2 Using "Weaker" Hash Functions

As of today there are many standardized hash functions, for example the functions from the SHA family. Their properties have been carefully and thoroughly analyzed by the scientific community to determine whether it is secure to use them in cryptographic protocols, or not.

Properties of Hash Functions

There are three important properties, which a hash function should fulfil in order to qualify for use in cryptographic operations (cf. [MvOV97]):

1. *preimage resistance:* It is infeasible to find an x' satisfying $x = \texttt{hash}(x')$, when given x only.
2. 2^{nd} *preimage resistance:* When given x, it is infeasible to find an $x' \neq x$ satisfying $\texttt{hash}(x') = \texttt{hash}(x)$. For almost all cases this implies preimage resistance.
3. *collision resistance:* It is infeasible to find any pair $x, x' \neq x$ satisfying $\texttt{hash}(x') = \texttt{hash}(x)$. Collision resistance implies 2^{nd} preimage resistance.

Note that the exact definition of "infeasible" in this context depends on the security requirements of the application.

The problem is that most standardized hash functions require considerable resources when implemented in hardware; at least in the context of passive contactless devices. Feldhofer and Rechberger have analyzed this situation in [FR06]. They conclude that commonly known hash functions, such as SHA-1, SHA-256, MD5, and MD4, are less suited for application in RFID tags than other cryptographic primitives.

Preimage Resistance in the Context of GPS

Most cryptographic primitives which employ hash functions absolutely require the three properties outlined above. A hash function which is not preimage resistant could not be used in most primitives without immediately compromising security. GPS could be an exception to this rule. This section will outline an idea which leads to the belief that preimage resistance might not be a necessary requirement for the hash function used in the GPS protocol. However, it shall be made clear that this is not at all a proof in a mathematical sense, but just an outline of a thought.

Suppose it would be possible to determine preimages of the hash function used in the GPS protocol. The question is whether this would help an attacker, or not. Recall the verification step of the protocol (cf. figure 4.1): After having received y, the verifier checks whether $\texttt{hash}(z) \stackrel{?}{=} x$, where $z = (g^y \cdot v^c) \bmod n$. Determining y necessitates knowing the secret key. However, an attacker could break the scheme, if it is possible to find a y', which satisfies $\texttt{hash}(z') = x$, where $z' = (g^{y'} \cdot v^c) \bmod n$, without knowing the secret key.

If the hash function is not preimage resistant, determining z' from x, which is known to the attacker, is feasible, since z' is per definition the preimage of x. However knowing z' does not help the attacker very much. The input to the verification step must be a y', satisfying $z' = (g^{y'} \cdot v^c) \bmod n$. When z' is known, determining y' from this equation still means solving the discrete logarithm problem modulo n, which is considered infeasible. For the same reason it would not be possible to determine the random value r from $x = \texttt{hash}(g^r \bmod n)$, even if determining the preimage of the hash function was feasible.

So although a formal proof is missing, this idea may indicate that hash functions in the scope of GPS maybe do not necessarily need to fulfil (all) the usual requirements. If so, maybe it would be possible to construct hash functions which are specifically tailored towards GPS, which could allow more efficient implementations as a result of "weaker" properties. However, proving this hypothesis is beyond the scope of this thesis.

4.4.3 k-Collision-Free Hash Functions

Girault and Stern have analyzed the hash function's influence on the security level of several zero-knowledge authentication schemes in [GS94]. Unfortunately they do not consider the GPS protocol in [GS94], but several other protocols including Schnorr's scheme which is quite similar to GPS. Their findings are somewhat weird at first glance. First they show that under certain conditions the use of hash functions can decrease the security level of authentication protocols, if the hash functions were not collision resistant. They subsequently prove that using collision-resistant hash functions is sufficient to get a security level equal to the expected one. However, although collision resistance is sufficient it is not necessarily required. For certain schemes, like GPS, the collision-resistance property can be relaxed without compromising the security level, if the (bit-)length of the challenge is increased in return.

Girault and Stern define so-called *k-collision-free hash functions*.[3] They refer to k pairwise distinct values $x_1, x_2, \ldots, x_k$ which satisfy $\texttt{hash}(x_1) = \texttt{hash}(x_2) = \ldots = \texttt{hash}(x_k)$ as a k-collision of the hash function. A k-collision-free hash function is a hash function for which it is computationally infeasible to find a k-collision, but finding $(k-1)$-collisions might be possible. By this definition a usual (collision-resistant) hash function would be referred to as 2-collision free. Girault and Stern then show by how many bits the challenge must increased to keep the level of security, when a k-collision-free hash function ($k > 2$) is used.

Application to GPS

Later Girault applied the findings of [GS94] to the GPS scheme and did some further analysis. His results are published in [Gir00]. That paper explains how (k-)collisions of the hash function interfere with the security of the protocol: Without hash functions an attacker can only follow the strategy which was indicated in the story in §2.4.1. The attacker could (randomly) guess the challenge c and choose the commitment accordingly. In GPS that means computing $x = (g^y \cdot v^c) \bmod n$, where the attacker picks y at random. If the attacker's guess of c was correct, the response y will be accepted by the verifier. However, the probability that the guess was right is only $2^{-\sigma}$, if the challenge has σ bits.

[3] Actually Girault and Stern refer to these functions as r-collision-free hash functions, but since in this work the letter r is used for the random value chosen in step 1 of the GPS protocol, the term k-collision-free will be used to avoid confusion.

If a k-collision-free hash function (with $k > 2$) was used, an attacker would have additional opportunities. Basically, suppose an attacker guesses values c_1, c_2 and selects values y_1, y_2 at random. Suppose further that the attacker would be able to find these values in such a way that the corresponding values $x_1 = (g^{y_1} \cdot v_1^c) \bmod n$ and $x_2 = (g^{y_2} \cdot v_2^c) \bmod n$ are a 2-collision of the hash function. That means that $\texttt{hash}(x_1) = \texttt{hash}(x_2) = x'$. Now if the attacker used x' as commitment there are now *two* possible challenges (c_1, c_2) which the attacker would be able to answer. Therefore the probability of impersonation increases to $2^{\sigma-1}$. However, this decrease in security can be compensated by making the challenge one bit longer.

This idea can be generalized to k-collisions ($k > 2$) in a straightforward way. This section only aimed at presenting the general idea behind showing the influence of k-collision-free hash functions on the security level. A detailed formal analysis of this observation can be found in [Gir00]. Therein Girault also considers "online" attacks, which only start after the challenge has been issued. That gives the attacker further possibilities on one hand, but on the other hand this also limits the attacker's time, and thereby the number of computations the attacker can perform.

4.5 Low-Hamming-Weight Variant

Yet another interesting variant of the GPS authentication scheme was proposed by Girault and Lefranc in [GL04]. They suggest using challenges with a low Hamming weight. Usually challenges are chosen (randomly) from the interval $\left[0, 2^\delta - 1\right]$. However, according to [GL04] any other subset of $\mathbb{Z}^+$ of the same cardinality is also suitable. Therefore it is suggested to use challenges where 1-bits are separated by at least $(\sigma - 1)$ 0-bits, where σ is the (bit-)size of the private key s. Furthermore the challenge should have a low Hamming weight, which means that the total number of 1-bits should be low; [GL04] suggests for example five 1-bits.

That effectively does away with the multiplication $(s \cdot c)$ in the response calculation step. Since the 1-bits in c are at least $(\sigma - 1)$ bits away from each other, determining the product $(s \cdot c)$ reduces to aligning "copies" of s with the positions of the 1-bits in c. Therefore the calculation of the response in the last step of the GPS protocol reduces to a single addition of r with the aforementioned aligned copies of s.

This approach necessitates increasing the total number of bits of the challenge c significantly. However, since the challenge has a low Hamming weight, efficient ways to compress it can be found. One such way is suggested by McLoone and Robshaw in [MR07a]. They have found a way to encode a challenge with 848 bits and a Hamming weight of 5, using 40 bits (5 bytes) only. Their solution is based on encoding only the positions of the five 1-bits.

It should be noted that according to [GL04] increasing the size of the product $(s \cdot c)$ necessitates increasing the size of the random value r too. Otherwise the zero-knowledge property would be lost. Girault and Lefranc suggest that $\rho = \sigma + \delta + 80$, where ρ, σ, δ are the number of bits of r, s, c, respectively.

4.6 State-of-the-Art

Very few work has been published on implementations of the GPS authentication scheme. The author of this work is not aware of any publications on software implementations

of GPS. Concerning hardware implementations, McLoone and Robshaw presented an architecture for GPS in [MR07b] and [MR07a]. Girault et al. presented a working FPGA prototype in [GJR07]. Some details about these findings and a comparison with results of this work will be presented in §7.1.1.

4.7 Further Reading

A thorough analysis of the GPS authentication scheme, including comparisons with similar schemes, is given in [GPS06]. Mathematic properties and background information are given, along with security considerations and references to almost all other literature about GPS.

Chapter 5

Hardware Aspects

This chapter will present some basic facts, which are of interest when designing and implementing *application specific integrated circuits* (ASICs). It will focus only on those concepts which are relevant to the implementations which will be presented in §6. References to literature containing more detailed information will be given along the way. A summary of relevant literature references is given in §5.6.

5.1 CMOS — Complementary Metal-Oxide-Semiconductor

Today the vast majority of digital ASICs is implemented in *CMOS technology*. CMOS is the acronym for *complementary metal-oxide-semiconductor*. It uses both n-type and p-type metal-oxide-semiconductor field-effect transistors (MOSFETs) to efficiently construct logic circuits.

Details about the physical principles of MOSFETs are beyond the scope of this work. Interested readers are referred to [TS02] and [Raz02].

5.1.1 Basic Principles

The basic principles of CMOS technology are best explained using the simplest example: The CMOS inverter. Figure 5.1 shows a schematic of a CMOS inverter. The inverter

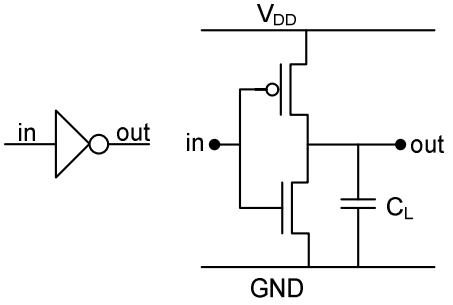

Figure 5.1: *Symbol and Schematic of a CMOS Inverter*

consists of a PMOS and an NMOS transistor in series between the supply voltage V_{DD} and the ground potential GND. Both transistors are driven by the same input signal in. Since the types of the transistors are complimentary, one of them will always have

low impedance (be "turned on"), while the other one will be cut-off mode ("turned off"), assuming that the potential of the input signal reaches (digital logic) values close to either V_{DD} or GND only. The case where *in* is close to $V_{DD}/2$, where both transistors would be (partially) turned on, is undesirable for digital applications and is of interest to analog applications only.[1]

Suppose the input signal *in* is low (meaning, it has a value close to GND). Then the PMOS transistor is turned on, while the NMOS transistor is turned off. Therefore there would be a low-impedance conductive connection between V_{DD} and the output node *out*. So the potential of the output node would be high (meaning close to V_{DD}) in that case.

In the opposite case, where *in* would be high, the PMOS transistor would be turned off, while the NMOS transistor would be turned on. In that case the output node would have a low-ohmic conductive connection to GND, which results in an output voltage close to GND.

So the basic principle of CMOS circuits is to have a so-called *pull-up network*, which provides a low-ohmic path from V_{DD} to the output node, and a (complementary) *pull-down network*, which provides a low-ohmic path to GND. The pull-up and pull-down networks must be driven by the same control signals. Since PMOS transistors need a low input signal to be turned on and NMOS transistors need a high input signal to be turned on, it is ensured that one of the networks is always turned off. Otherwise there would be a low-ohmic connection between V_{DD} and GND, short-circuiting the supply voltage. In the best case such a short circuit interferes with the desired operation of the circuit; in the worst case it leads to thermic destruction of the circuit.

The pull-up and pull-down network are not only complementary with respect to the transistors they are composed of. Usually they are also complementary with respect to their schematic layout: A serial connection in one branch normally corresponds to a parallel connection in the other branch and vice versa (cf. figure 5.2). The "C" in CMOS is due to this complementary setup of pull-up and pull-down networks.

CMOS circuits also have very interesting properties concerning power consumption. These properties will be discussed in §5.4.

5.1.2 Standard Cells

Large and complex digital circuits are usually not designed on transistor level. Instead so-called *standard cells* from homonymous libraries are used. A standard-cell library normally contains a few dozens up to a few hundred cells implementing basic logic functions. Examples for standard cells would be inverters, NAND gates, NOR gates, multiplexers, flip-flops, etc. Since these functions are needed in every digital circuit, standard-cell libraries eliminate the necessity to "reinvent the wheel" every time a new circuit is designed.

Example: NAND Gate

A very simple example of a standard cell has already been mentioned: The CMOS inverter. Another basic logic function which is needed quite often is the NAND relation. In its basic version it has two inputs and one output; versions with more inputs (in practise: up to four) are possible. Table 5.1 shows the truth table of a 2-input NAND.

[1] In analog applications the circuit depicted in figure 5.1 can be used as an inverting amplifier. Confer [Raz02] for details.

a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

Table 5.1: *Truth table of a 2-input NAND. The output out is* 0 *if and only if both inputs* a, b *are* 1*. In all other cases the output is* 1*. A realization in CMOS technology is depicted in figure 5.2.*

As the name suggests a NAND gate is the inverse of an AND gate, which would implement a simple logical and-relation. NAND gates can be realized quite easily in CMOS technology. Therefore they are used frequently. If AND functionality is required, it is normally achieved by using a NAND gate and an inverter.

The schematic of a NAND gate can be seen from figure 5.2. For the sake of clarity the connections of the inputs to the gates of the transistors have not been drawn, but only made clear through labeling. The NAND gate consists of four transistors: Two PMOS and two NMOS transistors. Each of the two input signals a, b is connected to the gate of one PMOS and the gate of one NMOS transistor.

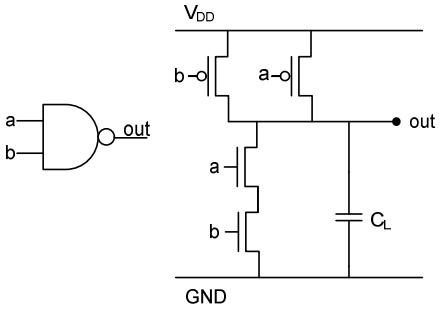

Figure 5.2: *Symbol and schematic of a CMOS* NAND *gate. The pull-up network comprehends a parallel circuit of two PMOS transistors, while the pull-down network consists of two NMOS transistors in series.*

The connection between the truth table (cf. table 5.1) and the schematic in figure 5.2 is easy to see: Whenever (at least) one input signal is 0, the corresponding NMOS transistor is turned off. Since the NMOS transistors are in serial connection, the pull-down network is disabled. However one input signal set to 0 already suffices to enable the pull-up network, since a 0 turns the corresponding PMOS transistor on, and the PMOS transistors are in parallel connection.

If more inputs are needed additional transistors can be added. For each additional input an additional PMOS transistor must be added (in parallel) to the pull-up network and an additional NMOS transistor must be added (serially) to the pull-down network. In practise, however, there are limits to this idea: The (non-zero) resistance of the transistors in serial decreases the performance of the NAND gate, with increasing number of transistors in serial. Furthermore there is the so-called body effect, which is responsible for decreasing the performance of NMOS transistors, which are not directly connected to GND (cf. [Raz02] for details). Therefore NAND gates with more than four inputs are highly unusual and rarely used.

NOR Gate

Another common logic function is the NOR function, which is the inverse of the or-relation. Its truth table is given by table 5.2.

a	b	out
0	0	1
0	1	0
1	0	0
1	1	0

Table 5.2: *Truth table of a 2-input NOR. The output out is* 1 *if and only if both inputs* a, b *are* 0*. In all other cases the output is* 0*.*

A NOR gate can be realized quite similar to a NAND gate. In fact the schematic is almost identical. The only difference is that for a NOR gate the PMOS transistors are in serial connection and the NMOS transistors are in parallel. Using the same reasoning as before it is easy to show that such a circuit would conform to the truth table above.

NOR gates can also be scaled to more inputs, just like NAND gates. However, the same limitations apply. In addition NOR gates suffer another disadvantage. PMOS transistors have a lower conductivity than NMOS transistors (of the same size). This is due to the fact the main charge carriers in PMOS transistors are "holes", which have a lesser mobility than the charge carriers of NMOS transistors (electrons).

Standard-Cell Libraries

It is customary that chip manufacturers provide a standard-cell library to their customers. What information exactly is provided varies from case to case. Usually the library includes a textual specification of the cells, including a truth table, timing behaviour, power consumption details and area requirements. A formal specification of the cells in a hardware description language is also included normally. These HDL modules are for use in conjunction with synthesis and place&route tools.

