KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering Science
Department of Electrical Engineering

Study and Design of a Reusable Embedded Hardware Architecture for Secure Wireless Communication

Geoffrey Ottoy

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

November 2013

Study and Design of a Reusable Embedded Hardware Architecture for Secure Wireless Communication

Geoffrey Ottoy

Jury:
Prof. Dr. Ir. Jean Berlamont, president
Prof. Dr. Ir. Bart Preneel, promotor
Prof. Dr. Ir. Lieven De Strycker, promotor
Dr. Ir. Nele Mentens
Dr. Vincent Naessens
Prof. Dr. Ir. Bart Nauwelaers
Prof. Dr. Sc. François Corthay
(HES-SO Vallais - Switzerland)
Dr. Ing. Thomas Groß
(University of Newcastle upon Tyne - UK)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

November 2013

D/2013/7515/122
ISBN 978-94-6018-737-7

Preface

Sitting on the ICE 954 on my way back from a conference in Magdeburg, writing this preface seems like a fitting end of the journey this PhD has been. Not only is it the end of four days in Germany, where I presented the results of the case study on attribute-based credential verification –you can find these results in Chapter 9 of this thesis, but bear with me for a few more minutes,– it is also the end of four years of work, of which the text you are holding in your hand is one of the results.

The journey, however, started already in 2006 when, fresh out of school and proud of my newly attained engineering degree, I was allowed to start work at the DraMCo research group on an IWT project on indoor localization with wireless sensor networks. It is only fitting that I thank Luc De Backer, Lieven De Strycker and Jean-Pierre Goemaere for their faith in me.

I had already expressed my interest in doing a PhD and when I was asked to start research on a still vague topic involving wireless security with resource-limited nodes, I could only but accept. However, I have to admit that it was with a heavy heart, because "security" was a mystery to me. The first two years, consequently, I felt like a conquistador, carving my way through an academic jungle of papers, not knowing where my journey would end or what I would discover. That I kept going, is largely thanks to of my family, friends and (ex-)colleagues. Thank you mom, dad, Emile, Gerwinde, Maarten, Anneleen, Kevin, Jorge, Bert, Jeroen, Tom, Henk, Steven, Bogdan, Davy, and Dries. Not only have you been there when "the going was tough", but I could share my successes with you as well.

For the final part of this PhD I have had the luck that I could supervise the theses of two master students with initiative and the will to work. They provided me with practical support and allowed me to focus on what really mattered for my thesis. Thank you Sam and Jonas. Also for one of the case studies I was fortunate to be able to work together with colleagues from the MSEC research group at KAHO Sint-Lieven. Their expertise proved invaluable and our combined work resulted in a nice paper. –Jorn and Vincent, next time you should accompany me. Let's say to... Hawaii?

The quality of this text has been safeguarded by the members of the Jury. Thanks to their remarks, I was able to patch the missing links, plug the holes and improve the coherency. My special thanks go to my promoters, Prof. Bart Preneel and Prof. Lieven De Strycker, who have guided and directed me whenever that was necessary during the past four years. I would also like to express my gratitude to Dr. Thomas Groß and Prof. François Corthay for agreeing to be members of the jury, as well as to my assessors Prof. Bart Nauwelaers, Dr. Nele Mentens and Dr. Vincent Naessens. I would explicitly like to thank Prof. Jean Berlamont for presiding the jury.

Finally, this text would not have been what it is today, without my beloved Charlotte. Not only her English skills, but also the simple fact that she was with me on this journey, has made that you, dear reader, are holding this text in your hands. I hope you enjoy reading it.

Geoffrey Ottoy
September 27, 2013
Somewhere in Germany

Abstract

The proliferation of wireless embedded devices and the boom of related applications have set design engineers the difficult task of supporting security for these emerging applications. This encompasses hiding a user's sensitive data, safeguarding his privacy and authenticating communicating parties as well as the data that is being exchanged. Implementing these security measures in an embedded context requires a multidisciplinary approach and often forces designers to make a trade-off between, processing speed, memory usage, energy, cost, etc., which are not only influenced by the security measures itself, but also by the application and the communication.

In this PhD, we have developed an embedded test platform that allows design engineers to quickly implement proof-of-concept applications, evaluate them, and make educated design choices on how to implement the required security measures. As an addition to this platform we have designed a hardware accelerator for offloading the modular exponentiations required for several public-key security protocols. To keep the range of possible applications and hardware platforms as broad as possible, we have made this design highly customizable. A second topic is the study of Near-Field Communication (NFC) as a medium to communicate between a mobile device (e.g., a tablet or smartphone) and an embedded terminal (e.g., a vending machine, access control point, or ticketing terminal). We also extend the functionality of our embedded test platform by adding support for NFC.

Finally, we have used our embedded platform in two case studies to validate our design and to evaluate different design approaches in a practical setup. A first case study focuses on attribute-based credential verification (a privacy-preserving technique) over NFC and evaluates the influence of communication and processing (both in hardware and software) on the application run time. A second case study evaluates the data rates and communication times of NFC compared to an approach in which NFC is used to initiate communication over a faster WiFi channel.

Samenvatting

De opkomst van draadloze ingebedde toestellen en de toename van bijhorende toepassingen, plaatst ontwerpingenieurs voor de moeilijke taak om de veiligheid in deze nieuwe toepassingen te ondersteunen. Dit omvat het verbergen gevoelige gegevens van een gebruiker, de vrijwaring van zijn privacy en het authenticeren van communicerende partijen, evenals de gegevens die worden uitgewisseld. Het toepassen van deze beveiligingsmaatregelen in een ingebedde omgeving vereist een multidisciplinaire aanpak en dwingt ontwerpers vaak tot het maken van een afweging tussen, snelheid, geheugengebruik, energie, kosten, enz., welke niet alleen worden beïnvloed door de veiligheidsmaatregelen zelf, maar ook door de applicatie en de communicatie.