Some companies also include more detailed information about their standard cells in the library. For example schematics (on transistor level) or even layout information. Such additional data is useful when doing parasitics extraction and/or transistor-level simulations.

Used Standard-Cell Libraries

Three different standard-cell libraries have been used for the practical work described in this thesis. One of them is from *austriamicrosystems* (AMS): [aus] is a $0.35\,\mu m$ CMOS technology.

The other two libraries are from UMC: [UMCb] is a $0.25\,\mu m$ CMOS technology, and [UMCa] is a $0.13\,\mu m$ CMOS technology.

5.1.3 Synchronous Circuits

Standard cells can be used to build so-called *synchronous circuits*. These kind of circuits is the most widespread in digital design. A synchronous circuit is composed of sequen-

tial elements, capable of storing data, and combinational elements which can be used to implement Boolean functions (cf. figure 5.3).

Sequential elements are also referred to as *flip-flops*. The simplest possible flip-flop has two inputs, the data input and the clock input, and one data output. Whenever the clock input has a transition from 0 to 1 (positive edge) the flip-flop stores the current value of its data input signal. The output signal follows accordingly, after a short delay. Therefore such a flip-flop is called *positive-edge-triggered flip-flop*.

For the sake of completeness it should be noted that it would of course also be possible for the flip-flop to react on the 1 to 0 transition (negative edge) of the clock signal. As long as all flip-flops in a circuit react on the same edge, either the positive or the negative one, this has no influence on the functionality of the circuit. Therefore positive-edge-triggered flip-flops will be assumed throughout the rest of this work.

There are also *level-sensitive* storage elements, which are called *latches*. The output of a latch follows the value of the data input, as long as the clock signal has a certain value. This state is called *transparent* state. If the clock signal has the opposite value, the output is stuck at its current value. This state is called *opaque* state. Synchronous circuits do not need latches. All storage elements in synchronous circuits should be edge-sensitive flip-flops.

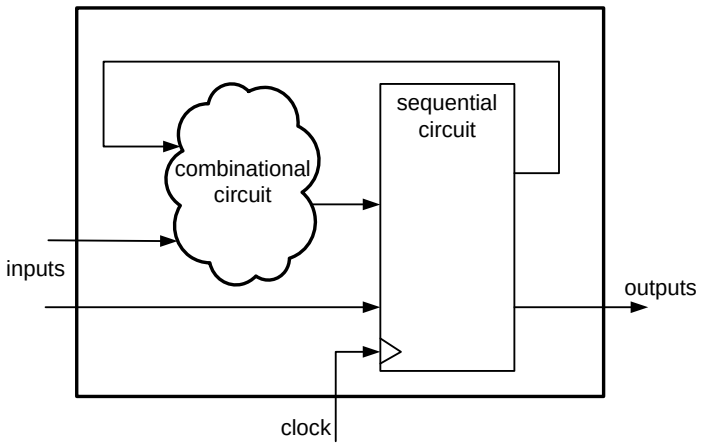

Figure 5.3: *Sketch of a synchronous circuit, consisting of sequential circuit, a combinational circuit, inputs, and outputs.*

Figure 5.3 depicts the basic schematic of a synchronous circuit. At each positive edge of the clock signal the data output by the combinational circuit, which depends on the feedback from the sequential circuit and the input signals, is stored by the flip-flops in the sequential circuit. After a very short delay the changes are visible at the outputs of the flip-flops. Thus new inputs are provided to the combinational circuit. Depending on the depth of the combinational network it takes some time until the outputs of the logic functions have settled. Temporary changes of these signals cause no harm, since the sequential circuit will store new values at clock edges only.

It must, however, be ensured that all signals are stable shortly before the next positive edge of the clock signal, because then new values will be stored. The settling time is determined by the longest delay which can (theoretically) occur. The path which causes the longest delay is called the *critical path*. The length of the critical path determines the maximum clock frequency at which the circuit can be operated. If the maximum

clock frequency is exceeded it is possible and likely that a clock edge arrives before all combinational signals have settled. Therefore the flip-flops would store invalid intermediate values instead of correct results from the Boolean functions. This causes the circuit to malfunction.

An essential property of synchronous circuits is the fact that they have only one clock signal, which is connected to *all* flip-flops of the circuit. So all the stored data is updated (synchronously) at the same moment of time. The remainder of this work will only consider such synchronous CMOS circuits.

5.2 Controller and Datapath

Synchronous circuits are often split into two parts: The *controller* and the *datapath*. The datapath stores all the data processed by the circuit and also contains the necessary combinational circuit to perform the required calculations and operations. The controller is responsible for storing the current state of the circuit and determining the order in which datapath operations are executed (cf. figure 5.4).

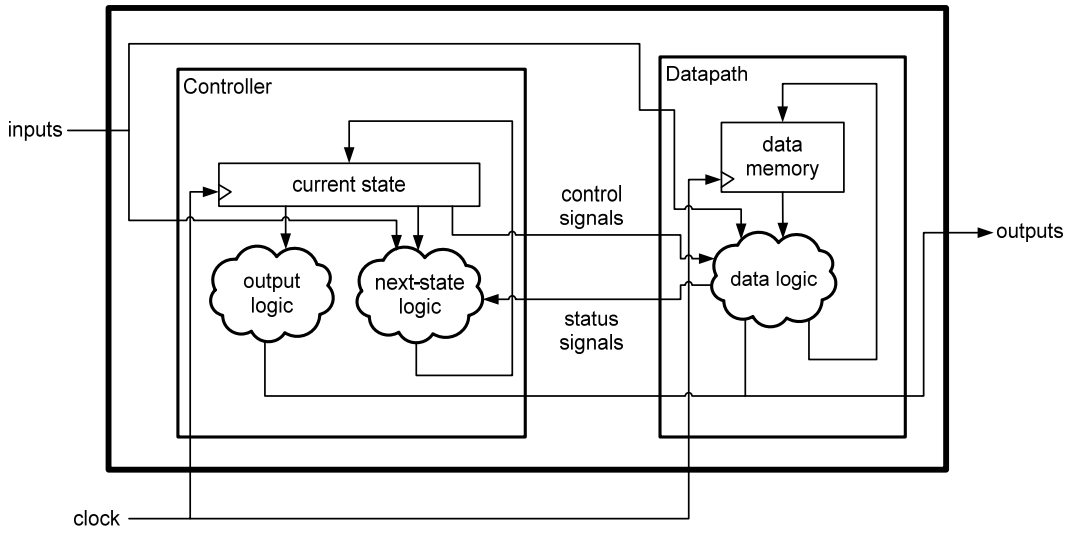

Figure 5.4: *A circuit split into* controller *and* datapath. *The controller controls the datapath via control signals and thereby ensures that the operations which are performed by the datapath actually lead to the desired overall result. To do so the controller keeps track of its internal state and also examines status signals from the datapath.*

5.2.1 Splitting Controller and Datapath

Metaphoric Example

Speaking in metaphors the datapath is like a car, and the controller is its driver. A car contains all parts which are necessary for driving on the road, like tires, an engine,.... Moreover it presents status signals to the driver, like for example the current driving speed. The driver monitors these status signals and also external input signals, for example traffic lights. From this data the driver derives an appropriate course of action ($\Rightarrow$ the *next state*). Based on that the driver issues commands to the car by using control signals, such as kicking down the accelerator, pedaling the clutch, or changing the gear. Thereby the

behaviour of the car is influenced, which causes it to update its status signals, and the cycle starts again.

Controller and Datapath in Synchronous Digital Circuits

In a synchronous digital circuit the datapath usually consists of memory elements and elements capable of performing arithmetic and/or logic operations. The latter normally have input signals which can influence their operation. For example input signals could be used to choose one of several possible operands. Further details about datapath modules will be presented in §5.3.

The controller is responsible for generating the control signals for the datapath in such a manner that the circuit actually performs its desired functionality. Suppose for example a circuit should calculate a modular exponentiation of a given number, by means of the square-and-multiply algorithm (cf. algorithm 4 in §3.4.1, page 31). The basic datapath elements which would be necessary for this calculation are a multiplier and memory for the operands and temporary variables. The controller's job would be to issue control signals which select the operands for the multiplier in the correct order (as outlined by algorithm 4). To do so the controller must analyze status signals such as for example the current bit of the exponent on one hand, and on the other hand it must keep track of the current state, which includes in this case storing and updating a loop counter.

5.2.2 Implementing Controllers

While datapath modules sometimes need to be modelled on lower abstraction levels (down to the logic level) to achieve good results, controllers are usually modelled by a behavioural description in an appropriate hardware description language (HDL). This behavioural description usually implements a corresponding (finite) state machine.

Transforming the behavioural description into a circuit comprised of standard cells is a task which can (in most cases) be left to synthesis tools. Therefore estimates of the controller's influence on the overall circuit's resource requirements cannot be done directly. Usually one assumes that the controller takes up approximately 10% of the overall area and does not contribute significantly to the timing behaviour of the circuit. These values are taught by experience.

There are, however, exceptional cases where these rules of thumb do not apply. One such example would be high-speed circuits which have datapaths with very short delays. In such cases the critical path of the circuit could (partially) lie in the controller. Therefore the controller would contribute significantly to the timing behaviour of the circuit.

5.3 Datapath Modules

This section will present some common elements which are used frequently in datapaths of digital circuits, such as adders, multipliers, and memories. An overview of some important implementation options of these elements will also be given.

5.3.1 Adders

Adders are the most primitive arithmetic units. They are used to calculate the sum of two (or possibly more) numbers. In the simplest case, two numbers are given in binary

representation and both have an equal number of bits. The most basic implementation to add such numbers is called *ripple-carry adder*.

Ripple-Carry Adders

Ripple-carry adders are composed of full-adder cells. All standard-cell libraries usually contain such cells, since they are frequently used. A full-adder cell has three inputs and two outputs. The inputs are usually called a, b, and $carry_{in}$. Although the naming might suggest that there is something special about the input $carry_{in}$, the three inputs are exchangeable, as it can be seen from the truth table (cf. table 5.3). The outputs, which are usually called sum and $carry_{out}$, however, are not interchangeable. They can be interpreted as a 2-bit number $(carry_{out}, sum)_2$ which is the sum of the three (1-bit) input numbers $a, b, carry_{in}$. So $carry_{out}$ is the most-significant bit of the result and sum is the least-significant bit.

inputs			**outputs**	
a	b	$carry_{in}$	$carry_{out}$	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 5.3: *Truth table of a full-adder cell. The two outputs can be interpreted as a 2-bit number* $()c_{out}, sum)_2$ *which is the sum of the three (1-bit) input numbers* a, b, c_{in}.

A ripple-carry adder for adding two k-bit numbers is composed of n full-adder cells. The inputs a, b of each cell are connected to corresponding bits of the two input numbers. The $carry_{in}$ input of each cell is connected to the $carry_{out}$ of the previous (less significant) cell. The $carry_{in}$ of the first (least-significant) cell can either be shorted to 0 or it can be connected to a flip-flop which stores a carry bit from a previous operation. The sum outputs of all full-adder cells, together with the $carry_{out}$ output of the last (most-significant) cell, form the $(k + 1)$-bit result of the addition.

Ripple-carry adders owe their name to the fact a carry bit could be propagated from the least-significant cell to the most-significant cell. In a sense the carry is "rippling" along the full-adder cells. This is also the most severe disadvantage of ripple-carry adders. The critical path traverses all cells. Therefore it is very long, and it grows linearly with the number of cells. That means that especially for cryptographic applications, which tend to have operands with a large number of bits, ripple-carry adders cannot achieve high speeds.

Improvements to Ripple-Carry Adders

Several improvements to simple ripple-carry adders are possible, which all aim at counteracting the effect of the carry-propagation delay. Since the implementations which will be presented in §6 do not incorporate such optimizations, they will be sketched very briefly and for the sake of completeness only.

So-called *carry-save adders* are one of these improvements. They do not propagate the carry from one cell to the next, but instead they output all carry bits of all cells as a binary number. So carry-save adders do not output one number, but two. These two numbers must still be summed up in order to obtain a final result. So carry-save adders perform a 3-to-2 compression. They take three input numbers and output two numbers, where the sum of the output numbers equals the sum of the input numbers.

Carry-save adders are especially useful when summing up multiple operands, because they can be arranged in a tree-like structure. Their critical path is a size-independent constant. Their disadvantage, however, is that their final result is still in the so-called *carry-save representation* and must be converted to normal binary representation. This can be done by another adder, for example a ripple-carry adder. Another possibility to do this conversion is pointed out by [Wol04]: By repeatedly adding 0 to the (redundant) carry-save representation, carries vanish. This procedure can be repeated until no further carries are present. This method does not need extra hardware resources; instead it takes more clock cycles to execute than one big ripple-carry adder would need for the conversion. On the other hand the critical path of a carry-save adder is much shorter than the one of a ripple-carry adder of the same size. Therefore the conversion outlined by [Wol04] can use higher clock frequencies. The corresponding shorter cycle times make this conversion method faster (with respect to total time) in the average case, although it needs more cycles.

Yet another improvement to ripple-carry adders are so-called *carry-lookahead adders*. They utilize the facts that a carry can only be generated if at least two input signals of a full-adder cell are 1 and that a carry cannot propagate through a cell with at least two input signals set to 0. These statements can easily be proven by looking at the truth table of a full-adder cell, as it is shown in table 5.3. By using additional logic to examine the carry-generation and carry-propagation conditions, these adders can determine the final result faster. Details about the implementation of carry-lookahead adders can be found in [Par00].

5.3.2 Multipliers

Bit-Serial Multiplication — The "Scholar's Method"

A very simple form of multiplication is the so-called *bit-serial multiplication*. Since it is based on the same principles as the multiplication method taught to young students in primary school, it is also called the *scholar's method*. It consists of two steps: Partial product generation and their accumulation.

One operand is scheduled at full precision. The other one is examined bit by bit. The first operand is (sequentially) multiplied by each bit of the second operand. Since a bit can only be 0 or 1 the result of this multiplication can only be either 0 or the first operand. Therefore these so-called *partial products* can be computed very easily with AND-gates.

All the partial products must be summed up. However either the partial products or the accumulator must be shifted first, to ensure correct alignment.

Hardware resources of a bit-serial multiplier are very modest. However the multiplication of two k-bit numbers takes k clock cycles.

Digit-Serial Multiplication

The idea of bit-serial multiplication can be generalized to digit-serial multiplication. Instead of multiplying the first operand by just one bit of the second operand, the first operand is multiplied by a complete digit (consisting of several bits, cf. §3.3.2) of the second operand. That complicates partial-product generation, since the partial product can now be one of 2^d different values, when using d-bit digits. Therefore additional combinational resources are necessary.

The gain of this method is that instead of k cycles, multiplication of two k-bit numbers now only takes $\lceil k/d \rceil$ cycles, where d is the digit size (in bits).

Combinational Multipliers

By setting the digit size d of a digit-serial multiplier equal to the number of bits of the operands (k), one obtains a combinational multiplier. Such a multiplier does not contain any storage elements or accumulators. Its output is derived in one cycle, as a pure combinational function of the inputs.

It is obvious that such a multiplier requires enormous resources. Therefore this approach is suited for rather small operand sizes. For larger operands such a multiplier could be used to multiply single digits during a multi-precision multiplication, as outlined by algorithm 2 on page 30.

Adders within Multipliers

Multiplications usually involves the accumulation of partial products. Therefore adders are necessary within multipliers. If the operands (and therefore the partial products) are large, these adders contribute significantly to the area demands and the critical path of the multiplier. As a result the choice of the adder(s) has an important influence on the overall performance of the multiplier.

Carry-save adders are particularly suited for use in multipliers, because of their 3-to-2 compression property. In each step they can calculate a new carry and sum out of the old carry and sum and the current partial product. After the multiplication is finished the result can be converted to binary representation utilizing the above-mentioned trick described in [Wol04], which does not require any additional hardware resources.

5.3.3 Remark on Optimizations of Arithmetic Modules

It should be noted that many different optimizations for basic arithmetic units such as adders and multipliers have been investigated. There are many papers and text books on this subject, [Par00] is just one of them. However, nearly all optimizations presented in such works aim at optimizing either the latency of a unit, its throughput or both of them. Most of these optimizations need additional hardware resources and therefore the optimized circuits always take up more area and consume more power.

Compared to latency and throughput optimization, very few literature is available on area and power optimization. This may be due to the fact that area- and power-efficient implementations are of particular interest in the context of contactless technology, which developed only recently, compared to other applications.

5.3.4 Memories

Random access memories (RAMs) are another frequently used component. Many circuits must store a significant amount of data for their operation. Memories provide a flexible solution for that.

Properties of Memories

A memory has the following main properties:

- *Size:* The number of data elements the memory is able to store. This determines the width of the memory's address bus.
- *Element size:* The number of bits of one element.
- *Number of ports:* There are single-ported memories, which have only one address bus. Reading from and writing to the memory must be done sequentially.
  There are also dual-ported memories, which have two address buses and allow concurrent reading and writing.
  Memories with even more ports, which would allow reading (or writing) more than one element at a time would be possible, but are rarely used.

Storage Elements

Memories can be based on different storage elements. Flip-flops are one possibility. However, flip-flops take up a considerable area. Therefore they are not ideal for large memories.

Since memories are such a vital component for nearly all systems, many chip manufacturers provide standardized memory blocks, which are often based on less area-consuming storage elements, such as static or dynamic RAM cells. These memories need less area than a flip-flop based RAM of the same capacity. Their drawback, however, is that they are customizable within certain constraints only. An example for such constraints would be that the number of elements of the memory can only be a power of 2.

Non-Volatile Memory

Many applications require that some data is stored even while the circuit is disconnected from power supply. Flip-flops or dynamic RAM cells cannot accomplish that. Memory that retains its data even without power supply is called *non-volatile memory* (NVRAM). Such memories can be based on technologies like EEPROM or flash memory.

NVRAM usually consumes much more power than normal RAM, especially for writing. Furthermore NVRAM usually has a longer access time, and the total number of write cycles is limited. Therefore it should only be used where necessary and not as a general replacement for other RAMs.

5.4 Power and Energy Consumption

This section will shed some light on the power and energy consumption properties of digital CMOS circuits. First the sources of power consumption in CMOS circuits will be analyzed, followed by a comparison between power consumption and energy consumption.

Power consumption of a CMOS circuit is divided into two parts: Static power consumption and dynamic power consumption (cf. equation 5.1).

$$P_{total} = P_{static} + P_{dynamic} \tag{5.1}$$

5.4.1 Static Power Consumption

As it can be seen from the figures 5.1 and 5.2, CMOS cells of this kind do not need any electric current to produce an output voltage. Other digital logic styles, such as for example transistor-transistor logic (TTL), require a current flowing through either the pull-up or the pull-down network, in order to establish a logic signal level at the output. Therefore power is consumed all the time.

In CMOS, however, the static case, where the values of the input signals do not change, causes no currents. Therefore there is no static power consumption, except for leakage currents. Leakage currents occur, because the electrical resistance of a transistor is inherently always finite, even when it is switched off. In other words: A transistor is not an ideal isolator. Although transistors which have a gate potential below their threshold are very high-impedance, there can still be some current, often called *subthreshold leakage current*. These currents are usually very small. However, the minimum size of transistors as well as the necessary threshold voltage to turn on a transistor have decreased significantly over the last decades. The corresponding decrease of other power consumption sources has made leakage currents more important in a relative view.

The total power dissipated through leakage currents is roughly proportional to the number of transistors in the circuit. Since leakage currents are something inherent to CMOS technology, the only way for a designer to minimize them is to minimize the circuit's area and thereby the number of transistors. Since that is a desirable goal anyway, leakage currents will not be considered any further.

5.4.2 Dynamic Power Consumption

Dynamic power consumption is due to switching activity. Each node in a CMOS circuit has a parasitic capacitance. For the output node of a cell, which is connected to several input nodes of subsequent cells, this capacitance is comprised of the following: The sum of the gate capacities of the connected gates, which are the inputs of the subsequent cells, and the parasitic capacity of the interconnect wire. The sum of the gate capacities is mostly determined by the number gates which are connected to the node. Therefore the number of inputs which are driven by an output node, is an important characteristic. It is called *fan out*. The fan out also influences the parasitic capacity of the interconnect, because the more inputs must be connected to the output node, the longer the interconnect wire will have to be.

The effective overall capacity of an output node is also called *load capacity* C_L. It is symbolically depicted in figure 5.1 and 5.2. Whenever the signal level of the output node changes, this capacity must be charged or discharged. That necessitates a current, which causes power consumption.

Equation 5.2 shows, on which parameters the switching power of a cell depends on:

$$P_{switch} \sim C_L \cdot V_{DD}^2 \cdot f_{switch} \tag{5.2}$$

where C_L is the load capacity driven by the cell, V_{DD} is the supply voltage and f_{switch} is the switching frequency. The switching frequency is often also expressed as the product

of the switching probability Pr_{switch} and the clock frequency of the circuit:

$$f_{switch} = Pr_{switch} \cdot f_{CLK} \tag{5.3}$$

There is also another source of dynamic power consumption. Suppose the input signal of the inverter depicted in figure 5.1 changes its value. While the signal does the transition, there is a short moment where the transistor which was turned off starts to turn on, while the other transistor is not yet completely turned off. So for a short moment, both transistors are (partially) turned on. That leads to a direct conductive connection between V_{DD} and GND. A current can flow along this connection and cause power consumption. This current is often called the *short-circuit current*.

5.4.3 Power Consumption versus Energy Consumption

Although it is easily overlooked, there is a distinct difference between power consumption and energy consumption. Depending on the application it might be desirable to minimize one or both of them. For example suppose there is a battery powered device. For such a device it is clearly desirable to minimize the *energy* necessary to perform a certain operation, because the overall amount of energy which is available is limited. The power consumption, however, is of lesser concern. Higher power demands can be outweighed by shorter execution times. Suppose there are two implementation alternatives for a certain operation. The first one needs twice as much power as the second one, but only a quarter of the time. Then the first option is clearly preferable for a battery-powered system, because it consumes only half as much energy as the other one. Of course other constraints such as external timing constraints or the maximum power output of the battery must be met.

There are other applications, where it is the other way round. Contactless smartcards and passive RFID tags for example. Such systems are powered by induction currents, caused by the oscillating magnetic field of the reader device. During each interval Δt the reader field puts a certain amount of energy into the passive device. The device must not consume more than this amount of energy *during that time* Δt. So in that case the important factor obviously is the amount of *energy per time* (or *power*) the device consumes. The overall amount of energy is of lesser concern, since (figuratively speaking) the reader field is an almost "infinite" energy reserve. So in that case, an implementation variant which requires for example only half the mean power, but more than twice the time to perform a certain operation, might still be preferable, although it consumes more energy altogether.

5.5 Power-Saving Techniques

Power and energy consumption are both important design factors for digital circuits. In most scenarios either one or both of them should be minimized. RFID tags for example impose very fierce constraints on power consumption. The upper bound for the current is in the range of a few microamperes (cf. [FW07], [FR06]). Together with supply voltages in the range of a few Volt, the power budget is limited by just a few microwatts. To achieve that, power-saving techniques have to be applied rigidly during the whole design and implementation process.

Most often reducing power and energy consumption comes at a price; for example longer execution times of certain operations. Such trade-offs, in combination with ap-

plication specific constraints which must be met, limit the possibility to save power and energy.

The degree to which the techniques described below can be applied to a circuit may depend on application-specific constraints. This section will only sketch the basic ideas behind some common techniques.

As §5.4.3 has outlined, power consumption is something different from energy consumption, although the two are related. The remainder of this section will focus on techniques to decrease the power consumption of a circuit, since this is of more interest in the context of contactless systems. But power saving is also important for completely other fields of application. The power consumed by a circuit is turned into thermal power which heats the circuit. This heat energy must somehow be dissipated, otherwise the circuit will overheat and be destroyed. Circuits like for example general-purpose processors need large cooling units to cope with this problem. So even if an "unlimited" amount of power is available it might still be desirable to limit and/or minimize the power consumption of a circuit.

5.5.1 Clock Gating

Clock gating is a technique which is based on the assumption that certain blocks of the circuit are not needed all the time. So while their functionality is not required, they can be deactivated. This works in the following way: So-called *clock-gating cells* are introduced into the clock tree of the circuit. The output of a clock-gating cell depends on the value of the corresponding *enable signal*. If the enable signal is 1, the clock signal is passed through the clock-gating cell. However, if the enable signal is 0, the output is always 0; the clock signal is *gated*.

Clock gating saves power in two ways: First, as already said, it effectively deactivates entire blocks of the circuit. Flip-flops store the values of their input signals whenever they receive a (rising) edge of the clock signal. If there are no such edges, the flip-flops do not do anything. Therefore they do not consume any dynamic power (cf. §5.4.2) either. In addition, all combinational cells which are driven (only!) by these flip-flops do not consume dynamic power either. Since the outputs of the flip-flops are fixed, there is no signal switching activity in the logic driven by them.

It should be noted that there are also other ways to deactivate flip-flops. Most standard-cell libraries contain so-called *enable-type flip-flops*, which have an enable signal to activate or deactivate them. Such flip-flops, however, take up more area than standard flip-flops, which store a new value at *every* (rising) clock edge.

Furthermore clock gating saves power in a second way, which cannot be achieved by using enable-type flip-flops. The switching transitions of the clock signal itself consume considerable power. This is due to the fact that the clock signal is routed throughout the entire circuit and is connected to a huge amount of cells. Therefore the parasitic capacitance of the clock net is significant. In addition to that the clock signal does a lot of transitions too. Its value changes twice per cycle. By masking the clock signal for an entire block of the circuit the parasitic capacitance of the clock line is reduced, which leads to decreased power consumption.

Disadvantages

Clock gating can save significant amounts of power, up to 50%. Furthermore it comes at a very low price. Only one clock-gating cell (per block) is necessary, which requires only

an insignificant area. The additional requirements for control logic, which generates the enable signal for the clock-gating cell, are usually negligible too.

The obvious disadvantage of clock gating is that the disabled blocks cannot be used any more. Some applications, however, might require that (almost) all their blocks are operational all the time.

5.5.2 Sleep Logic

The basic idea of sleep logic is in some respects similar to the idea of clock gating: Unnecessary signal activity is prevented by masking certain signals with corresponding enable signals. Sleep logic can be introduced into signals going from one block of the circuit to one ore more other blocks. The signals are passed through an AND-gate together with the corresponding enable signal. If and only if the enable signal is 1, the wanted signal can "pass" the AND-gate. Otherwise the output of the AND-gate will always be 0.

There are two scenarios where this is beneficial. For the first, suppose the signal under consideration is the input to some very complex logic, comprised of many (consecutive) gates. In that case a transition of this signal could cause many transitions within the driven logic. So the overall (dynamic) power consumption would be significant. However, the final output of the logic block might not even be of interest to the application at the moment. So the power to evaluate the logic function would have been wasted. If the signal is routed through sleep logic, it can be masked whenever the output of the attached logic block is not of interest. In that way signal activity within the logic block is prevented and power is saved.

The second scenario addresses blocks which have a large fan out. A good example is given by memory blocks. The input signal of the memory must be routed to every single memory location. Therefore the parasitic capacitance of this signal line is very high; it correlates linearly with the number of memory locations. However, usually only one location of the memory really "needs" the data: The one specified by the write address. If the input signal is routed to the memory locations in a tree-like manner, sleep-logic can be introduced in each branch of the tree. By doing so, signal activity is limited to "active" branches. The correlation between the number of active branches (and hence the parasitic capacitance) and the size of the memory is logarithmic. Therefore power consumption significantly reduces.

Disadvantages

Just like clock gating, sleep logic implicates introducing additional cells into the circuit, which increases the area demands. Furthermore control logic is necessary to generate the enable signals. Whether or not these demands are outweighed by the amount of power saved by the sleep logic, depends on application-specific circumstances; the question cannot be answered in general.

5.5.3 Avoiding Glitches

Glitches can be another source of unnecessary power consumption. Suppose there are six different signals a to f, which all change their value at the same time. Figure 5.5 depicts a circuit where the signals are routed through a different number of cells before they reach the last cell, which generates the output signal *out*. Therefore each of this signals has a different delay. Signal a is directly connected to the last cell, whereas other signals pass

one, two, or three other gates before reaching the last cell. Each signal which arrives at the final cell causes this cell to evaluate its inputs and thereby maybe changing its output value. So if all signals arrive at different times, there may be unnecessary changes of the output node, before it settles on its final value.

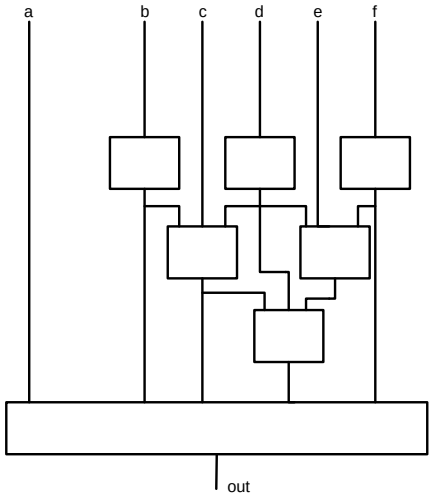

Figure 5.5: Bad Example: *Due to the unbalanced lengths of the signal paths and the corresponding delays, glitches will occur, which consume unnecessary power.*

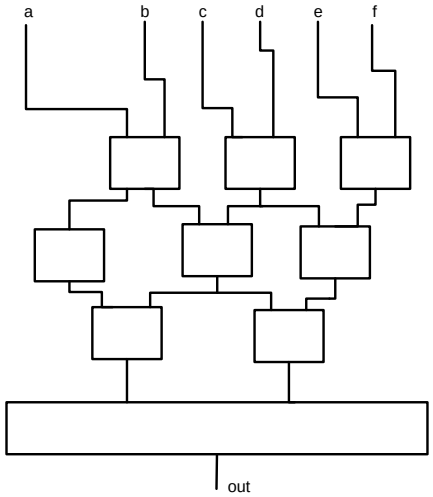

Figure 5.6: Good Example: *The signal paths are more or less equal, which makes glitches less likely to occur.*

In contrast to this figure 5.6 shows a circuit where the signals a to f all pass the same number of cells before they reach the final cell. Therefore the input signals of the last cell will most likely change (almost) at the same time. Glitches of the output signal are prevented.

If implementing the desired functionality necessitates a circuit layout similar to the one shown in figure 5.5, it can be turned into something more like the circuit in figure 5.6 by inserting buffers. That could even out the signal delays without changing the functionality. It should, however, be noted that these buffers would consume power as well, but if the fan out of the overall block was very large, there could still be a decrease in the overall power consumption.

It should also be noted that glitches are less likely to occur in circuits which have a rather short critical path. The longer logic paths in the circuit are, the more likely glitches will occur.

5.5.4 Decreasing the Clock Frequency

Reexamining equations 5.2 and 5.3 shows another way of saving power. The power consumption is proportional to the clock frequency f_{CLK} of the circuit. Therefore reducing the clock frequency also reduces the power consumption. For circuits working with an external clock, which is not generated on-chip by the circuit itself, this method can even be applied after the circuit has been manufactured, because no changes to the circuit itself are necessary.

However, if the clock frequency is reduced this increases the time the circuit needs

to complete an operation. Whether this is an acceptable price to pay or not depends on application-specific circumstances.

5.5.5 Synthesis Optimization

Yet another possibility to reduce the power consumption of a circuit is to use the power-optimization features of a synthesis tool. Once a circuit is specified on standard-cell level, it can be simulated. During simulation the value changes of all internal nodes can be recorded. Later on these records can be used to calculate the switching probability of each node, thereby finding the nodes with the highest switching probability. These nodes are the most significant source of power consumption.

State-of-the-art synthesis tools are capable of analyzing that kind of information and changing the circuit in a way that preserves functionality but reduces the power consumption. To do so the tools can employ the tricks described in §5.5.1, §5.5.2, and §5.5.3 on the one hand. On the other hand some tools are also capable of doing other transformations, such as for example implementing specific Boolean functions in other ways (using different cells), which might consume more (delay) time and area resources, but less power.

Since tool-based power optimization is dependent on the switching probability of each node in the circuit, great care must be taken when doing simulations to determine these probabilities. The operations which are simulated must be representative examples for the real operation of the device. Therefore input data must be carefully chosen.

For circuits with cryptographic functionality this is not so difficult, since in most of the cases the input data is random or pseudo-random. However for applications such as for example an mp3 decoder, it might be much more difficult to choose appropriate input data, which is representative for the average use case of the device.

5.6 Further Reading

A commonly known text book about designing digital integrated circuits, including relevant background information is [Rab96]. It contains a very good overview of all topics which might bother a designer while trying to implement a digital circuit.

Readers more interested in details about electronics should refer to [TS02], which presents detailed knowledge about the principles of semiconductor devices like diodes and transistors, including their application in simple logic cells.

Another work worth mentioning is [Raz02]. Although it focuses on analog design, it contains valuable information about the basics of field-effect transistors (MOSFETs).

Interesting details about hardware implementations of arithmetics can be found in [Par00].

Chapter 6

Design Space

As §4 has outlined, there are many different variants of the GPS authentication scheme. Furthermore most variants offer many parameters which can be set to convenient values with respect to application-specific circumstances. This chapter explores the design space spanned by all these variants and parameters. The focus lies on area- and power-efficient implementations on the prover's side. Effects of design decisions on verifier implementations will be neglected.

6.1 Overview of Design Methodology

Before presenting details of the practical work performed for this thesis, a short overview of the design methodology will be given. Basically the approach which was used is a top-down design flow. That means that the work started at a high level of abstraction, modelling complete systems in a crude way, to subsequently break those systems into parts ("divide&conquer"), which were then modelled more detailed on the next lower abstraction level. This procedure is recursively applied until the desired exactness of the models and the corresponding abstraction level has been reached.