In dit doctoraat hebben we een ingebed testplatform ontworpen, dat ontwikkelaars toe laat om snel tot een proof-of-concept applicatie te komen, deze te evalueren, en op basis daarvan gefundeerde keuzes te maken over hoe men het beste de benodigde beveiligingsmaatregelen kan implementeren. Als aanvulling op dit platform hebben we een hardware-accelerator ontworpen voor het versnellen van de modulaire exponentiaties die nodig zijn voor verschillende publieke-sleutel beveiligingsprotocollen. Om het aantal mogelijke toepassingen en hardware platformen zo groot mogelijk te houden, hebben we dit ontwerp in hoge mate aanpasbaar gemaakt. Een tweede onderdeel van dit werk, is de studie van Near-Field Communication (NFC) als communicatiemedium tussen een mobiel apparaat (bijvoorbeeld een tablet of smartphone) en een ingebedde terminal (zoals een drankenautomaat, elektronisch slot, of ticketing terminal). We hebben ook de functionaliteit van ons ingebedde testplatform uitgebreid door het toevoegen van ondersteuning voor NFC.

Ten slotte hebben wij in twee case studies gebruik gemaakt van ons platform om ons ontwerp te valideren en om verschillende ontwerpmogelijkheden te evalueren in een praktische opstelling. Een eerste case studie richt zich op verificatie van *credentials* gebaseerd op attributen (een privacy-beschermende techniek) over NFC en evalueert de invloed van communicatie en berekeningen (zowel in hardware en software) op de uitvoeringstijd van de toepassing. Een tweede case studie evalueert de datasnelheden van NFC in vergelijking met een aanpak waarbij NFC gebruikt wordt om communicatie via een sneller WiFi kanaal te starten.

Abbreviations

ABC	Attribute-Based Credentials
ABM	Asynchronous Balanced Mode
AES	Advanced Encryption Standard
AMBA	Advanced Microcontroller Bus Architecture
ANSI	American National Standards Institute
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASK	Amplitude Shift Keying
AXI	Advanced eXtensible Interface
BIOS	Basic Input/Output System
BMM	Bipartite Modular Multiplication
BPI	Byte Peripheral Interface
CLB	Configurable Logic Block
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CTR (mode)	Counter Mode
DDR3	Double Data Rate (type 3 SDRAM)
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DIP	Dual In-line Package

DMA	Direct Memory Access
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
DSP	Digital Signal Processor
DTS	Device Tree Source
ECC	Elliptic Curve Cryptography
FF	Flip Flop (1-bit register)
FIFO	First In First Out (buffer)
FIPS	Federal Information Processing Standard
FMC	Field Programmable Mezzanine Card
FPGA	Field-Programmable Gate Array
GDB	The GNU project debugger
GE	Gate Equivalent
GIPS	Giga Instructions Per Second
GNU	"GNU's Not Unix" (recursive acronym)
GPIO	General Purpose I/O
HECC	Hyper Elliptic Curve Cryptography
HSM	Hardware Security Module
HSU	High-Speed UART
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IRQ	Interrupt Request
JNI	Java Native Interface
JTAG	Joint Test Action Group; designates the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture
JVM	Java Virtual Machine

LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LGPL	GNU Lesser General Public License
LLCP	Logical Link Control Protocol
LMB	Local Memory Bus
LUT	Look-Up Table
MMU	Memory Management Unit
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFS	Network File System
NIC	Network Interface Card
NTP	Network Time Protocol
OS	Operating System
P2P	Peer-to-Peer
PCB	Printed Circuit Board
PCIe	PCI Express
PCI	Peripheral Component Interconnect
PDU	Protocol Data Unit
PE	Processing Element
PHY	Physical layer
PIN	Personal Identification Number
PKC	Public-Key Cryptography/Cryptosystem
PLB	Processor Local Bus
RAM	Random Access Memory
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computer

RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
RSA	Rivest Shamir Adleman (public-key algorithm)
SAP	Service Access Point
SDRAM	Synchronous Dynamic RAM
SE	Secure Element
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SNEP	Simple NDEF Exchange Protocol
SO-DIMM	Small Outline Dual In-line Memory Module
SoC	System on Chip
SPA	Simple Power Analysis
SPI	Serial Peripheral Interface
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
UART	Universal Asynchronous Receiver/Transmitter
UIO	Userpace I/O (Input/Output)
URL	Uniform Resource Locator
USB	Universal Serial Bus
VHDL	Very High-Speed Integrated Circuit Hardware Development Language
WLAN	Wireless LAN
ZKPK	Zero-Knowledge Proof-of-Knowledge

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Setting

1.1.1 The Mobile Revolution

Whether they are sceptical or enthusiastic about its effect on our everyday lives and our future, most people will agree that today's technology is getting smarter. Moreover, it is seeping into all aspects of our society. Boosted by Moore's Law, electronic devices are getting smaller, more energy-efficient and more powerful. To illustrate this, let us take a look at the Intel Pentium 4 (Extreme Edition), a single core processor often found in home computers by the turn of the millennium. At 3.2 GHz it obtains 9 to 10 GIPS, dissipating more than 100 W. If we compare this with the NVIDIA Tegra 3, released in 2011 and found in several of the latest smartphones, the difference is clear. This quad core can obtain between 13 and 14 GIPS, running at 1.5 GHz but only dissipating a few Watts.

So we are carrying around electronic devices that are already more powerful than our personal computers of a decade ago. Moreover, chips such as the Tegra not only house a general purpose processor, but also combine a graphics processing unit, memory controller, and bridges for peripherals in the same package. This makes them highly suitable for digital signal processing, such as playing media or recognizing a song of which you cannot remember the title.

Combined with several wireless interfaces such as Bluetooth, GPRS, WiFi, GPS or NFC, and packed with sensors such as a touch screen, accelerometers and gyroscopes, smartphones are becoming an extension to the human body. They have become like an extra sense and act as a physical interface for ubiquitous computing applications [BBRS06]. Where a mobile phone was previously used for calling and

sending text messages, it is now much more: a book, a camera, a map, a key, a wallet,...