In this particular case the practical work was divided into two main parts: Developing high-level models by means of an *high-level modelling and evaluation framework*, custom-built for and tailored towards exploring different implementation variants of GPS on one hand, and on the other hand implementing some of the promising variants in synthesizeable HDL specifications.

6.1.1 High-Level Models

There are many reasons for creating high-level models. One is for example that they facilitate trying out even completely different implementation variants with relatively low effort. This is due to the fact that the systems are only modelled in a crude way, reflecting only little details. Therefore making changes to them, even major ones, is much easier than applying the same changes to more detailed implementations. To give a somewhat extreme example: It is quite easy to change a few "+" operators to "−" in a high-level model implemented in any suitable programming language. However, achieving the same semantical changes by rearranging standard cells, or maybe even transistors in a schematic of a circuit is much more work.

Proofs of concepts are another useful purpose of high-level models. Since they are generally very flexible, certain parts of a system could be modelled in more detail to see

whether a specific idea or concept really works the way it is supposed to. For example when trying out a special multiplication architecture, the multiplication could be modeled more detailed, for example on a level reflecting bit-serial partial-product generation and accumulation, while all other operations are still performed using the usual high-level arithmetic operators. Thereby it can be checked whether the multiplication architecture contains inherent design flaws, without having to model the complete system on a lower level. The latter would require a larger amount of work.

High-level models are also useful to generate test data for HDL implementations. Most programming languages have some mechanism to generate random data, which can be used as input to the model. The same input can later be applied to a HDL implementation and the outputs of the high-level model and the HDL implementation can be compared. More details of the high-level modelling and evaluation framework, which has been created during the course of this thesis will be presented in §6.2.

6.1.2 HDL Implementations

The drawback of high-level models is that they can only give estimates about the system's performance and resource requirements. The accuracy of high-level estimations varies, depending on the detailedness of the model. *Hardware description languages* (HDLs) provide more exact possibilities to model and simulate systems. Furthermore HDL implementations can be used as input to synthesis tools to generate standard-cell circuits. Thereby one gets good approximations of the systems area and timing requirements. By using *place&route* tools it is possible to obtain a complete layout of the circuit. This is all the information a semiconductor manufacturing company needs in order to produce silicon realizations of the circuit. Therefore the output of place&route tools not only provides estimates but exact values for the area requirements of the circuit. Timing behaviour can be simulated quite accurately too, by means of parasitics extraction and back annotation. Good approximations of the circuit's power-consumption characteristics can be obtained by transistor-level simulations.

As it has already been said, creating HDL implementations takes more engineering effort than implementing high-level models. However, a compromise can be found by the following approach: Only the datapath of the system is modeled on HDL level. Control signals are generated by scripts, created by the high-level model and interpreted by the HDL simulator. For design-space exploration this approach seems well suited. As a rule of thumb the controller of a circuit consumes approximately 10% of the overall resources. For comparisions between different implementation variants this approximation is good enough and the considerable effort to implement the controller on HDL level would not pay off. Therefore this approach is followed for the most part of the remainder of this thesis.

6.1.3 Design-Flow Tools

The HDL used in the course of this thesis is *Verilog*, as standardized by [IEE01]. Tools from *Cadence Design Systems, Inc* were used to compile, simulate and synthesize HDL implementations: `ncsim` for simulation and `PKS Shell` for synthesis.

6.2 High-Level Modelling and Evaluation Framework

When it became clear that several different variants of the same algorithm would have to implemented and evaluated, the decision was made to embed all these variants in a common *high-level modelling and evaluation framework* (hereafter: "framework"), to ensure consistency and comparability between the variants, and to minimize redundant code. This section will give an overview of the framework that has been developed, and present its most important features.

6.2.1 Development Environment

One of the first design decisions in the course of this thesis was to implement high-level models and the corresponding framework in the *Java* programming language. There are numerous reasons supporting this decision:

- *Platform independence:* Since Java programs are executed within a virtual machine, they are platform independent. The implementation of the high-level models was done on a local workstation. However, they were later on executed on the server hosting the design-flow software from Cadence. Since the server and the local workstation use different operating systems, the platform independence of the high-level models was particularly convenient.

- *Object-oriented programming:* Java is an object-oriented language. Some concepts of object-oriented programming, such as inheritance, came in very handy to minimize redundant code during the implementation of several different variants of the same algorithm.

- *Eclipse:* The open-source development platform Eclipse provides many features which considerably ease a Java programmer's work. In particular it contains a versatile debugger which proved invaluable during the course of this thesis.

- *Arbitrary-precision arithmetic:* Java comes with an extensive library of built-in classes, including classes for arbitrary-precision arithmetic. The `BigInteger` class for example enables basic arithmetic operations (addition, multiplication, modular reduction,...) of integers of arbitrary bit-size.

6.2.2 Class Hierarchy

Models

The starting point of the whole framework is the abstract class `GPSModel`. It provides a common interface to all models for other parts of the framework. This interface includes methods to query the model for parameters sizes, simulation results, and performance estimates. There are two direct subclasses derived from `GPSModel`: `GPSModelZn` and `GPSModelEC`. `GPSModelZn` is the base class for all models that operate on groups $\mathbb{Z}_n^*$, and `GPSModelEC` is the base class for all models that are based on elliptic-curve cryptography. Each of these classes structures the computations necessary for protocol execution, and provides interface methods to set and get parameters and operands.

The first "models" which have been implemented were purely functional reference models: `ZnReferenceModel` and `ECReferenceModel`. They do not model any architectural

details. As the name suggests their sole purpose is to perform all necessary calculations and store the results, thereby serving as a reference for comparison for other models.

Details about the actual high-level models, which are subclasses of either `GPSModelZn` or `GPSModelEC` will be presented in §6.3.

Other Classes

Besides the classes which contain different models, there are also some other classes in the framework. Some of them merely provide utility methods such as for example conversion between `BigInteger` instances and `BitSet` instances[1], or random number generation. Others facilitate important functionality like actually performing simulations of the models, possibly iterating over different parameter sets.

6.2.3 Performance Estimations

One of the main purposes of the framework is to evaluate the performance estimations of different variants, to decide which variants are worth implementing in HDL. Two performance measures can be estimated: Area requirements and number of clock cycles to perform an authentication. Power consumption cannot be estimated directly. However, assuming a more or less balanced switching probability of all nodes in the circuit, power consumption correlates linearly with area. Therefore relative comparisons of power consumption can done based on area estimations in first approximation.

The estimation of the number of clock cycles which are necessary to complete an authentication is done in the following way: The `GPSModel` class keeps track of a *cycle counter*. All subclasses can increment this counter. Whenever an operation, such as for example a digit-level multiplication, is modeled, the cycle counter is incremented accordingly by the respective piece of code.

To estimate area requirements `GPSModel` keeps records of the number of *gate equivalents* of all its components. Gate equivalents are a measure of area which is used to compare technologies of different minimum feature sizes. The area of a NAND gate with two inputs corresponds to 1 gate equivalent (GE). Other standard cells which are used in the datapath are ascribed a number of gate equivalents according to the ratio of their area and the area of a NAND gate. This work used values averaged over the three CMOS technologies which have been used, to determine the number of gate equivalents for all major standard cells. The values are presented in table 6.1.

Each piece of code that corresponds to a particular datapath component adds a respective entry to the records of `GPSModel`. For example a k bit bit-serial multiplier could add the following: k NAND gates and k inverters for partial-product generation, and k full-adder cells and k flip-flops for accumulation. `GPSModel` does not only store the total GE estimation. Instead it stores the type of cell, the number of instances and a short comment for each entry. Thus it is possible to later on check what parts have been taken into account for the overall sum.

6.2.4 Parameterization

Another important aspect of the framework is that it allows models to be parameterized. Parameters like for example the bit size of the secret key, the bit size of the modulus, the

[1] The `BitSet` class is used extensively throughout the entire framework to resemble numbers in binary representation, especially in the context of bit-serial architectures.

	Technology						
	AMS 0.35μm		UMC 0.25μm		UMC 0.13μm		
Cell	Area	Ratio	Area	Ratio	Area	Ratio	**GE**
NAND (2 Inputs)	55	1.000	23.76	1.000	5.184	1.000	1.000
NOR (2 Inputs)	55	1.000	23.76	1.000	5.184	1.000	1.000
Inverter	36	0.655	15.84	0.667	3.465	0.668	0.663
Full-Adder	273	4.964	126.72	5.333	29.376	5.667	5.321
XOR (2 Inputs)	127	2.309	55.44	2.333	13.824	2.667	2.436
MUX (2 : 1)	109	1.982	55.44	2.333	12.096	2.333	2.216
MUX (4 : 1)	237	4.309	142.56	6.000	32.832	6.333	5.547
Flip-flop	273	4.964	150.48	6.333	36.288	7.000	6.099
Flip-flop (reset)	310	5.636	190.08	8.000	41.472	8.000	7.212
Flip-flop (enable)	328	5.964	174.24	7.333	41.472	8.000	7.099
Flip-flop (enable *and* reset)	346	6.291	221.76	9.333	51.840	10.000	8.541

Table 6.1: *Calculation of the gate equivalent for all major standard cells. For each technology the ratio between the cell's area and the area of a* NAND *gate in the same technology is determined. The gate equivalent* (GE) *is the average of the three ratios. All areas are given in square microns* [μm^2].

number of coupons a model can store, etc. can all be varied without having to change the code of the models. That way it is easily possible to run simulations with different values for the parameters and investigate the influence on performance estimations.

It should be noted that all $\mathbb{Z}_n^*$-based models use a pseudo-random number generator to select their operands, based on the specifications given by the parameters. Pseudo-randomness is important in this context, because it allows repetition of simulations with the same values. Using a real source of randomness would mean that simulations would be irreproducible. That would complicate debugging very much.

ECC-based models do not select all internal values at random. The EC domain parameters (curve parameters, field modulus, base point) are fixed. This ensures that secure parameters are used. The values for these parameters were taken from [Sta00].

6.2.5 Output and Script Generation

The framework is also capable of producing different forms of output. The most important ones are summaries of performance estimations, simulation transcripts, and scripts for HDL simulations. Instead of simply writing such outputs to the standard output stream, the framework allows different output streams to be set for each kind of output. These streams can point to different files. This is especially useful for generating script files, which are picked up later by the makefile-based design-flow [IAI].

6.3 Architectures and Implementations

This section will present architectures and implementations of models which have been created during the course of this thesis. Different variants will be discussed more or less chronologically, in the order in which they have been created. Interesting observations and problems which turned up during the implementation phase will be addressed along the way.

6.3.1 Assumptions and Premises

Several assumptions and premises apply to all the models presented below. For example all implementation which are discussed in this section are $\mathbb{Z}_n^*$-based variants, conforming to figure 4.1, with one exception: The subtraction in the last step is replaced by addition, as outlined in §4.1.2.

The models aim at presenting different approaches only. Some of them will only model the datapath of the circuit, because the influence of the controller on resource requirements is estimated to be approximately 10% of the overall circuit in any case. Furthermore components such as the pseudo-random number generator to select a random exponent or the (optional) hash function will be completely disregarded.

It should also be noted in the beginning that the implementations which will be presented do not always exhaust all possible optimization options. This work focuses on describing several approaches to GPS implementations, and not on optimizing each of them to the maximum extent.

Another common property is that all models use an 8-bit data interface. All data inputs (keys, random values, challenge) and outputs (commitment, response) are transfered in 8-bit blocks. The rationale behind this assumption is that 8-bit interfaces are very common for buses between different modules (NVRAM, analog front-end, cryptographic module; cf. figure 1.3) of RFID tags.

Furthermore the models will focus on the calculation of the commitment, in a way suitable for the coupon-recalculation approach, which was presented in §4.2.3. That means that the time which is necessary to complete the operation will be (almost) disregarded for design decisions. To the best of the author's knowledge this work is the first attempt to find area- and power-efficient architectures, conforming to the fierce constraints of passive RFID devices, but still capable of performing coupon recalculation, by disregarding execution time.

Neglecting execution time has two different effects on design decisions: First it means that the length of the critical path is more or less irrelevant. The clock frequency can simply be lowered until the clock period is longer than the critical path's delay. Lowering the clock frequency has the convenient side effect of reducing (dynamic) power consumption as well (cf. equation 5.2). Second it means that the number of clock cycles is also not so relevant. Therefore the circuit size can most probably be reduced by trading area for time. This can be achieved by conducting certain operations sequentially instead of parallel. It is obvious that this also reduces the power consumption, because less concurrent operations mean less switching activity during one clock cycle.

It should be noted that a long critical path increases the probability of glitches and therefore also the power consumption (cf. 5.5.3). However the main factor that prolongs the critical path of the GPS architectures presented in this section is the ripple-carry adder. Since the output of this adder is (almost always) directly connected to a register, the effect of glitches is not as bad as it might look on first sight. The fan out of the full-adder cells is very low: Their output is connected to the input of one flip-flop only. Therefore glitches cannot propagate and their power consumption is limited.

Parameters

If no other values are given the following parameters were used for simulations and estimations:

- Bit size of modulus: 1024
- Bit size of secret key: $\sigma = 160$
- Bit size of random value r: $\rho = 264$
- Bit size of challenge: $\delta = 20$

6.3.2 First (Simple) Implementation — Implementation A

The first attempt to implement GPS was completely straightforward. It does not implement modular exponentiation. It is based on complete coupons as outlined in §4.2.1. The only operations which are performed by this implementation are the multiplication and the addition during the response calculation. These operations are performed by a simple bit-serial multiplier and a ripple-carry adder. Using a ripple-carry adder for rather large operands might seem weird at first glance, but since execution time is disregarded it is the simplest and most efficient solution.

It should be noted that the multiplier and the adder do not share any resources. That means that the multiplier uses its own internal adder to accumulate partial products. Certainly this is something that could be improved.

This variant (hereafter referred to as "implementation A") has been implemented as high-level model and in HDL. It has been created mainly to quickly try out the design-flow and the interaction of different tools, and to get a comparison between high-level estimations and synthesizer outputs. Therefore it will not be discussed in great detail.

Special Features

An interesting feature of this model is that due to the fact that it is completely parameterized it allows to study the effect of parameter changes on the total amount (=number of bits) of NVRAM necessary for coupons and domain parameters. However it is difficult to estimate the size of the NVRAM in gate equivalents, since it is usually implemented by hard-macros provided by the manufacturer. In the absence of a better estimate one enable-type flip-flop was estimated for each bit of NVRAM. This is a rather immoderate estimate. Actually NVRAM can be implemented much more efficient. However, the estimate of the size of the NVRAM plays a rather insignificant role for the rest of this thesis anyway. For the most part the share of the NVRAM will be taken out and only the remaining circuits will be compared.

Another fact worth mentioning is that in contrast to the other models this one includes its own controller. Bit-serial multiplication and subsequent addition does not depend on any external control signals. However, concerning loading of operands and reading results, the model behaves as a slave module which needs to be controlled by a master.

Results

The high-level model of implementation A accounted for 116186 gate equivalents (GE). This estimate includes the simple multiplier and adder, as well as the NVRAM for 10 coupons, the private key, and the public key. Using the standard parameters stated above (§6.3.1) 14064 bits of NVRAM are necessary. That corresponds to 99840 GE. That means that the actual arithmetic components require approximately 16346 GE. This figure is

interesting because the NVRAM is not modelled in the HDL model. Thus the synthesizer output can only be compared to this figure and not to the overall GE count.

Table 6.2 shows the synthesis results of the HDL model of implementation A. When multiplying the area of one NAND gate (1 GE) in the respective technologies with the estimated gate count of 16346 it can be seen that the predictions of the estimations are quite accurate. Some deviation was to be expected, due to the fact that the synthesizer is able of performing some optimisation on one hand, and on the other hand that the high-level estimation did not take into account all control components.

The longest combinational path is also reported by the synthesizer. The values show that the circuit could be clocked with several megahertz without causing timing violations.

Technology	**Area** $[\mu m^2]$	**Critical Path** $[ns]$
AMS 0.35 **[aus]**	791718.20	98
UMC 0.25 **[UMCb]**	373721.03	76
UMC 0.13 **[UMCa]**	87174.14	35

Table 6.2: *Comparison of synthesis results of implementation A for different CMOS technologies.*

Since this implementation is neither very efficient nor sophisticated its results will not be discussed in any more details. As it has already been pointed out, this implementation's main purpose was to establish the design- and tool-flow, and to investigate how well the high-level estimations match the results from synthesis.

6.3.3 Adding Commitment Calculation — Implementation B

§4.2.3 introduced the idea of recalculating coupons during idle time of the tag. To implement this approach the tag must be capable of performing a modular exponentiation. Remember, the commitment x is the base g raised to the power of the random value r, reduced modulo n: $x = g^r \bmod n$. "Implementation B" is the first attempt to add commitment-calculation capability to the GPS module. It has been implemented within the high-level framework only, because its implementation suggested several possible improvements. Therefore it did not seem worth the effort to do an HDL implementation of this variant. Instead the improved version which will be discussed in §6.3.4 was implemented on HDL level.

It should be noted that although this implementation and the subsequent ones are intended to be used with the coupon-recalculation approach, they do not incorporate any estimations about NVRAM necessary for the coupons. This is due to the fact that the amount of NVRAM is mainly dependent on the total number of coupons. Since this parameter is particularly application-dependent, there is no point in distracting the attention from the resource estimates and results of the architecture itself.

Architecture

The basic architecture of implementation B is depicted by figure 6.1. The operands are stored in shift registers which shift 8 bits per clock cycle. Thus they can easily be filled with data coming from the 8 bit input bus. These registers are clock-gated. That way they do not have to be enable-type flip-flops, which saves considerable area resources. In general the use of shift registers is discouraged in low-power implementations, because all

flip-flops might change their value simultaneously during shifting, which causes significant power consumption. However, when using bit-serial multiplication, clocking (and changing the values of) all flip-flops of a register simultaneously is required anyway (for example during partial-product accumulation). So if there is not enough power to do that, the whole approach would not work anymore. Assuming that there is enough power to change the values of all flip-flops of one register at the same time, shift registers become the obvious choice, because they do not require any additional overhead such as address logic.

The register r which stores the exponent has a second shift mode which allows this register to be shifted 1 bit per clock cycle as well. This is necessary to execute the square&multiply algorithm (cf. algorithm 4). To achieve these two different shift modes, each flip-flop in register r is prepended with a multiplexer. This is not shown explicitly in figure 6.1, but it is taken into account in the estimation of the total gate count.

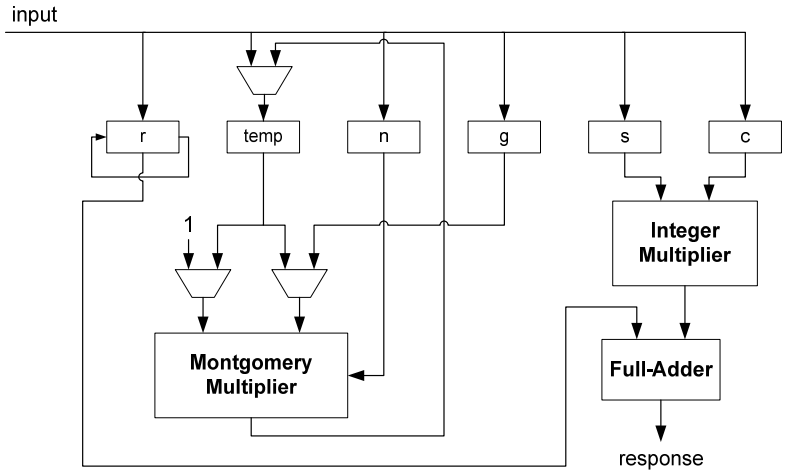

Figure 6.1: *Architecture of implementation B. The operands are stored in shift registers which are directly connected to the arithmetic components.*

The inputs of the Montgomery multiplier are prepended with multiplexers. Thus it is possible to select different operands: Either the *temp* register is squared, or multiplied by the base g. One input of the Montgomery multiplier can also be set to the constant value 1. This is necessary for converting the final result of the exponentiation back to the integer domain (cf. 3.4.5).

The Montgomery multiplier is implemented in a bit-serial way, as outlined by algorithm 7 (page 37). The integer multiplier is also implemented in a bit-serial way. However, several improvements have been made compared to the version used in implementation A. For example this integer multiplier does not store both its operands in flip-flops, but only the one it needs to shift. The other one is supplied to the multiplier module by means of wires only. Therefore redundant storage space is saved. The same improvement was applied to the Montgomery multiplier.

It is interesting to note that there seems to be a trade-off between removing redundant components and modularity. In this instance it was possible to remove redundant flip-flops which would have stored the same values as some other flip-flops, but the price was that the multiplier is not a self-contained module any more. Its functionality relies on the fact that the values of some of its input wires do not change during its operation. Thus

the module is not as reusable any more. Furthermore some other optimizations which have been made during the course of this thesis made it increasingly harder to divide the architectures into distinct HDL source files, because the interdependency of components grew larger with almost each optimization.

Nevertheless the integer multiplier was additionally improved by utilizing the fact that its operands are not of the same size. By shifting the operand with the larger number of bits and accumulating partial products of the operand with the smaller number of bits, it is possible to manage with fewer full-adder cells. Thereby the area demands and the power consumption are reduced. The price of this optimization is that multiplication will take more clock cycles than if the operands are interchanged.

Results

The high-level model of implementation B gave an estimation of approximately 80884 gate equivalents. That estimate includes 2208 bits of NVRAM for the public key, the secret key, and the base g (converted to the Montgomery domain). Excluding the estimate for the NVRAM, implementation B requires approximately 65210 GE. The reason why this figure appears relatively small compared to the results of implementation A is that no memory for storing coupons is included in this estimate.

Simulations of the high-level model also revealed that one complete authentication (commitment calculation and response calculation) takes approximately 406000 clock cycles. This is just an average number, since the actual value depends on the exponent; the more 1-bits the exponent contains, the more multiply operations are necessary during execution of the square & multiply algorithm, and the more clock cycles are necessary.

6.3.4 Improved Commitment Calculation — Implementation C

When looking at the components of implementation B, it is obvious that there are many redundancies. The integer multiplier and the Montgomery multiplier for example both have the ability to generate and accumulate partial products. Moreover both have some kind of shifting mechanism for their accumulator. Since the integer multiplier and the Montgomery multiplier are not required to operate concurrently, it stands to reason to remove these redundancies by designing a new arithmetic unit capable of performing both integer multiplication and Montgomery multiplication.

Architecture

In order to design a custom-built arithmetic unit it is necessary to analyse what kind of operations are required first. This analysis will be done starting from the accumulator, or “result register”. Obviously this register must be shiftable (1 bit to the right). This shift operation is required during Montgomery multiplication. Moreover partial product accumulation and conditional modulus addition must be realized. These two operations are actually the same: The value of an operand register is conditionally added to the result register. In case of partial product accumulation the value which should be conditionally added is the multiplicand, and the condition is given by one bit of the multiplier. In case of conditional modulus addition the value to add is the modulus, and the condition is given by the least significant bit of the result register. Thus both these operations can be realized by means of masking (utilizing AND gates) and addition.

The lower part of figure 6.2 shows the realization of this idea. The registers 0 and 1 are the *operand registers* of the arithmetic unit. Register 2 is the result register. The thick lines symbolize full-precision buses[2]. The thin lines in figure 6.2 represent single bit wires. There are also a few signals composed of a few bit, namely the select signals of the multiplexers.

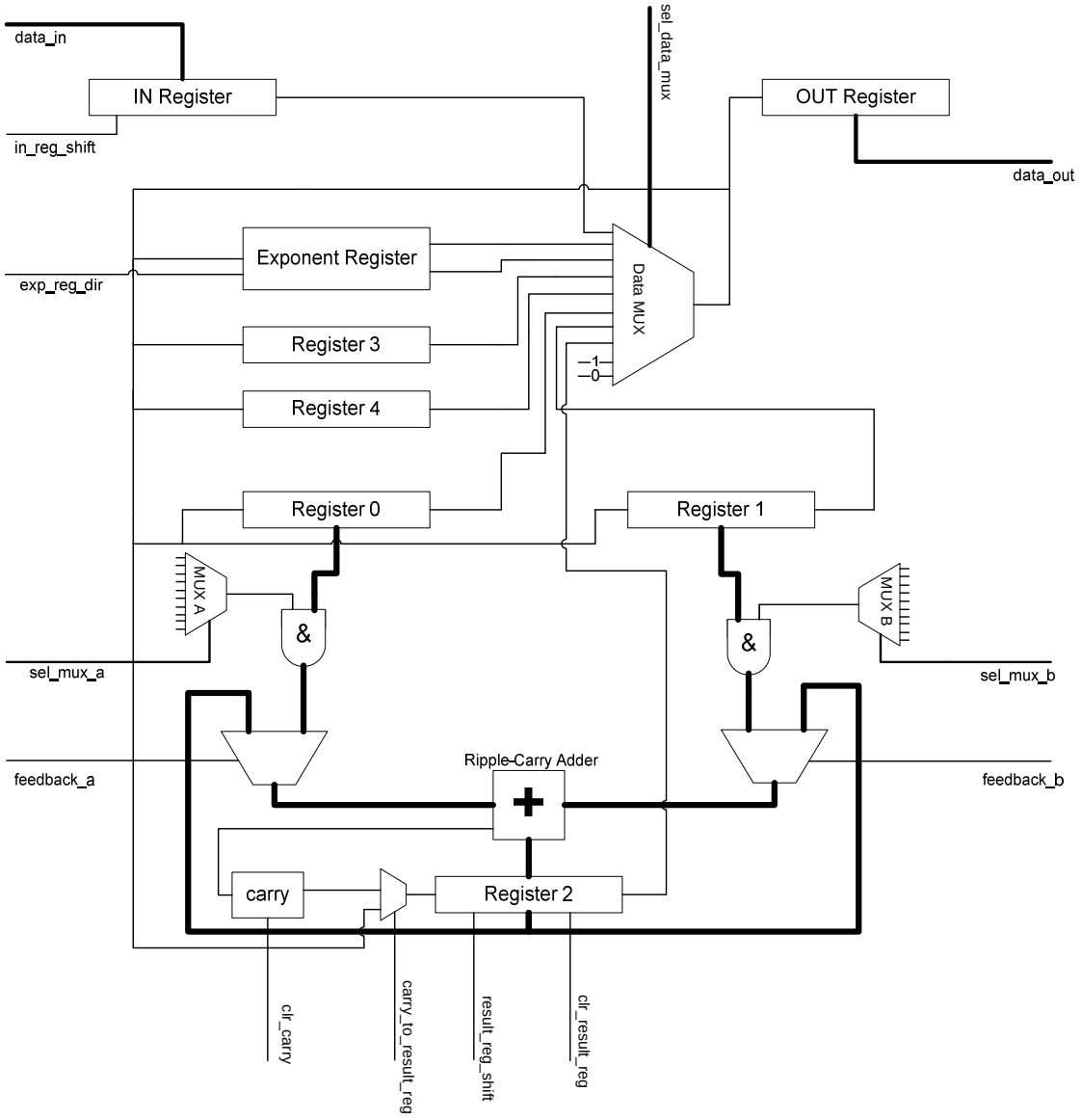

Figure 6.2: *Architecture of implementation C. A custom-built arithmetic unit is used to avoid unnecessary redundancies.*

The remaining parts of figure 6.2 are easily explained. Two more (full-precision) registers are necessary. One for the base g and one for a temporary variable which is necessary

[2]That means that these buses have the same width as the registers themselves. This width is 2 bit larger than the bit-size of the modulus to allow using Gueron's enhanced Montgomery multiplication (cf. 3.4.5, or [Gue03] respectively).

during square & multiply. All these registers are shift registers which can be shifted 1 bit to the right. The reason for this will be given below. The least significant bit of each register is accessible. One might wonder why parameters like the base g or the modulus n, which are constant throughout an entire domain of users, are stored in registers instead of being fixed during synthesis. There are two reasons for doing so. The first is that for bit-serial multiplication it must be possible to access each single bit of the values one after another. Shift registers provide a very easy way of achieving that. If the values were fixed there would be the need for some sort of address logic to access single bits. Furthermore if the values were fixed it would not be possible to reuse the registers for other operations, such as for example the integer operations in the last step (response calculation) of the GPS authentication scheme.

In addition to the registers 0 to 4 there is an exponent register which is smaller than the other registers. Its size corresponds to the parameter ρ, the size of the random exponent r. This register can be shifted in both directions, and its least significant *and* its most significant bit are accessible.

In addition to that there are the following components: An input register `IN`, an output register `OUT`, and three multiplexers. The `IN` and `OUT` register are both 8 bits wide. They are both shiftable (1 bit to the right). The `IN` register supports parallel load. The `OUT` register has a parallel output. These two registers are the data interface which performs conversion between the external 8-bit-parallel data-transfer mode and the internal bit-serial data-transfer mode.

The multiplexers are all connected to the same inputs, however these connections are *not* drawn explicitly for `MUX A` and `MUX B` in figure 6.2 for the sake of clarity. Nevertheless these two multiplexers are supposed to be connected to the same inputs as the `Data MUX`. The following signals are inputs to the multiplexers:

- The LSBs of the registers 0–4.
- The LSB and the MSB of the exponent register.
- The LSB of the `IN` register.
- Constant 1.
- Constant 0.

It should be noted that all registers in the circuit are clock gated. Thereby it is ensured that they only store new values when they are explicitly activated. Thus their flip-flops do not need to be of enable-type, and, although they are shift registers, the registers 0, 1, 3, and 4 do not need any multiplexers to retain their values for several cycles.

It is also important to note that register 2 has two modes of operation: It can either load a full-precision value coming from the adder, or it can operate as a shift register, just like the other registers. To achieve that multiplexers are prepended to each flip-flop of the register.

Mode of Operation — Loading, Copying, Output

The circuit works in the following way: First operands are loaded to the registers. To do that the first byte of the value to load is stored to the `IN` register. The `sel_data_mux` signal sets the data multiplexer in a way that it lets the LSB of the `IN` register pass

through. Thus this bit is visible at the input of all shift registers. The clock gating signal corresponding to the desired destination register is activated and thereby the 8 bits from the `IN` register are shifted into the destination register. Then the next byte is stored to the `IN` register, and so on. In that way all registers can be loaded to their initial values. If two or more registers shall be initialized to the same value they could be loaded concurrently. However, this would consume more power and has therefore not been implemented.

Note that the content of one register can be copied to another register easily. To do so both register's clock-gating signals are enabled, and the `sel_data_mux` signal is set so that the `Data MUX` lets the LSB of the source register pass through. After k clock cycles (where k is the size of the registers) the source register is back at its original value, as it has performed exactly one complete circular shift cycle, and the destination register now holds the same value as the source register. If one register were to be copied to multiple destinations, this could be achieved by simply enabling multiple (destination) registers at the same time. However, this would also consume more power than sequential copying.

To output the value of a register via the 8-bit output-interface, the register's clock-gating signal is enabled and the least significant 8 bits are shifted into the `OUT` register, which must also be enabled. The select signal of the `Data MUX` must be set to let the data from the register in question pass through. After all 8 bits have been shifted into the `OUT` register, the value is visible to the outside world, and can be copied by an attached module. As soon as this is complete, the next 8 bits can be shifted to the `OUT` register.

Registers can also be cleared, by setting `sel_data_mux` in a way that the constant input 0 is passed through the `Data MUX`. The clock-gating signal corresponding to the register to clear is enabled and after k clock cycles the register as been cleared. This can be applied concurrently to as many registers as desired, at the cost of increased power consumption. Register 2, the result register, must be cleared more often than all other registers. Therefore it stands to reason to use some additional logic circuitry to allow setting all the flip-flops of this register to 0 simultaneously.

Mode of Operation — Exponentiation

The actual exponentiation $x = g^r \bmod n$ works as follows: The modulus n is loaded to register 1, which remains unchanged during the entire operation. The exponent is loaded to the `Exponent` register. The base g is loaded to the registers 0, 3, and 4. Register 3 will retain the value of g during the entire operation, while register 4 serves as temporary variable for the square & multiply procedure.

The first operation in the loop of algorithm 4 is a square operation. So register 0 is multiplied by register 4, which both hold the base g in the beginning. The result g^2 is stored to register 2, the result register.[3] Then the result is copied to the registers 0 and 4. As it has been said in the previous section, this can either be done concurrently or sequentially, depending on how much power one is willing to spend.

The next operation of algorithm 4 is a conditional multiply operation, where the condition is given by the current MSB of the exponent register. Suppose the exponent's MSB is 1. Then the temporary variable (stored in register 4) must be multiplied by the base g (stored in register 3). After this multiplication has finished, register 2 holds the result. This result is subsequently copied to the registers 0 and 4. After that the exponent register is shifted one bit to the left, followed by the next square operation, followed by the next

[3]For the sake of brevity the appendix "mod n" is not written explicitly in this section when referring to powers of g.

conditional multiply operation, and so on. Table 6.3 shows the register contents during this procedure.