The role of the Internet has also changed in the past few years. The days where people chatted on MSN and searched for information using Altavista are long gone. Now we can "Google" everything, share our pictures (and "like" them) on Facebook. We check the news, manage our finances and even file our taxes online. The upswing of smartphones has lifted this (r)evolution to a whole new level. It is yet another medium for people to stay in touch with friends, and manage their social lives [Hay05]. Companies have also discovered this new market, as evidenced by the many online banking apps or support for storage and validation of e-tickets on your mobile device. Even schools and healthcare have expressed their interest in the possibilities offered by mobile computing.

1.1.2 Designing for Embedded and Mobile Security

As mobile devices are increasingly used for financial transactions or to manipulate personal data, there is an urgent need for reliable security. In cases where embedded stand-alone devices, often with less computational power than a mobile phone, are controlling the access to a service, the implementation needs to be well-thought out as well. Think, for instance, of vending machines where you can pay with you phone or lockers and storage containers that are accessible by using your mobile device.

Another upcoming technology is the so-called *Internet-of-Things* (IoT). Sensors and actuators are interconnected locally or even worldwide in order to cooperatively offer an enhanced service. An example here is a healthcare or home monitoring application for seniors.[1] This can include fall detection based on cameras. These images are processed locally, but in case of a fall detection they might be accessed remotely. It is clear, however, that they should not be accessible to "spy" on the residents of the home.

Also in smart metering applications (the *smart grid*) electronic meters transmit their measurements to the gas, water or electric companies. When insufficiently secured, the data communication can also be used for more malicious purposes. In these cases, where data processing and the accompanying communication[2] happens on and between autonomous devices, it is more a matter of not revealing the inherently obtained personal information to third parties. This requires encryption, authentication and a well-conceived key management.

[1] This is a use case for the IoT, given by Intel: `http://www.embeddedintel.com/special_features.php?article=2721`.

[2] The term *Machine-to-machine* (M2M) communication is a much-used buzz word.

When it comes to making the design choices on how to implement the security in embedded devices, engineers have to make several trade-offs in terms of standardization, energy, memory, complexity, form factor, cost, etc. to meet the proposed requirements. This is a daunting task that requires knowledge of e.g., encryption algorithms, entity authentication protocols, communication standards, hardware design, and energy management.

Another important part of embedded security design is physical or electronic security. When embedded devices store secret information, for instance, a symmetric key or intellectual property, they can be physically attacked. For example, if the JTAG interface is not disabled, it can be used to read the FLASH contents of a microcontroller, possibly revealing a key or proprietary software. A nice overview of JTAG attacks and how to protect against them, is given by Rosenfeld and Karri [RK13]. One way to deal with this kind of attacks is to store information essential for the security in a tamper-proof element e.g., a SIM card or Trusted Platform Module (see also Chapter 3). These keys and hash values can then be used to bootstrap security at board power-on. This bootstrapping can include verification of signed executables, decryption of data stored in unprotected memory, etc.

During the operation of the device, secret information can be leaked to the outside world over so-called *side channels* [Koc96, KJJ08]. These include current drawn by the circuitry, temperature variations, the electromagnetic field generated by the device, etc. Preventing this kind of attacks is a difficult task often requiring low-level hardware and software design. For instance, a metal casing can serve as electromagnetic shielding. However, holes for I/O and power supply lines are required through which information can still leak to the outside world. Furthermore, removal of the shield should be detected and render the device inoperable.

Typically, embedded system designers do not build a complete system from scratch. They use *black boxes* (or IP cores) with known specifications and merely provide the glue logic and software to make them work together in offering the required functionality. Security is often perceived as a nuisance and is overlooked in the primary design phase: it is seen as something that will be added later because it is not viewed as an essential part of the basic functionality. This often leads to dubious implementations or results in performance drop, because the effects of the security measures have been misjudged. Even more reprehensible is that algorithms and protocols that have proven their worth are discarded in favor of the *security by obscurity* i.e., some proprietary protocol.

1.1.3 Your Most Dangerous Possession? Your Smartphone[3]

Bad guys are following where the people are going – and people are going to smartphones[4]

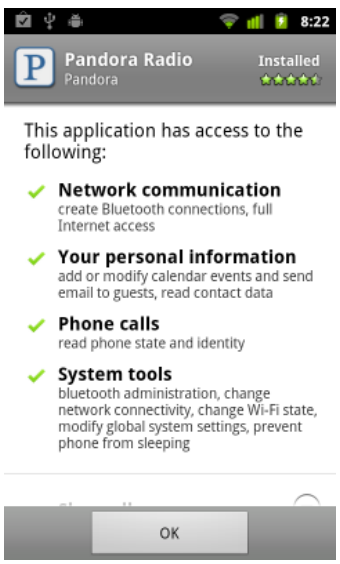


Figure 1.1: Granting permissions to the Pandora Radio app.

As always, there is also a downside to the evolution in mobile computing. As mobile devices are increasingly being used to store personal information, for mobile banking or to participate in social network sites, they have become an attractive target for cyber criminals. Spoofing attacks (man-in-the-middle) with a malicious website sitting in between the victim and his bank's website to retrieve his personal information have now come to the mobile domain. Typically they are initiated by a fake text message with the link to the malicious website. Unsuspecting victims often believe this to be a genuine message from their bank. The same method is used to install *trojans* or illegal data collection services.

Companies will also be able to exploit your personal data if you allow them to (see Fig. 1.1). For instance, the *Pandora Web Radio* or *CBS News* apps for Android include third-party advertising libraries. A study by *Veracode* has shown that these libraries transmit personal information such as your GPS location or contact addresses to advertising agencies in mass quantities.[5] With mobile broadband subscriptions still rising (Fig. 1.2), the problem is likely to only get bigger.