Operation	Register 0 (Operand Register)	Register 1 (Modulus Register)	Register 2 (Result Register)	Register 3 (Base Register)	Register 4 (Temporary Register)
Initialization	g	n	–	g	g
Square (0×4)	g	n	g^2	g	g
Copy $(2 \rightarrow 4)$	g	n	g^2	g	g^2
Copy $(2 \rightarrow 0)$	g^2	n	g^2	g	g^2
Multiply (0×3)	g^2	n	g^3	g	g^2
Copy $(2 \rightarrow 4)$	g^2	n	g^3	g	g^3
Copy $(2 \rightarrow 0)$	g^3	n	g^3	g	g^3
Square (0×4)	g^3	n	g^6	g	g^3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 6.3: *Register contents during exponentiation. Each line shows the contents of the registers* after *the operation stated in the first column has been executed. The numbers in brackets designate the registers which are multiplied or copied respectively.*

Mode of Operation — Montgomery Multiplication

The previous section showed how the architecture depicted by figure 6.2 can be used to perform modular exponentiation by employing Montgomery multiplication. This section will explain how one single Montgomery multiplication can be realized with this architecture. As it was already outlined, the modulus n is stored to register 1. The multiplicand is stored to register 0. The multiplier can be stored either in register 3 or in register 4. The result will be placed in register 2.

Multiplication is performed bit-serial, according to algorithm 7, but using Gueron's enhancement, instead of the final conditional subtraction of the modulus (cf. [Gue03]). After the operands have been loaded and the result register (register 2) has been cleared the next step in algorithm 7 is to add a partial product to the current value of the accumulator (register 2) and store the result of the addition. This is achieved by setting the `feedback_b` signal in a way that the current content of register 2 is fed back to the ripple-carry adder, whereas `feedback_a` is set to pass the output of the AND gates through to the adder. The signal `sel_mux_a` is set to let the LSB of the register containing the multiplier (either register 3 or 4) pass through `MUX A`. That way this bit controls whether or not the multiplicand in register 0 can pass through the AND gates.

The next step is the conditional addition of the modulus. To achieve that `feedback_a` is set to feed back the current value of register 2 to the adder, while `feedback_b` is set to let the signal coming from the AND gates pass. `MUX B` is set to let the LSB of register 2 pass through. Thus this bit control whether or not the modulus is passed through the AND gates. After that the result register is shifted one bit to the right. That corresponds to the division by 2, as outlined by algorithm 7.

Then the register holding the multiplier is also shifted one bit to the right. During this shift operation the `Data MUX` is set to let the LSB of the shifted register pass through. That way the bit which is shifted out on the right side is shifted back in at the left side.

Therefore the content of the multiplier register seems unaltered after multiplication is complete, because it has undergone one complete circular shift cycle.

It is interesting to note that shifting the result register and shifting the multiplier register could be done concurrently, because the result register obtains its input value from the `carry` register. Doing so would save a significant amount of clock cycles, but at the expense of increased power consumption.

Mode of Operation — Integer Multiplication

During the response calculation step of the GPS algorithm a non-modular integer multiplication must be performed. There are several ways how this can be realized with the architecture of implementation C. One way is for example to perform a modular multiplication and rely on the fact that the result is so small that no modular reduction actually occurs. For the usual GPS parameters suggested by literature this would be the case: A 1024 bit modulus compared to a 180 bit result from the multiplication of a 160 bit secret key with a 20 bit challenge. However, this approach necessitates converting the operands to the Montgomery domain, and converting the result back to the integer domain. This is undesirable since it takes a significant amount of time, especially relative to the time of the actual multiplication. Furthermore the conversion and also the multiplication itself would take much longer than necessary. This is due to the fact that they would have to be performed as if the operands were of the same bit size as the modulus. However, since they are much smaller faster implementations would be possible.

Mind also that unlike the commitment calculation the response calculation must be done "online" and is therefore bound by application-specific timing constraints. Therefore it is undesirable to waste time during this calculation.

Fortunately there are also other ways to implement integer multiplication based on the architecture of implementation C, if the operands fulfill certain size constraints. One operand is loaded to the exponent register, since this register can be shifted in both directions. It is pre-shifted so that the operands MSB is at the exponent register's MSB position. The other operand is loaded to register 0. Register 0 is then circularly shifted to the right until its value has effectively been shifted α bits to the left, where α is the bit size of the other operand. In order to make this possible the operand must at least be α bits smaller than register 0.

Then the actual multiplication starts: Partial products are generated by letting the exponent register's MSB pass through `MUX A`, and feeding back the current value of the result register by setting `feedback_b` accordingly. After that the exponent register is shifted one bit to the right, and register 0 is shifted one bit to the left. Then the next partial product is accumulated in the same way. This is repeated until all bits of the operand in the exponent register have been taken into account. To speed up the multiplication it is preferable to place the smaller operand in the exponent register and the larger one in register 0, if the operands are not of the same size. Thus for a 20-bit challenge and a 160-bit secret-key the actual multiplication (not taking the loading and pre-shifting into account) would only take 20 cycles.

Mode of Operation — Integer Addition

For the sake of completeness it should be noted that the architecture of implementation C is of course also capable of performing integer addition in a straightforward way: The

operands are loaded to registers 0 and 1, `MUX A` and `MUX B` are both set to let their constant 1 input pass through, and both feedback signals are disabled.

Simulation Issues

The high-level model of implementation C simulated its operation by employing functions on different abstraction levels. First the most primitive operations, such as shifting of a single register, or determining the output of the arithmetic unit (depending on the value of the signals `feedback_a`, `feedback_b`, `sel_mux_a`, and `sel_mux_b`) were implemented. More complex operations such as loading a complete full-precision value, or adding two integers, were composed of the more primitive operations. Based on that, even more complex operations, such as modular exponentiation were implemented.

As it has been mentioned before, the HDL model of this implementation only modelled the datapath of the circuit and not the controller. To test the circuit anyway, it was controlled by simulation scripts. The basic idea was to use the high-level model to create these scripts. During simulation of the high-level model each primitive function was supposed to output a call to a corresponding function of the script language to a script file. That way only the most primitive functions would have needed to be implemented in the script language, which would have been very convenient, because the script language does not offer as many nice debugging features as Java and Eclipse do. Furthermore it was planned that after each primitive operation the high-level model would create script code to compare the contents of its own registers to the contents of the HDL model's registers. That way it would have been verified that both models actually perform exactly in the same way; and if there would have been differences, it would have been very easy to precisely locate the error.

However unfortunately this idea did not work out. The script files which would have been produced would have been too large. During the first attempt to test this approach the simulation of the high-level model (and the concurrent script-file generation) had to be aborted, after the generated file grew larger than several gigabytes and projections led to the conclusion that the final file size would have exceeded the available hard-disk space. Thus there was no other choice but to implement also some more abstract functions in the script language. By doing so the script-file's size reduced to reasonable values. However, this made debugging the HDL model much more difficult, because now the registers of high-level model and HDL model could only be compared after each complex operation. Therefore errors were much harder to localize.

This problem was made worse by the fact that during simulation only very few internal signals of the circuit could be stored for later analysis. Otherwise the data file grew too large on one hand, and simulation was also significantly slowed down on the other hand.

To give a few numbers: Simulation of one modular exponentiation (1024 bit modulus, 260 bit random exponent; thus approximately 2.4 million simulated clock cycles) using the synthesized HDL model took approximately 12 to 14 hours on a 1.1 GHz Pentium III processor.

Results and Further Remarks

The high-level model of implementation C estimates approximately 65880 GE, including NVRAM for the keys and the base in Montgomery domain representation (2206 bit total). Without the NVRAM the estimate is 50219 GE. That means that implementation C needs almost 15000 GE less than implementation B, due to the improvements discussed above:

Implementation C contains less redundant components. Instead of a separate integer multiplier and Montgomery multiplier a custom-built arithmetic unit was used.

Technology	**Area** $[\mu m^2]$	**Critical Path** $[ns]$
AMS 0.35 [**aus**]	2765581.04	25.20
UMC 0.25 [**UMCb**]	1428015.63	18.16
UMC 0.13 [**UMCa**]	326560.90	24.34

Table 6.4: *Comparison of synthesis results of implementation C for different CMOS technologies.*

Synthesis results are shown in table 6.4. The timing results suggest that the circuit could be operated at frequencies up to approximately 40 MHz. However, high frequencies might not be desirable because of the higher power consumption.

An HDL simulation of implementation C revealed that one Montgomery multiplication takes 12388 clock cycles. Assuming approximately $1.5 \cdot 260$ Montgomery multiplications that leads to approximately five million clock cycles for one commitment calculation. That means that one such calculation would take about 5 seconds at 1 MHz, or about 50 seconds at 100 kHz.

6.3.5 Digit-Level Commitment Calculation — Implementation D

Although a few tweaks and optimizations can still be applied to implementation C, their influence on the overall circuit size and power consumption will be limited. Concerning area resources, the number of flip-flops cannot be significantly reduced any more. A way to decrease the necessary area of the storage elements could be using RAM hard macros which might be more area-efficient than flip-flop-based registers. Such hard macros, however, do not allow direct connections of the storage elements to arithmetic components, as employed in implementation C. Concerning the power consumption, there is a lower bound as long as operations are performed on full-precision values: At least one register has to be clocked in each cycle, which causes a significant number of nodes to switch.

There is yet an obvious way to further decrease the circuit's area and power consumption: Instead of performing the calculations on full-precision values, a digit-level approach could be implemented. This would allow to use RAM hard macros on one hand, and on the other hand the size of the arithmetic components could be reduced to digit size.

Overview of Digit-Level Architecture

The basic schematic of a digit-level GPS architecture is sketched in figure 6.3. The two components of interest are the arithmetic unit and the dual-ported RAM. As outlined in §5.3.4 a dual-ported RAM allows concurrent read and write access. One digit can be written to the RAM at the same time another one is read from the RAM. However, two write operations or two read operations are always done sequentially. The reason for using such a standardized approach is that hardmacros of such dual-ported RAMs exist which allow very (area-)efficient implementation of the RAM.

The width of the data buses shown in figure 6.3 is equal to the digit size of the arithmetic unit. The digit size may vary, approximately between 8 and 64 bits. Larger digit sizes seem impractical, because the corresponding digit multiplier would become to

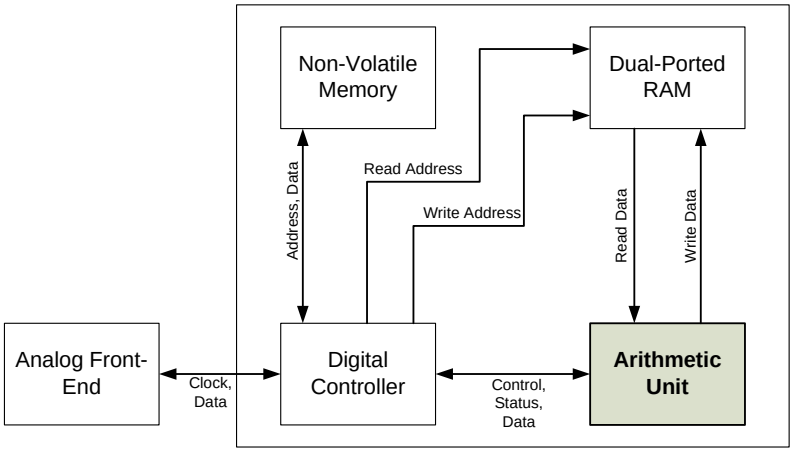

Figure 6.3: *Main components of a digit-level GPS architecture. The remainder of this section will focus on the arithmetic unit.*

large. Digit sizes below 8 bits also seem impractical since that would cause a too large amount of digit operations.

The *arithmetic unit* contains the actual datapath for performing the required operations. The remainder of this section will focus on this component. Different implementation approaches will be discussed and compared, design decisions will be explained.

Analyzing Digit-Level Operations

In order to implement a digit-level arithmetic architecture tailored towards GPS, one must first analyze which digit-level operations it will have to perform. GPS requires modular exponentiation, integer multiplication, and integer addition. Modular exponentiation can be broken down into a series of Montgomery multiplications, by means of the square & multiply algorithm (cf. algorithm 4). So the basic operations which are necessary are Montgomery multiplication, integer multiplication, and integer addition. These operations can be performed on digit level by means of algorithms 8, 2, and 1 respectively.

A digit-level architecture of digit size d would consist of a dual-ported RAM module (with a data bus-width of d bit) and an arithmetic unit. The arithmetic unit would consist of two (or more) *input registers* (of size d) for the operands, and two *output registers* (of size d). Two output registers are necessary, because digit-level operations can produce results which are larger than the digit size (cf. §3.3.3). These two registers will be referred to as H (for "high", or most significant) and L (for "low", or least significant).

When analyzing algorithms 8, 2, and 1, one can extract four different digit-level operations on which they are based:

- *Operation* 1*:* $\langle H, L \rangle = D_1 + D_2 \cdot D_3 + H$
- *Operation* 2*:* $\langle H, L \rangle = D_1 + H$
- *Operation* 3*:* $\langle H, L \rangle = D_1 \cdot D_2 \bmod 2^d$
- *Operation* 4*:* $\langle H, L \rangle = D_1 + D_2 \cdot D_3$

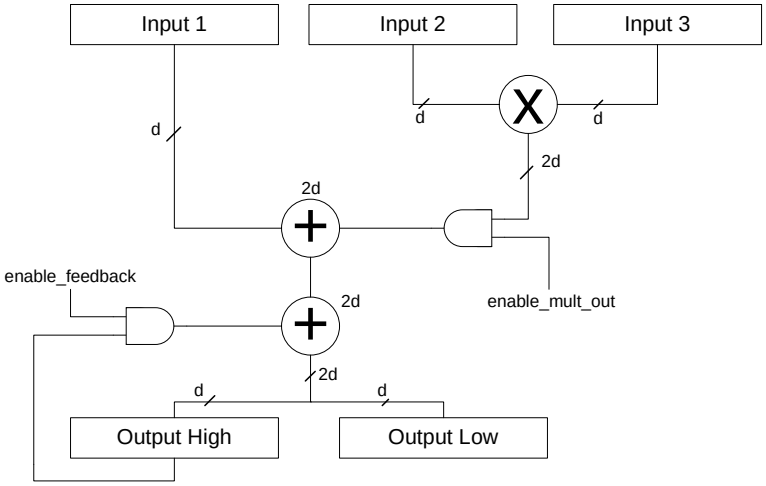

Figure 6.4: Variant 1: *This architecture uses three input registers. It can perform all operations in one clock cycle each.*

D_1, D_2, and D_3 are input digits. There are several different architectures which are capable of performing the aforementioned operations. They differ in their area requirements, their critical paths, and the number of clock cycles they need for each of the four operations. Before presenting some of these variants in the subsequent sections, it is interesting to investigate how often each of the four operations is executed, to get an idea for which operation the arithmetic unit should be optimized.

The modular exponentiation is dominant over the other operations, therefore only algorithm 8 will be analyzed. Suppose the full-precision size of the operands is $k = w \cdot d$, and each operand consists of w digits of size d, then operation 1 is executed $2w^2$ times, operation 2 is executed $3w$ times, operations 3 is executed w times, and operation 4 is executed w times, during one execution of algorithm 8. Furthermore the H register is cleared w times, and there are $2w^2 + 3w$ store operations, during which the content of one of the output registers is stored to RAM.

The subsequent sections will present five different architectures for a digit-level arithmetic unit. Their area and cycle estimations will be compared, as well as their critical path. It should be noted that all architectures contain a pure combinational multiplier of size d (cf. §5.3.2). The area of this multiplier will not included in the estimations, because it is the same for all architectures. Moreover for cycle estimations loading and storing will not be taken into account, because depending on the order of the operations some load and store operations can be done concurrently.

Arithmetic Unit — Variant 1

Figure 6.4 shows the first variant of a digit-level arithmetic unit. Since two of the required operations need three operands, this architecture uses three input registers. The advantage of variant 1 is that it can perform all four required operations in one clock cycle each. The way how the operations can be implemented is straightforward and will therefore not be described in detail.

As a price for the low cycle-requirements this circuit takes up a rather large area. Two ripple-carry adders, each consisting of $2d$ full-adder cells are necessary. In addition to that

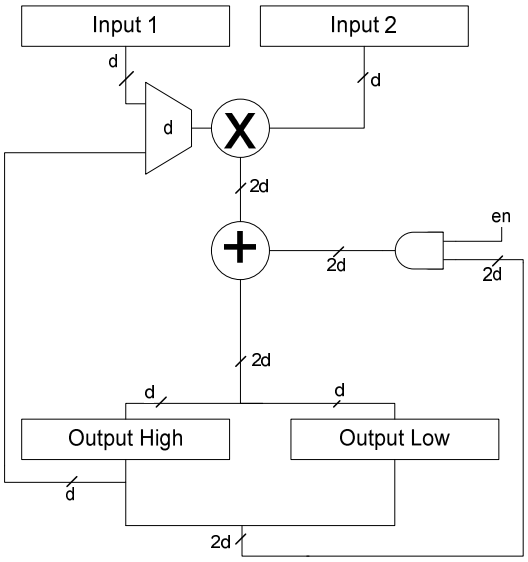

Figure 6.5: Variant 2: *A very slim architecture, in terms of area, which needs more clock cycles to complete operations.*

$3d$ AND gates are required to mask signals if necessary. These AND gates are not absolutely required for the functionality, but they help in decreasing the power consumption, because the serve as sleep logic (cf. §5.5.2). Note that the $2d$ AND gates at the output of the multiplier could also be prepended to the multipliers inputs.