Several projects exist that enhance a user's privacy or help him stay anonymous; one of the most notable being TOR [DMS04].[6] The main problem with most of these projects is that they are either small scale, not easily accessible, even obscure and associated with criminal activities. On the other hand, why would the leaking of personal information be a problem? Only to get a chance of winning, for instance, some cinema tickets, people fill in a web form and "give away" their

[3]Taken from CNN Money: `http://money.cnn.com/2011/01/11/pf/smartphone_dangers/index.htm`

[4]George Peabody, director of emerging technologies at Mercator Advisory Group

[5]Mobile Apps Invading Your Privacy: `http://www.veracode.com/blog/2011/04/mobile-apps-invading-your-privacy/`
Mobile App Privacy Continued... : `http://www.veracode.com/blog/2011/04/mobile-app-privacy-continued/`

[6]https://www.torproject.org/index.html.en

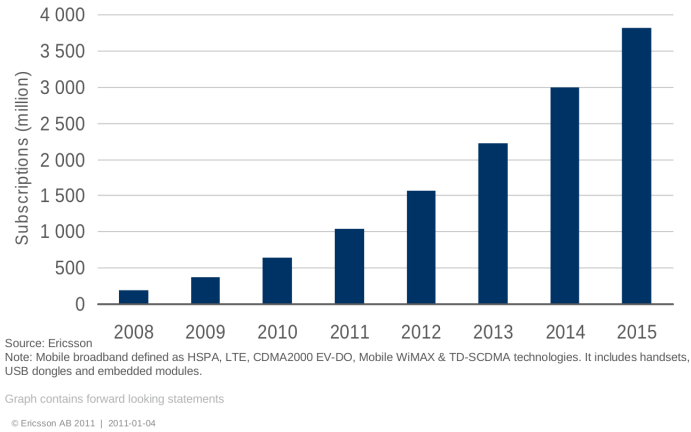


Figure 1.2: Evolution (prediction) of the number of mobile broadband subscribers (courtesy by Ericsson).

personal info. Or what about Facebook?[7]

1.1.4 Privacy Preserving Techniques

The need for privacy is a socially created need. Without society there would be no need for privacy.[8]

But what is privacy? In his 2002 article [Sol02], Solove states that previous attempts to conceptualize privacy can be summarized in six overlapping approaches: the right to be let alone,[9] limited access to the self, secrecy, control of personal information, personhood, and intimacy. He argues, however, that this generalization leads to imperfection and that privacy should be conceptualized in a specific context. This is especially important when ensuring up-to-date and to-the-point legislation for the *Digital Information Age*.

As Solove states, the term "privacy" is an umbrella term. It refers to a broad and heterogeneous group of related things [Sol06]. Rather than focusing on a definition of privacy itself, he argues that, with regard to lawmaking, effort should be put into addressing the activities that *can* create a problem. His taxonomy divides these activities into four types (each with their own subgroups) acting upon a subject and most of them related to the personal information of the subject: information collection, information processing, information dissemination and invasions. The

[7] ... to kick in an open door.

[8] Barrington Moore, Jr., Privacy: Studies in Social and Cultural History 73 (1984).

[9] Possibly the first attempt to "boil down" privacy into a single phrase by Warren and Brandeis (1890) [WB90]

relationships between theses activities and their corresponding subgroups are shown in Fig. 1.3.

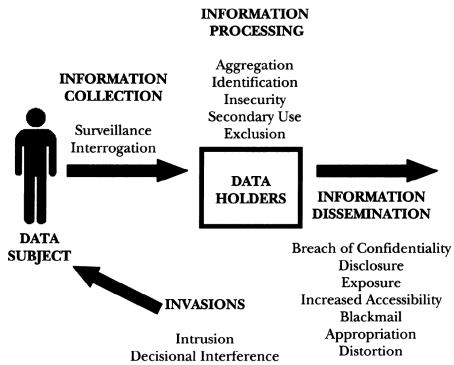


Figure 1.3: Activities that harm the privacy of a subject (courtesy by D. Solove [Sol06]).

The activities do not have to be harmful per se. When a subject agrees to them, there is no privacy violation. On the other hand, people are willing to consent to some activities for small rewards [Acq04]. Also, the fact that an activity is harming the subject's privacy does not mean the activity is not justified, i.e., there are occasions where the subject should be held accountable for his actions [All03]. It always comes down to maintaining a (precarious) balance between personal liberties and government control. With the rise of the digital information flow (i.e., your personal (digital) dossier flows via the plumbing of data banks and computer networks from one organization or company to another) [Sol04] and the fact that "the web never forgets" [Ros11], the complexity of the problem only seems to grow.

Privacy, only for those with something to hide?

By suggesting that people only care about their privacy when they have something to hide, any discussion regarding this subject is avoided. For instance, in the US, a country proud of its civil liberties, those liberties are trampled in the name of national security. However, the recent disclosures on the NSA Prism program tapping into our Google, Apple, and Microsoft user data, have stirred the people's concern [Mac13].

So why should people have to be aware of the risks associated with the release of personal information? First of all, social network sites are a great resource for social engineering attacks on mailboxes and computers (in case of a trivial password). Also, burglars can scour for potential targets, for instance, people who are on vacation and who have disclosed this information on their public profile. Your surfing behaviour, interests and online purchases are monitored and used to create an extensive (but possibly incorrect) profile that is used to send you personalised advertising.

But there is no telling what the next step will be. Will your profile be sold, distributed to insurance companies or financial institutions, that will use it to decide whether to grant you another insurance or loan? Your smartphone can be used as a hub for, a personal area network (PAN), or body area network (BAN),

monitoring your heart rate, brain activity and emotional state.[10] It is clear that the dangers may be great, that *digital security awareness* must be taught or even that service providers should even protect their users against their own ignorance.

On the other hand, why would service providers care to do this? It might require a more complex infrastructure, even a complete redesign. The answer is that they will only do this if some kind of benefit can be gained from it. So what benefits could there be in offering privacy-friendly services? Spalding argues that there are several reasons why service providers should take steps towards protecting their users, or as he states, to reduce the provider's liability [Spa13]. Some of these reasons are of legal nature, e.g., to prevent potential lawsuits or to prevent users and third parties from using the services in unintended ways. Other reasons are more economically inspired: happy users keep returning.