The critical path of variant 1 runs through the multiplier, one AND gate, and two $2d$-ripple-carry adders, which is rather long. Therefore this circuit would not be applicable to higher clock frequencies.

An improvement to this variant would be to replace the two ripple-carry adders with one carry-save adder and one ripple-carry adder. This change does not save any area, but it reduces the critical path significantly, because the delay of the carry-save adder is negligible, compared to that of the ripple-carry adder.

Arithmetic Unit — Variant 2

The architecture of the second variant of the arithmetic unit is depicted by figure 6.5. Since flip-flops consume considerable area this architecture tries to manage with only two input registers of size d, thus saving d flip-flops compared to variant 1. Note that the second input register must be resettable to $(000\ldots00)_2 = 0$ and to $(000\ldots01)_2 = 1$ in order to be able to perform all four required operations. This can be achieved by using additional logic (multiplexer with one input constant $(000\ldots00)_2 = 0$, one input constant $(000\ldots01)_2 = 1$, and one input connected to the data bus from the RAM) prepended to the register. This property applies also to the second input register of variants 3, 3a, and 4.

Unfortunately saving a third input register comes at a price: Operations now take more than one clock cycle. Especially operation 1, which is executed most frequently, takes 3 clock cycles on this architecture. Since for this architecture it is less obvious how the four required operations are realized, one of them (operation 1) will be explained in

detail.

Remember that operation 1 was defined as $\langle H, L\rangle = D_1 + D_2 \cdot D_3 + H$. So the first thing that must be done is "backing up" the current value of the `Output High` register (denoted H_0), because if any other action was taken first, this value would be lost. To do so input register 2 is reset to $(000\ldots01)_2 = 1$, and the multiplexer is set to let the value of H pass through. Since input register 2 is set to 1, the output of the multiplier will be $1 \cdot H_0 = H_0$. The enable signal `en` of the feedback loop is set to 0 so that the output of the adder is also $H_0 + 0 = H_0$. That way the output registers hold the following values after the first clock cycle: $\langle H_1, L_1\rangle = H_0$.

Concurrently to "backing up" H_0, the value D_1 can be loaded to input register 1. In the second clock cycle the multiplexer is set to let the value of input register 1 pass through to the multiplier, where it is multiplied by 1, just like H_0 before. However this time the `en` signal is set to 1 and the current value of the output registers is fed back to the adder. So the output of the adder is $D_1 + H_0$. Thus after the second clock cycle the output register store $\langle H_2, L_2\rangle = D_1 + H_0$.

Then the values D_2 and D_3 are loaded to the input registers. The multiplexer settings remain unchanged. Thus the output of the multiplier will be $D_2 \cdot D_3$. The `en` signal in the feedback loop is also not changed so the output of the adder will be $D_1 + H_0 + D_2 \cdot D_3$, and the output registers will store $\langle H_3, L_3\rangle = D_1 + D_2 \cdot D_3 + H_0$ after the third clock cycle. That is exactly what was required.

Operations 2, 3, and 4 are accomplished in a similar way. One fact worth mentioning is that the modular reduction (mod 2^d) in operation 3 is achieved by simply disabling the clock-gating signal for the `Output High` register, so that only the lowest d bits of the multiplication result are stored.

It should also be noted that the critical path of this architecture is shorter than the one of variant 1. It only runs through the multiplier and *one* $2d$-ripple-carry adder, instead of two.

Arithmetic Unit — Variants 3 and 3a

The most significant drawback of variant 2 is the fact that operation 1, which is the operation executed most frequently, takes 3 clock cycles. Therefore it stands to reason to spend some additional hardware resources to decrease the number of cycles for this operation. Variants 3 (figure 6.6) and 3a (figure 6.7) try to do so.

Compared to variant 2 these variants need d additional full-adder cells and d additional two-to-one multiplexers. In exchange for that operation 1 only requires 2 clock cycles on these variants.

Variant 3a differs from variant 3 only regarding the position of the multiplexer. Variant 3a has a shorter combinational path in the feedback loop. Possibly this implies that it also has a lower power consumption, because less nodes are switching when the value along the feedback loop changes. This assumption is, however, unproven.

Arithmetic Unit — Variant 4

Variant 4 (figure 6.8) tries a more symmetric approach. However, this variant needs d multiplexers more than variant 3 (and 3a), but brings no speed-ups. Therefore it will not be discussed in more detail.

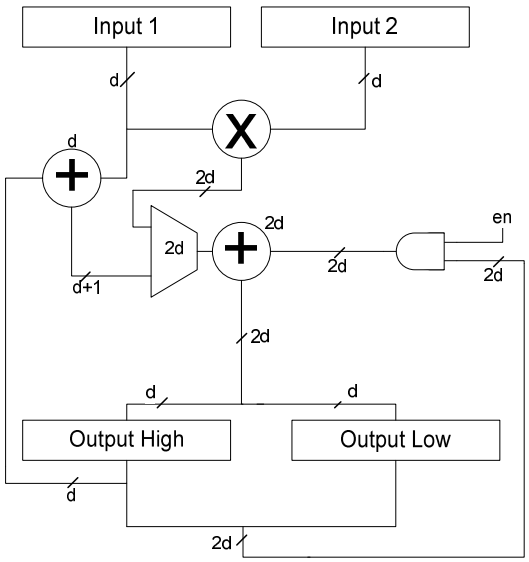

Figure 6.6: Variant 3: *By means of d additional full-adder cells and d additional multiplexers the number of cycles necessary for operation* 1 *can be reduced to* 2*.*

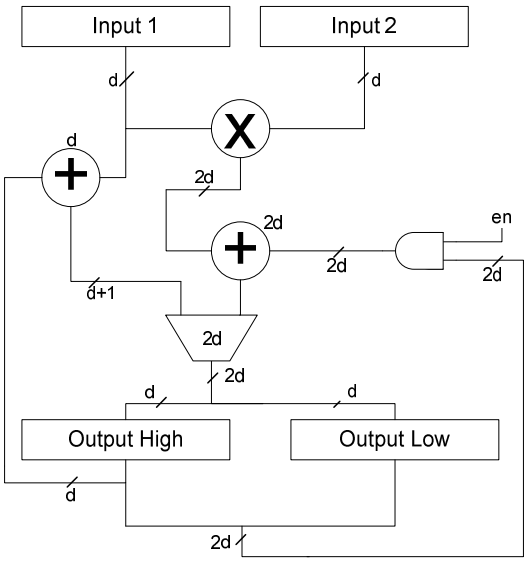

Figure 6.7: Variant 3a: *Almost identical to Variant 3, except for the position of the multiplexer.*

Comparison of Variants

Table 6.5 compares the required area resources of the variants presented above. As it can be seen the area depends on the digit size d. It should be noted that the combinational $d \times d$-multiplier is not taken into account for the estimations, because it is the same in all variants.

Table 6.6 summarizes how many clock cycles each variant needs for each of the four

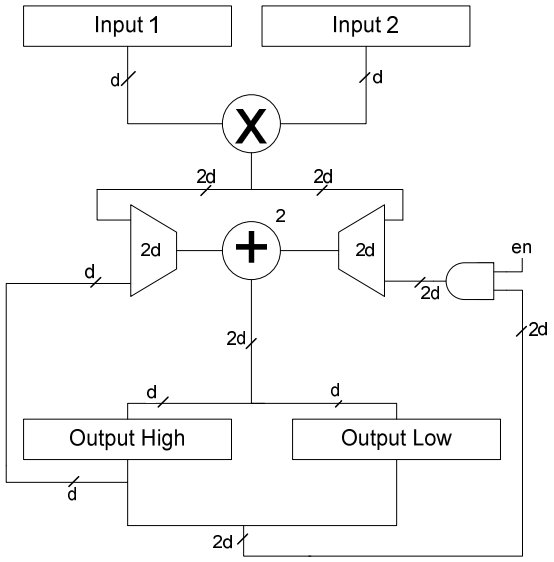

Figure 6.8: *Variant 4*

		Variant				
	GE	**1**	**2**	**3**	**3a**	**4**
Flip-Flops	8.541	$5d$	$4d$	$4d$	$4d$	$4d$
Full Adder	5.321	$4d$	$2d$	$3d$	$3d$	$2d$
AND Gates	1.663	$3d$	$2d$	$2d$	$2d$	$2d$
Multiplexer	2.216	—	d	$2d$	$2d$	$4d$
SUM of GE (rounded to 1 decimal		$69.0 \cdot d$	$50.3 \cdot d$	$57.9 \cdot d$	$57.9 \cdot d$	$57.0 \cdot d$

Table 6.5: *Area estimations of implementation-D variants. The $d \times d$-multiplier is not taken into account, since it is of the same size for all implementations and does not influence the relative comparison.*

operations. It is interesting to note that although variants 3 and 4 differ in size, they require the same amount of clock cycles for operations. Variant 2, which is the smallest variant in terms of area, is unfortunately also the slowest one. However, that was to be expected since area is almost always a trade-off for speed and vice versa.

Following the general design directive to try finding small and power-efficient architectures, disregarding the necessary execution time, variant 2 was chosen to be implemented on HDL level.

Results

The results discussed in this section refer to the second variant of the five different variants that have been presented above. Both the high-level model and the HDL implementation are parameterizable with respect to the digit size. Smaller digit sizes lead to smaller arithmetic units, but the time necessary to complete one authentication increases with smaller digit sizes. To illustrate this correlation table 6.7 shows the high-level estimates

	Variant				
	1	**2**	**3**	**3a**	**4**
Operation 1	1	3	2	2	2
Operation 2	1	2	1	1	1
Operation 3	1	1	1	1	1
Operation 4	1	2	2	2	2

Table 6.6: *Cycle estimations for the four required operations on all variants that have been presented. Loading from and storing to RAM is not accounted for in these estimations.*

for two different digit sizes.

It should be noted that these estimates do not include the $d \times d$ multiplier. The reason for that is that the high-level model does not consider the architecture of this multiplier. It is implemented as a combinational function in HDL. The implementation details are left to the synthesizer. That is why the numbers for the area in table 6.7 are in the same order of magnitude. The size of the multiplier grows quadratically with increasing digit size. Therefore a 32×32 multiplier should be approximately 16 times the size of an 8×8 multiplier.

Concerning the RAM the estimates in table 6.7 include only space for as many digits as are necessary to perform the calculations. For HDL modeling and synthesis this number was rounded up to the next power of 2 (cf. table 6.8).

		Area $[GE]$ (without multiplier)			
Digit Size	**Number of Digits in RAM**	**RAM**	**Remaining Circuit**	**Total**	**Clock Cycles per Authentication**
32	151	40107	24742	64849	4723471
8	560	37233	16601	53834	66603587

Table 6.7: *High-level estimates for implementation D for two different digit sizes. The $d \times d$ multiplier is not taken into account in this estimate, therefore the difference between the two area estimates is not so large.*

Smaller digit sizes than 8 would be possible but seem unpractical, since the number of necessary digit-level operations (and hence the number of clock cycles to complete an authentication) is roughly proportional to the square of the number of digits per operand. Table 6.7 reveals that decreasing the digit size from 32 by a factor of 4 down to 8 bits per digit increases the cycle count by a factor of approximately 14.

Table 6.8 shows the synthesis results for implementation D. Although the area requirements are still considerable, most of the area is occupied by the RAM module. The synthesizer created a flip-flop-based RAM, which is not the most efficient implementation for a RAM. By using an efficient dedicated RAM hard macro a considerable amount of area could be saved.

Furthermore it is interesting to note that the critical path of this implementation is much shorter than those of the other implementations. This is due to the fact that this implementation does not contain a ripple-carry adder of full-precision size, but only of digit size. That means that this implementation could be clocked much faster; frequencies up to 290 MHz are possible with UMC 0.13μm CMOS technology. Since the number of

	Area $[\mu m^2]$			
Technology	**RAM**	**Remaining Circuit**	**Total**	**Critical Path** $[ns]$
	Word Size: 32 **bit; RAM for** 256 **digits**			
AMS 0.35 **[aus]**	3258364.25	361561.20	3619925.45	28.93
UMC 0.25 **[UMCb]**	1841495.08	181470.96	2022966.04	20.94
UMC 0.13 **[UMCa]**	431984.45	44115.45	476100.29	10.86
	Digit Size: 8 **bit; RAM for** 1024 **digits**			
AMS 0.35 **[aus]**	3449919.00	43880.47	3493799.47	10.16
UMC 0.25 **[UMCb]**	1872295.95	20599.92	1892895.88	5.98
UMC 0.13 **[UMCa]**	444569.47	4855.68	449425.15	3.41

Table 6.8: *Comparison of synthesis results of implementation D for different CMOS technologies, and two different digit sizes.*

nodes is significantly lower in the digit-level architecture compared to the full-precision architecture, the increased power consumption due to higher frequencies has less influence on this implementation than on full-precision implementations. To find the optimum balance in this trade-off one would need to know more about application-specific circumstances and constraints.

Chapter 7

Conclusion and Outlook

7.1 Comparison with Related Work

This work focused on presenting architectures for the coupon-recalculation approach (cf. §4.2.3) of the GPS authentication scheme. The basic idea was to meet the area- and power-constraints by disregarding the execution time of the commitment calculation. To the best of the author's knowledge this is the first time that this approach has been followed. Therefore it is difficult to present fair comparisons with other implementations. Nevertheless some other important results from the field of RFID security will be presented in the subsequent sections to provide a context for the results which were presented in §6.3.

7.1.1 Other GPS Implementations

McLoone and Robshaw presented hardware implementations of GPS in [MR07b] and [MR07a]. They focused on the response calculation step and did not consider performing on-tag commitment calculation. That means that they followed the complete-coupon approach (cf. §4.2.1). To increase performance they also utilized the low-Hamming-weight variant of GPS (cf. §4.5). The smallest implementation they achieved requires 431 gate equivalents, not taking into account NVRAM for storing coupons. The calculation of the response takes 136 clock cycles to complete.

Girault et al. presented another implementation of the GPS authentication scheme in [GJR07]. Their system is also based on coupons, but they use the partial-coupon approach (cf. 4.2.2) and implement a PRNG on the prover's side. The rationale behind that design decision is the fact that they also use the low-Hamming-weight variant. That means that the size of the random value r is quite large and that storing it in coupons would require significant space, which would limit the maximum number of coupons that could be realized. Therefore it is more reasonable to regenerate r on the fly, by means of an on-tag PRNG.

It is also interesting to note that the implementation presented in [GJR07] is based on the ECC variant of GPS (cf. §4.3). However, since a coupon approach is used, and the commitments are hashed before they are stored, this does not actually influence the hardware results; it is only interesting concerning the security level.

Furthermore it should be noted that Girault et al. created a working prototype of their implementation on a Xilinx Spartan 3E FPGA. The prototype requires approximately 6000 GE, of which 2600 are required for the cryptographic functionality (including memory to

store 10 coupons). The remaining 3400 GE are required for the supporting module, which performs the front-end communication. The time required for one complete authentication process is approximately 200 ms.

7.1.2 Other Authentication Schemes

An implementation which many papers use a reference for comparisons is the low-power AES implementation by Feldhofer et al. (cf. [FDW04] and [FWR05]). This implementation requires only approximately 3400 GE, and 1032 clock cycles for encrypting one 128 bit data block. The average power consumption is approximately 4.5 μW at 100 kHz.

In [FW07] Feldhofer and Wolkerstorfer analyze different cryptographic primitives and conclude that AES is most suited for passive devices like RFID tags. However the disadvantage of AES compared to GPS is the fact that it is a symmetric cipher, based on secret-key cryptography. As §2.3 has pointed out, such authentication methods have a limited field of application. Due to the problem of key management they are not very well-suited for open-loop systems. Thus the higher hardware requirements of public-key schemes like GPS are not in vain.

7.2 Future Work and Further Ideas

The exploration of the design space of the GPS authentication scheme is far from finished. Although this thesis tried to give a good and wide overview of interesting aspects of GPS, the practical implementations were focused more narrowly on investigating different architectures for the coupon-recalculation approach (cf. §4.2.3). Thus there are still many unanswered questions left to explore. The remainder of this section will summarize a few thoughts and ideas on some of these issues.

7.2.1 ECC Variant

§4.3 presented an ECC-based variant of the GPS authentication scheme. It is interesting to note that the digit-level architecture presented in §6.3.5 is almost capable of performing ECC operations. It is already capable of (modular) addition, multiplication, and squaring. With very little changes it could be enhanced to support additive inversion and thereby subtraction.

When counting the number of multiplications, additions, and subtractions necessary for scalar multiplication, one can extrapolate the performance of the architecture from the results presented in §6.3.5. By multiplying the number of clock cycles, which the architecture needs to perform a certain operation, with the number of times the respective operation is executed during scalar multiplication, one obtains a good approximation of how many clock cycles would be required for ECC operations.