To help companies in offering privacy-friendly services, several cryptographic protocols have been proposed in the literature. These so-called *attribute-based credentials* can be compared with classic certificate technology in the sense that they provide a user with a certified set of attributes (i.e., the credential). However, instead of showing the entire certificate and thus revealing all the associated attributes, every time a service is requested, the user can choose to reveal only the attributes required for accessing the service. Moreover, the next time this user makes a service request, it cannot be linked to his previous interactions with the provider. To protect the companies' interests, credentials can be revoked and when necessary a user's identity can be traced back.

IBM with the `Identity Mixer` [17] en Microsoft with `U-Prove` [35] have both implemented a cryptographic library to support the use of attribute-based credentials. Each implementation is based on different concepts. `U-Prove` is not as computationally intensive as the `Identity Mixer`, but requires more memory when unlinkablity of transactions is required.

1.2 Contributions

As we have seen, the mobile revolution will require design engineers to come up with fitting measures to counter the security issues that come with the whole range of new applications that emerge. The design of embedded devices needs to incorporate the system design, the communication and the application, as well as features like energy consumption, form factor, and cost. In this work, we will focus on the former three. This is presented graphically in Fig. 1.4.

[10] IMEC Body area network applications: `http://www.imec.be/ScientificReport/SR2011/1414067.html`

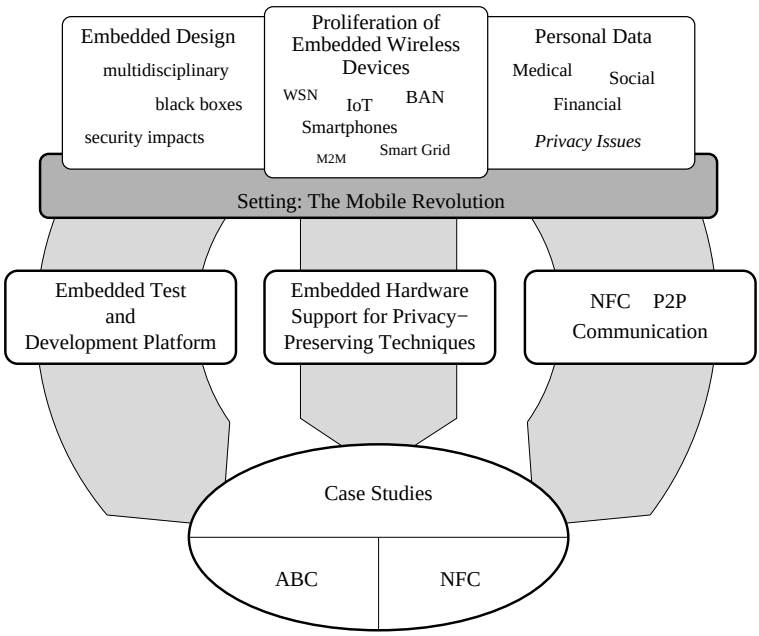


Figure 1.4: Structure of the thesis.

Starting from the setting of the mobile revolution, we evolve along three separate tracks. The first track is the design of a configurable and expandable embedded platform to aid engineers in prototyping and design evaluation. A second track is the development of hardware support for cryptographic computations on the embedded test platform. The main focus is on supporting privacy preserving techniques, while just like the platform itself, the hardware must be configurable to maximize the range of applications that can be evaluated. A third track will focus on the communication between a mobile device and this embedded test platform. We will consider Near-Field Communication (NFC).

These three tracks lead to two practical cases where the developed test platform, with its support for communication and computation, is used to evaluate security protocols. This approach serves to reach our **overall goal:** *building an embedded test platform that developers can use to implement security protocols as well as evaluate all their aspects e.g., communication, computation, system requirements.* With the two case studies we validate our design and demonstrate its worth.

The main contributions of this work hence follow from this approach:

Our first main contribution is the design of an embedded platform for testing and evaluating the impact of hardware offloading of cryptographic operations

as well as evaluating the performance of certain communication protocols on an embedded terminal. With this platform a design team can easily experiment with different hardware IP cores, prototype different blocks separately, join them together, prematurely detect design flaws and quickly come to a proof-of-concept that can serve as the basis for the final product. The embedded Linux on the central processor enables the use of standard libraries for arithmetic or communication, hence reducing the design effort. We have also provided libraries for NFC communication and examples on how to write custom hardware drivers and on how to use the NFC libraries. Currently designers are forced to work with development kits that are only partially configurable, hence forcing them to focus on only one aspect of an embedded application. This platform provides the (missing) synergy between commonly available wireless development platforms on the one hand, and embedded hardware development boards on the other.

As a second main contribution we have designed an IP core to offload (to hardware) the multiple modular exponentiations required for attribute-based credentials and specifically targeted for our embedded test platform. This means that the main objective of this IP core design is customizability to the designers' needs, while maintaining an acceptable performance in comparison to specifically tailored designs. We have validated the core's performance in terms of resource usage and run time. In addition, we provide several expressions that allow a developer to estimate beforehand the run time and required resources for a certain configuration of our IP core. As the core is able to carry out two exponentiations simultaneously, we have also validated how this increases performance in comparison to a single exponentiation. In that regard, we also provide expressions to determine the run time and performance speedup for simultaneous exponentiations.

Our third main contribution focuses on NFC, a relatively new standard for short-range wireless communication. We have investigated the requirements for the mobile device as well as the embedded terminal in order for them to communicate with one another. We have also implemented several solutions in practice.

A fourth and final contribution is the use of our developed test platform in two case studies. All the previous work culminates in these two cases, thus validating our overall goal (p. 8). A first case study focuses specifically on the verification of attribute-based credentials on an embedded terminal. We have examined the total run time of the `Identity Mixer` credential proof protocol and investigated how the length of the parameters, the number of attributes and the number of revealed attributes all influence the time required for both communication (over NFC) and computation. For the latter, we have compared an implementation with both a single-base exponentiation and a dual-base simultaneous exponentiation offloaded to hardware. A second case study focuses specifically on the NFC (peer-to-peer) communication between an Android smartphone and our embedded test platform. We have evaluated the maximum attainable data rate and compared this to a scenario with connection handover from NFC to WiFi.

1.3 Outline

The remainder of this text is structured according to the contributions presented above.

Chapter 2: A Platform for Developing, Testing and Evaluating Embedded Security presents the embedded test platform we have developed. We clarify our design choices and discuss the most important features.

Chapter 3: Introduction to Hardware Support for Modular Multi-Base Exponentiations is an introductory chapter with regard to the applications and requirements of such hardware support. We provide the evolution and latest work for modular exponentiation hardware. This gives us an insight into what features are available and what is the state of the art for resource usage and timing. We look at some commercial products as well.

Chapter 4: Design of an Open-Source IP Core for Modular Simultaneous Exponentiation details the goals, underlying ideas and design choices for our hardware accelerator. We provide insight in the operation and the implemented features.

Chapter 5: Multiplier Performance demonstrates how the multiplier, which is the kernel of our IP core, performs in terms of execution speed and resource usage.

Chapter 6: IP Core Performance demonstrates the complete IP core as part of our embedded test platform. We provide expressions for exponentiation run time and speedup[11] and verify these expression in practice.

Chapter 7: NFC Peer-to-Peer Communication highlights the important features of the current specification for NFC and NFC peer-to-peer communication. We specify the requirements for setting up an NFC peer-to-peer connection between an Android smartphone and an embedded terminal and give examples as to how to do this for both the smartphone and the terminal.

Chapter 8: Attribute-Based Credentials provides an insight into the topic of attribute-based credential systems. The first goal is merely to illustrate the difficulties with embedded implementations of these protocols. A second goal to show how credential verification is specified in the `Identity Mixer`, which is the subject of one of the case studies.

[11] i.e., the decrease in run time when using simultaneous exponentiations instead of single-base exponentiations.

Chapter 9: Attribute-Based Credential Authentication with Embedded Devices demonstrates a practical setup for attribute-based credential verification on an embedded platform. We look at communication as well as computation run times and investigate the effect of using a hardware accelerator.

Chapter 10: NFC P2P versus NFC handover to WiFi investigates the NFC communication between an Android smartphone and our embedded test platform. We compare the communication speed attainable over NFC with an approach that uses communication handover to WiFi.

Chapter 11: General Conclusions restates our most important realizations and findings.

Chapter 2

A Platform for Developing, Testing and Evaluating Embedded Security

2.1 Introduction

When it comes to designing and more specifically making design choices on how to implement embedded security, engineers have to make several trade-offs in terms of standardization, energy, memory, complexity, form factor, cost, etc. to meet the proposed requirements.

If we look, for instance, at the energy cost of encrypting a 128-bit data packet and sending it over a ZigBee network, encrypting with dedicated hardware can be up to 20% more energy-efficient compared to software encryption [DOH+10, OHP+10a, OHP+10b]. However, for a higher transmission power, this gain will be significantly smaller. The fact that communication is in most cases responsible for the biggest energy consumption has been shown for several other wireless standards too [SSBV11].

It is clear that, in order to make educated choices, the designer (or design team) needs to be multi-disciplinary, bringing together knowledge of e.g., encryption algorithms, entity authentication protocols, communication standards, hardware design, energy management, ...

With the use of a flexible, re-usable hardware platform, a design team can prototype different design parts separately, integrate them, prematurely detect design flaws,

and quickly come to a proof-of-concept they can build on to get to the final product. This chapter describes our design of such a platform, where the main requirements are:

- Testing (new) security protocols in an embedded setup.
- Making practical proof-of-concept applications without having to worry about specific technical problems e.g., communication stack, hardware drivers,... but just focusing on the application itself.
- Designing and evaluating hardware accelerators to support cryptographic operations.

Contributions. Starting from an existing Xilinx FPGA development board, we have designed an embedded hardware platform for designing and evaluating embedded security. This platform offers the missing link between wireless development kits and embedded hardware development boards. The hardware is assembled from existing Xilinx IP cores, e.g., standard embedded hardware like GPIO, memory controllers, communication interfaces, and custom IP cores i.e., cryptographic accelerators (one of which built from scratch).

On top of the embedded processor, we have deployed an embedded Linux operating system. To this OS, we have made several additions:

- Driver libraries for the cryptographic accelerators.
- A hardware expansion board with two extra UART interfaces, one of which is used to control an NXP PN532 NFC chip.
- Communication libraries for NFC P2P.
- A boot sequence tailored to speed up development.
- Support for Compact Flash, NFS and NTP.
- Startup scripts to automatically enable this additional functionality.

With this platform set up, proof-of-concept applications can be rapidly deployed (see Chapter 9). In addition, newly designed hardware accelerator IP cores can easily be added in a similar fashion to the currently supported cores. The design of such a core is detailed in Chapter 4.

The next section explains our design choices. Then, we provide the details of the features we implemented. Section 2.4 then discusses how this design methodology is applicable to other embedded platforms. Section 2.5 finalizes this chapter with some conclusions.

Publications. A first version of this platform has been presented at *the 12th Joint IFIP TC6 and TC11 Conference on Communications and Multimedia Security - CMS 2011* in Gent, Belgium [OMS+11]. A first practical case with an updated

design has been detailed in an article and presented at the *XXI International Scientific Conference on Electronics – ET 2012* [ODW+12]. With regard to design choices on AES data encryption in ZigBee networks, several articles have been published [DOH+10, OHP+10a, OHP+10b].

2.2 Design Choices

2.2.1 The Xilinx ML605 FPGA Development Kit

Systems on Chip (SoCs) are used in many of today's embedded hardware designs. These SoCs integrate a complete embedded system into a single chip, housing a central processor, memory, timing sources, communication interfaces and other peripherals. Using a SoC allows to more easily manage the complexity of an embedded system. Designers can focus on the implementation of their specific hardware/software block, which is then used as a black box by others.

The prototyping of SoCs is typically done with FPGAs so engineers are able to validate and test the system in a real-life situation at (or close to) the operating frequency of the final design. This greatly reduces the development and debugging time of the digital design.