Wolkerstorfer has recently done high-level-model-based estimations of the operations count of ECC (cf. [Wol07]). His findings could be the basis for extrapolating the performance of implementation D (cf. §6.3.5) to ECC. However, although implementation D is, with slight enhancements, capable of performing ECC operations, it has not been specifically designed for them. Therefore many optimizations might be possible by designing a new digit-level architecture, specifically tailored towards and custom-built for ECC.

Due to the fact that ECC operands are usually smaller (in terms of bit-size), full-precision architectures might also be of interest. Furthermore ECC operations can be

implemented by means of many different algorithms, which have different operation counts and memory requirements. An analysis of these algorithms, focused on low-power and low-size implementations, is currently conducted by Hein (cf. [Hei08]). His results will be very interesting in the scope of GPS.

Summing up it becomes clear that there are yet many questions left to answer, concerning ECC-based GPS-variants.

7.2.2 CRT Variant

The *Chinese remainder theorem* (CRT) provides an interesting optimization for commitment calculation if the factors of $n = p \cdot q$ are known to the prover. Instead of performing operations modulo n, it is possible to do them modulo p and modulo q. This is of less interest to full-precision architectures, because determining the final result still requires some operations modulo n. That means that the size of the arithmetic hardware cannot be reduced from the bit-size of n to the bit-size of p, q.

However, digit-level architectures do not care about the total size of the operands. The number of digits per operand only influences control flow, not the arithmetic architecture itself. Therefore a digit-level architecture could exploit the CRT to significantly reduce the number of digit-operations. The price for this reduction is the increased amount of NVRAM which is necessary to store the factors p, q. Whether or not this optimization pays off depends on application-specific circumstances.

7.2.3 Side-Channel Attacks

Side-channel attacks are attacks based on implementation specific behaviour of a cryptographic system. Characteristics like timing behaviour or power consumption are analysed to deduce information which was intended to be kept secret. Although side-channel attacks pose a serious threat, they have not been taken into consideration during the design of the architectures which have been presented in §6.3.

Nevertheless one interesting observation shall be noted here: The coupon-recalculation approach can easily be used to make side-channel attacks much more difficult. If coupons were chosen at random during authentication instead of the sequence in which they have been calculated, an attacker would not know to which commitment the side-channel information he or she observed would belong. That makes deducing secret information much harder.

7.2.4 Simple Compression Functions

As it has already been suggested in §4.4, simple compression functions could maybe replace the hash function in the GPS protocol. That way significant amounts of NVRAM could be saved, because coupons would have less bits; and if the compression function is sufficiently simple, its hardware requirements would only be modest.

However, according to [Gir07], so far there are no proofs that employing simple compression functions does not affect the security of the protocol. On the other hand there are also no proofs that simple compression functions necessarily impair the security. Certainly the question of security of such approaches is one which must be answered by cryptographers and mathematicians *before* they can be deployed to applications. Thus finding suitable compression functions, which allow for sufficiently simple hardware implementa-

tions, but do not impair security in the context of GPS, will be an interesting challenge for the cryptographic community in the future.

7.2.5 Efficient (Pseudo-)Random-Number Generators

The (pseudo-)random number generator (PRNG) is another challenge. According to [MvOV97] many up-to-date PRNG-implementations make use of ciphers. Therefore their hardware requirements would be significant, compared to the rest of the circuit. Finding more efficient PRNGs would certainly be beneficial not only for GPS implementations, but for many cryptographic protocols and applications.

In the scope of GPS, it should be noted that the output of the PRNG is never directly revealed. The random value r is only used for the calculation of the commitment $x = g^r \bmod n$ and the response $y = r + s \cdot c$. Therefore it might be possible that, just like the hash function, a PRNG used in GPS does not need to fulfil all usual requirements. Easing the requirements and prerequisites may very well allow for more efficient implementations.

Finding the minimum requirements which must be satisfied by a PRNG employed in GPS, and designing efficient implementations of such PRNGs is another interesting task which must yet be mastered.

7.2.6 Storing Checkpoints

§4.2.3 has already shortly mentioned another interesting issue concerning the coupon-recalculation approach. If tags are never powered long enough to complete calculating one coupon due to application-specific circumstances, then the coupon-recalculation approach cannot be employed. However, §6.3.1 explained that time had been disregarded for design decisions while developing different arithmetic architectures, to find more area- and power-efficient solutions. Therefore the time to complete the coupon-recalculation operation can become rather long, and the recalculation approach becomes applicable to a smaller set of possible applications; only those applications which guarantee that the tag is powered continuously to complete a recalculation operation are suited.

By storing checkpoints to NVRAM, from which the operation can resume at a later time, the set of possible applications can be extended to those which have shorter times of continues power supply for the tags. To give an example, if a tag calculated a modular exponentiation with a 260-bit random exponent by means of the square & multiply algorithm it would have to perform an average of $260 \cdot 1.5 = 390$ Montgomery multiplications, because in the average case 50% of the exponent's bits are 1-bits, which cause multiply operations. Now suppose that the tag would store the current values of all (auxiliary) variables of the square & multiply algorithm to NVRAM after each Montgomery multiplication. Then the time for which the tag must be powered continuously in order to make the coupon-recalculation approach work is decreased by a factor of (almost) 400.

However, if the tag was powered long enough to complete all 390 Montgomery multiplications then storing the intermediate results as checkpoints is a disadvantage, because it requires additional time. Furthermore NVRAM deteriorates with each write cycle; its life time decreases if unnecessary write cycles are performed. Thus finding the right balance between short times between checkpoints, overhead from writing to NVRAM, and avoiding storing unnecessary checkpoints, remains an interesting research task.

There must also be a way to check whether the data contained in a stored checkpoint is consistent and valid for resuming the operation or not, since it is possible that a tag looses power supply while it is storing a checkpoint. Therefore there must storage space

for at least two checkpoints. In case one checkpoint is in an inconsistent state the system must fall back to the previous one.

7.3 Summary and Conclusion

This work has presented an overview of the GPS authentication scheme and its many variants and options. Due to its high flexibility GPS can be employed in numerous application scenarios. On the other hand given so many choices it is difficult to thoroughly investigate the design space on a purely theoretical level, without knowing anything about application-specific constraints and circumstances. Therefore the implementations presented in this work focused on finding slow but area- and power-efficient ways to recalculate coupons. The results which were achieved might not yet be applicable to low-cost passive RFID devices, but the basic ideas and architectures which were presented seriously challenge the doctrine that public-key cryptography is completely out of the question on such devices from the outset.

The full-precision arithmetic unit which was presented requires approximately 50000 gate equivalents, plus approximately 2200 bits of NVRAM to store keys and domain parameters (if typical values for parameter sizes are used). It is capable of performing all arithmetic operations which are necessary for the execution of the GPS protocol: Modular exponentiation, integer multiplication, and integer addition. Calculating one commitment takes approximately 2.4 million clock cycles (including loading of parameters and storing results). The maximum clock frequency of the circuit is about 40 MHz, although it is suggested to use lower frequencies to decrease power consumption. §6.3.5 presented a digit-level arithmetic unit, which (using a digit size of 8 bits) requires only approximately 800 GE, plus RAM space for 560 digits. One commitment calculation takes about 66.6 million clock cycles, and the maximum clock frequency of the circuit, when synthesized for UMC 0.13μm CMOS technology, is approximately 290 MHz.

Furthermore this work has brought up several interesting questions, which are yet unanswered (cf. §7.2). If adequate answers to (some of) these questions can be found in the future, even more efficient implementations of GPS might be possible. In any case due to the broad field of possible applications it seems reasonable to further investigate and research low-resource implementations of public-key-based authentication schemes like GPS.

Appendix A

Definitions

A.1 Abbreviations

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
ATM	Automated Teller Machine
DLP	Discrete Logarithm Problem
DoS	Denial of Service
CMOS	Complementary Metal-Oxide-Semiconductor
CIOS	Coarsely Integrated Operand Scanning
CRT	Chinese Remainder Theorem
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
EEPROM	Electrically Erasable Programmable ROM
FPGA	Field Programmable Gate Array
GE	Gate Equivalent
IFP	Integer Factorization Problem
LSB	Least Significant Bit
LSD	Least Significant Digit
MOS	Metal-Oxide-Semiconductor
MOSFET	Metal-Oxide-Semiconductor Field Effect Transistor
MSB	Most Significant Bit
MSD	Most Significant Digit
$\boldsymbol{MonMul}$	Montgomery Multiplication
NMOS	n-type MOS Transistor
NVRAM	Non-Volatile Random Access Memory
$\boldsymbol{NRMM^s}$	Non Reduced Montgomery Multiplication of Order s
PDA	Personal Digital Assistant
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PMOS	p-type MOS Transistor
PRNG	Pseudo-Random Number Generator
RAM	Random Access Memory
RFID	Radio-Frequency Identification
ROM	Read-Only Memory
TTL	Transistor-Transistor Logic

A.2 Used Symbols

GND	Common ground identifier
V_{DD}	Supply voltage identifier
f_{CLK}	Clock frequency
$\mathbb{F}_p$ or $GF(p)$	Finite field with p elements, where p is a prime number
$\mathbb{Z}_n^*$	The set of invertible integers modulo n
$\otimes$	Abstract group operation
$+_n$	Addition modulo n
$\cdot_n$	Multiplication modulo n
$ld(x)$	Logarithm of x to the basis 2
$\gcd(a, b)$	Greatest common divisor of a, b

For symbols and variable names used in the scope of the GPS authentication scheme confer table 4.1 in §4.1.3 on page 43.

Bibliography

[aus] austriamicrosystems. 0.35μm CMOS Process Standard-Cell Library. `http://asic.austriamicrosystems.com/databooks/index_c35.html`.

[DK90] S.R. Dussé and B.S. Jr. Kaliski. A cryptographic library for the Motorola DSP56000. In I.B. Daamgard, editor, *Advances in Cryptology — Eurocrypt 90*, number 473 in Lecture Notes in Computer Science, pages 230 – 244, New York, 1990. Springer.

[FAFoA] Federal Army Forces of Austria. Basic military service. Instructions on Security Issues.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, August 2004. ISBN 978-3-540-22666-6.

[Fin02] Klaus Finkenzeller. *RFID-Handbuch — Grundlagen und praktische Anwendungen induktiver Funkanlagen, Transponder und kontaktloser Chipkarten*. Hanser, Munich; Vienna, 3rd edition, 2002. ISBN 3-446-22071-2.

[FR06] Martin Feldhofer and Christian Rechberger. A case against currently used hash functions in RFID protocols. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4277 of *Lecture Notes in Computer Science*, pages 372–381. Springer, October/November 2006. ISBN 978-3-540-48269-7.

[FT03] Clemens Fuchs and Robert F. Tichy. Mathematische Grundlagen der Kryptografie. Lecture Notes, Graz University of Technology, 2003.

[FW07] Martin Feldhofer and Johannes Wolkerstorfer. Strong crypto for RFID tags — a comparison of low-power hardware implementations. In *IEEE International Symposium on Circuits and Systems, ISCAS 2007*, pages 1839–1842, May 2007.

[FWR05] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES implementation on a grain of sand. *IEEE Proceedings on Information Security*, 152(1):13–20, October 2005.

[Gir91] Marc Girault. An identiy-based identification scheme based on discrete logarithms modulo a composite number. In I.B. Daamgard, editor, *Advances in Cryptology – Eurocrypt '90*, number 473 in Lecture Notes in Computer Science, pages 481 – 486. Springer, 1991.

[Gir92] Marc Girault. Self-certified public keys. In D. Davies, editor, *Advances in Cryptology — Eurocrypt '91*, number 547 in Lecture Notes in Computer Science, pages 490 – 497. Springer, April 1992.

[Gir00] Marc Girault. Low-size coupons for low-cost ic cards. In *Smart Card Research and Advanced Applications, Proceedings of the Fourth Working Conference on Smart Card Research and Advanced Applications, CARDIS 2000, September 20-22, 2000, Bristol, UK*, volume 180 of *IFIP Conference Proceedings*, pages 39–50. Kluwer, 2000. ISBN 0-7923-7953-5.

[Gir07] Marc Girault. Email correspondence, summer/autumn 2007.

[GJR07] Marc Girault, L. Juniot, and M.J.B. Robshaw. The feasibility of on-the-tag public key cryptography. In *Proceedings of the International Conference on RFID Security 2007*, 2007. To be published.

[GL04] Marc Girault and David Lefranc. Public key authentication with one (online) single addition. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2004. ISBN 978-3-540-22666-6.

[GPS06] Marc Girault, Guillaume Poupard, and Jacques Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19(4):463–487, October 2006.

[GS94] Marc Girault and Jacques Stern. On the length of cryptographic hash-values used in identification schemes. In *Advances in Cryptology - CRYPTO '94: 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 1994. Proceedings*, volume 839 of *Lecture Notes in Computer Science*, page 202. Springer, 1994.

[Gue03] Shay Gueron. Enhanced montgomery multiplication. In *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13-15, 20. Revised Papers*, volume 2523/2003 of *Lecture Notes in Computer Science*, pages 87 – 100. Springer, 2003.

[Hei08] Daniel Hein. Elliptic-Curve Cryptography suitable for RFID systems. Master's thesis, Graz University of Technology & ETH Zurich, to be published 2008.

[HMV03] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, New York, 2003. ISBN 0-387-95273-X.

[IAI] IAIK. Designflow. Internal Designflow Framework.

[IEE01] IEEE. IEEE Standard Verilog Hardware Description Language, 2001.

[ISO04] ISO/IEC. International Standard ISO/IEC 9798 Part 5: Mechanisms using zero-knowledge techniques, December 2004.

[KAK96] C. K. Koç, T. Acar, and B.S. Jr. Kaliski. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. In *Mathematics of Computation*, volume 44, pages 519 – 521, 1985.

[MR07a] M. McLoone and M. J. B. Robshaw. New architectures for low-cost public key cryptography on RFID tags. In *IEEE International Symposium on Circuits and Systems, ISCAS 2007*, pages 1827–1830, May 2007.

[MR07b] M McLoone and M.J.B. Robshaw. Public key cryptography and RFID tags. In *Topics in Cryptology — CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*. Springer, 2007. ISBN 978-3-540-69327-7.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, New York, 1997. ISBN 0-8493-8523.

[NES04] NESSIE. Final report of European project number IST-1999-12324, named new european schemes for signatures, integrity, and encryption. `https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf`, April 2004.

[NIS00] NIST FIPS 186-2. Digital signature standard. National Institute of Standards and Technology, 2000.

[NIS02] NIST FIPS 197. Advanced Encryption Standard, 2002.

[Osw06] Elisabeth Oswald. IT security. Lecture Notes, Graz University of Technology, May 2006.

[Par00] Behrooz Parhami. *Computer Arithmetic — Algorithms and Hardware Designs*. Oxford University Press, New York, 2000. ISBN 0-19-512583-5.

[PS98] Guillaume Poupard and Jacques Stern. Security analysis of a practical "on the fly" authentication and signature generation. In *Advances in Cryptology — EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, page 422. Springer, 1998. ISBN 978-3-540-64518-4.

[QGB90] Jean-Jacques Quisquater, Louis C. Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology – CRYPTO '89: Proceedings*, pages 628–631, 1990.

[Rab96] Jan M. Rabaey. *Digital Integrated Circuits — A Design Perspective*. Prentice Hall Electronics and VLSI Series. Prentice Hall, New Jersey, 1996. ISBN 0-13-178609-1.

[Raz02] Behzad Razavi. *Design of Analog CMOS Integrated Circuits*. Tata McGraw-Hill Publishing Company Limited, 2002. ISBN 0-07-052903-5.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sta00] Standards for Efficient Cryptography Group. SEC 2: Recommended elliptic curve parameters. `http://www.secg.org/download/aid-386/sec2_final.pdf`, September 2000.

[TS02] Ulrich Tietze and Christoph Schenk. *Halbleiter-Schaltungstechnik*. Springer, 12th edition, 2002. ISBN 3-540-42849-6.

[UMCa] UMC. UMC standard cell library — 130 nm CMOS process.

[UMCb] UMC. UMC standard cell library — 250 nm CMOS process.

[Wer02] Anette Werner. *Elliptische Kurven in der Kryptographie*. Springer, Berlin – Heidelberg, 2002. ISBN 3-540-42518-7.

[Wik07a] Wikipedia. Alice and Bob. `http://en.wikipedia.org/wiki/Alice_and_Bob`, November 2007.

[Wik07b] Wikipedia. Zero-knowledge proof. `http://en.wikipedia.org/wiki/Zero-knowledge_proof`, November 2007.

[Wol04] Johannes Wolkerstorfer. *Hardware Aspects of Elliptic Curve Cryptography*. PhD thesis, Graz University of Technology, Institute for Applied Information Processing and Communications (IAIK), 2004.

[Wol07] Johannes Wolkerstorfer. ECCU3. Internal Research, 2007.

Index