To offer the same features with our design, we use an FPGA development kit; more specifically the Xilinx ML605 embedded development kit [44] (Fig. 2.1). The main component of this development board is the Xilinx Virtex-6 XC6VLX240T FPGA [45]. This device houses a large amount of general purpose logic (e.g., LUTs, FFs, MUXs, ...), but also enough RAM for user applications (both hardware and software), hardware multipliers and Multi-Gigabit transceivers. The FPGA can be configured from one of the non-volatile memories present on the board, or by using the USB-JTAG interface for on-the-fly reconfiguration (and debugging) [19].

The non-volatile memories can also be used for user application data storage. To this end, there is also an interface provided for removable Compact Flash cards. Next to the on-chip block RAM, the board also offers a 512 M DDR3 SO-DIMM for user applications.

The board also hosts several communication interfaces, most notably an Ethernet PHY layer and UART-to-USB bridge. Typical GPIO hardware found in embedded applications such as switches, LEDs, and an LCD (2x16 characters) are also accessible from the FPGA. Extra pins of the FPGA are available from the FMC connectors.

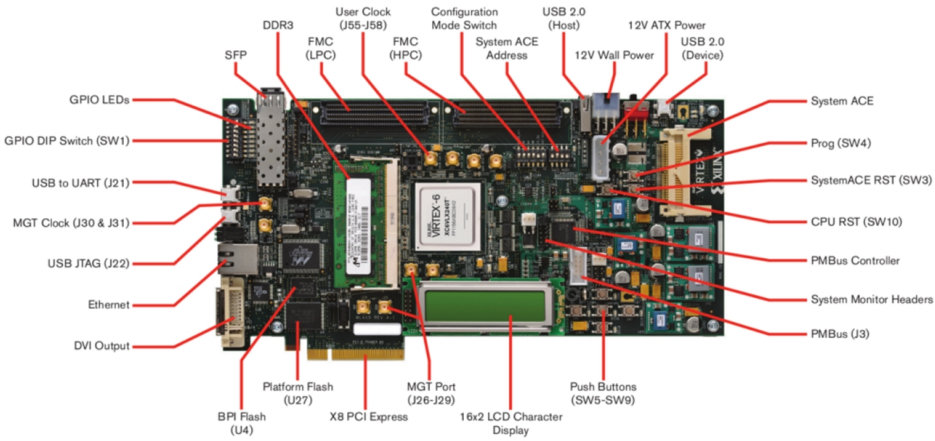


Figure 2.1: The ML605 development board, with the most important components (courtesy of Xilinx).

2.2.2 MicroBlaze Processor with Memory-Mapped Peripherals

Xilinx offers a processor IP core optimized for implementation in Xilinx FPGAs. This is the MicroBlaze™ RISC Harvard architecture embedded processor [TG06]. The MicroBlaze processor is highly configurable (see Fig. 2.2). Next to a fixed feature set, additional functionality can be selected when instantiating the core in the design [43].

To be able to run an embedded operating system, we have chosen to add a memory management unit (MMU) and to speed up most computations, we also included a barrel shifter and multiplier. In our design we use the processor local bus (PLB) interface to connect peripheral hardware to the controller. Currently, Xilinx has adopted the AXI4 interface as the embedded hardware interconnect.[1] However, migrating from PLB to AXI4 is rather straightforward when the hardware was created with the Xilinx tools. For existing PLB cores that need to remain unchanged, Xilinx also provides an AXI to PLB bridge [42].

We need to note that the MicroBlaze also supports a streaming data interface e.g., for video, audio and signal processing in general, with the FSL (legacy) interface [40] and current AXI4-Stream interface [10, 42]. Because of the nature of our envisioned applications, we have currently not implemented this interface.

[1]The reason why the PLB interface is used for this platform, is that at the development start date the PLB interface was the standard bus for Xilinx memory mapped IP cores.

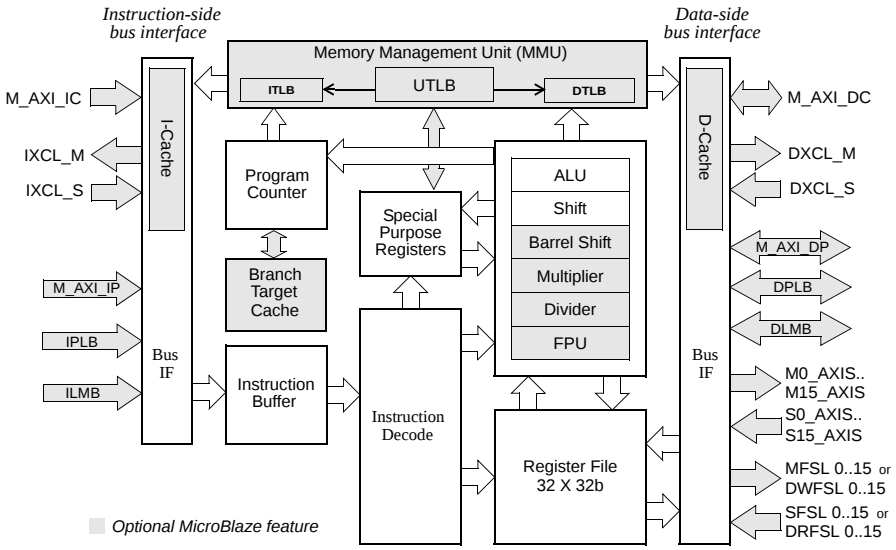


Figure 2.2: MicroBlaze core block diagram (courtesy of Xilinx).

All peripheral IP cores are accessible as memory-mapped devices from the PLB/AXI4 bus. This means that they can be controlled and used by reading and writing to their local registers and memory of which the addresses are a subset of the total available memory addresses. Like most microcontrollers the MicroBlaze uses a Harvard memory architecture. Next to the PLB/AXI4 bus, memory locations can be accessed with the local memory bus (LMB) and the Xilinx Cache Link (XCL). The LMB is used to access on-chip RAM while the XCL is typically used to interface with external RAM, using the multi-port memory controller (MPMC).[2] Both the LMB and XCL can be split to create separate instruction and data memory.

2.2.3 Embedded OS

To deal with the complexity of modern embedded systems, more and more of them run an operating system. Embedded Linux is highly suitable for that purpose.[3]

[2]For current designs, Xilinx has also adopted AXI4 to interface with the MPMC. [47]

[3]All embedded Linux-based distributions have a collective market share of 50%:

- EE Times' 2013 embedded survey: `http://www.eetimes.com/electronics-news/4407897/Android--FreeRTOS-top-EE-Times--2013-embedded-survey`
- Embedded developers prefer Linux, love Android: `http://linuxgizmos.com/embedded-developers-prefer-linux-love-android/`

Advantages of embedded Linux. The first interesting feature is the availability of a complete TCP/IP stack, together with a set of standard networking applications. This greatly increases the ease of developing and debugging applications on the embedded system as will be detailed in Section 2.3.

A second advantage (especially in comparison to stand-alone embedded applications) is the support for threading. Together with the MMU this makes that applications can be tested on the system each in their own memory space, without crashing the system because of bugs in the application. It is also another way of dealing with the complexity of the system and allows to partition and develop an application in smaller parts; e.g., a main application (running as a thread) relies on a communication stack (several other threads) and a service for writing to the LCD.

Another benefit of using embedded Linux is the availability of standard libraries for data and memory manipulation, arithmetic, communication (security), as well as the possibility to add non-standard tools and libraries available for Linux distributions; e.g., in the case study (Chapter 9) we use GMP [38] for several arithmetic operations and a ported version of libnfc [20] for the NFC communication.

A last advantage is the support for non-standard memory-mapped hardware devices. In Linux this is provided by the *userspace I/O* (UIO) kernel module. We will use UIO to control the hardware accelerators and general purpose peripherals.

The UIO kernel module. Creating a complete Linux kernel driver for devices that are not handled by standard kernel subsystems (like networking or serial or USB) is often overkill. Many devices only require some way to handle an interrupt and provide access to the memory space of the device. The logic of controlling the device does not necessarily have to be within the kernel, because the device does not need any of the other resources that the kernel provides. Originally designed for custom PCI-cards, the userspace I/O system now proves its worth on embedded systems. UIO only provides a very small kernel module to access device memory and handle interrupts. The rest of the driver functionality resides in user space.

Devices that are configured in the device tree source (DTS)[4] as UIO devices, will appear as `/dev/uioX` in the OS file tree, where `X` is a number. To obtain a file descriptor to these device, the following example code can be used:

```
int uiofd;
uiofd = open("/dev/uioX", O_RDWR|O_NONBLOCK);
```

[4]The DTS is a textual description of the hardware tree structure, where the CPU is the root, buses are branches and devices are leaves. Every device has several properties e.g., register address space, driver compatibility,... By setting the `compatible` property to `"generic-uio"`, the device is treated as a UIO device by the OS.

Subsequently, to gain access to the device's memory and read or write:

```
int uio_memptr;
uio_memptr = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE,
    MAP_SHARED, uiofd, 0);
// reading
int data =  *((unsigned *)(uio_memptr));
// writing
*((unsigned *)(uio_memptr)) = data;
```

To enable the interrupts, it suffices to write a '1' to the device (using the previously obtained file descriptor).

```
int enable = 1;
write(uiofd, &enable, sizeof(int));
```

A developer can check for interrupts by reading from the UIO device.

```
int ints = 0;
read(uiofd, &ints, sizeof(int));
```

When an interrupt has occurred, the interrupts have to be enabled again in source code. This is not done automatically.

All source code written to control the platform's UIO devices follows the same structure. However, the developer is responsible for freeing and un-mapping all allocated memory.

The PetaLinux distribution. Xilinx together with PetaLogix offer a Linux port targeted for Xilinx FPGAs and Zynq SoCs. The PetaLinux distribution comes with several board support packages and a tool chain to simplify the deployment on a broad range of platforms. It also automatically generates a DTS file from an embedded hardware design created with the Xilinx embedded development tools. The tool chain also offers the possibility to easily add new applications and libraries to the basic distribution. For that purpose it creates the necessary make files and kernel configuration files. Aside from this, the distribution comes with the lightweight but powerful Busybox shell which contains the most common Linux utilities.[5]

A license is required to run the PetaLinux SDK. We use an academic license, which is free of charge but with limited support.

[5] Busybox project website: `http://www.busybox.net/`

2.3 Implemented Features

With the ML605 development kit and embedded Linux running on a MicroBlaze processor system we created an example platform for testing and evaluating embedded security. The resulting block diagram is represented in Fig. 2.3. The implemented features are explained in detail in the following paragraphs.

2.3.1 Debugging and Application Development

Network file system support. A standard capability of Linux operating systems is the *network file system* (NFS). With NFS a remote directory can be mounted over the network and used as if it were physically located on the local host. We included NFS because it simplifies the embedded application development. A developer can write applications on a powerful work station, in his favorite development environment. The development directory is mounted on the embedded platform. After cross compilation, the application is immediately available on the embedded platform for testing, without the need to rebuild and download a complete kernel image. This greatly reduces the development time.

Remote application debug. With the GNU debugger (GDB) a program can be debugged, i.e., stepping, breaking execution,... One of the utilities that PetaLinux offers on the embedded platform is a GDB server. This way, applications running on the platform can be debugged remotely over TCP/IP. A developer can connect to the server from his workstation and debug an application as if it were running on the workstation itself.

2.3.2 Communication

TCP/IP. The PetaLinux distribution offers a complete TCP/IP stack. This is, for instance, necessary to support NFS and remote debugging. Standard kernel configuration is that the IP address is obtained with DHCP. However, Busybox offers a broad range of (standard Linux) utilities to make changes to the configuration e.g., `ifup`, `ifdown`, `ifconfig`,... Other networking tools like `ping` and `wget` are supported as well.

Another advantage of the TCP/IP stack is that application developers can use socket programming to access remote services as well as offer services on the platform itself.

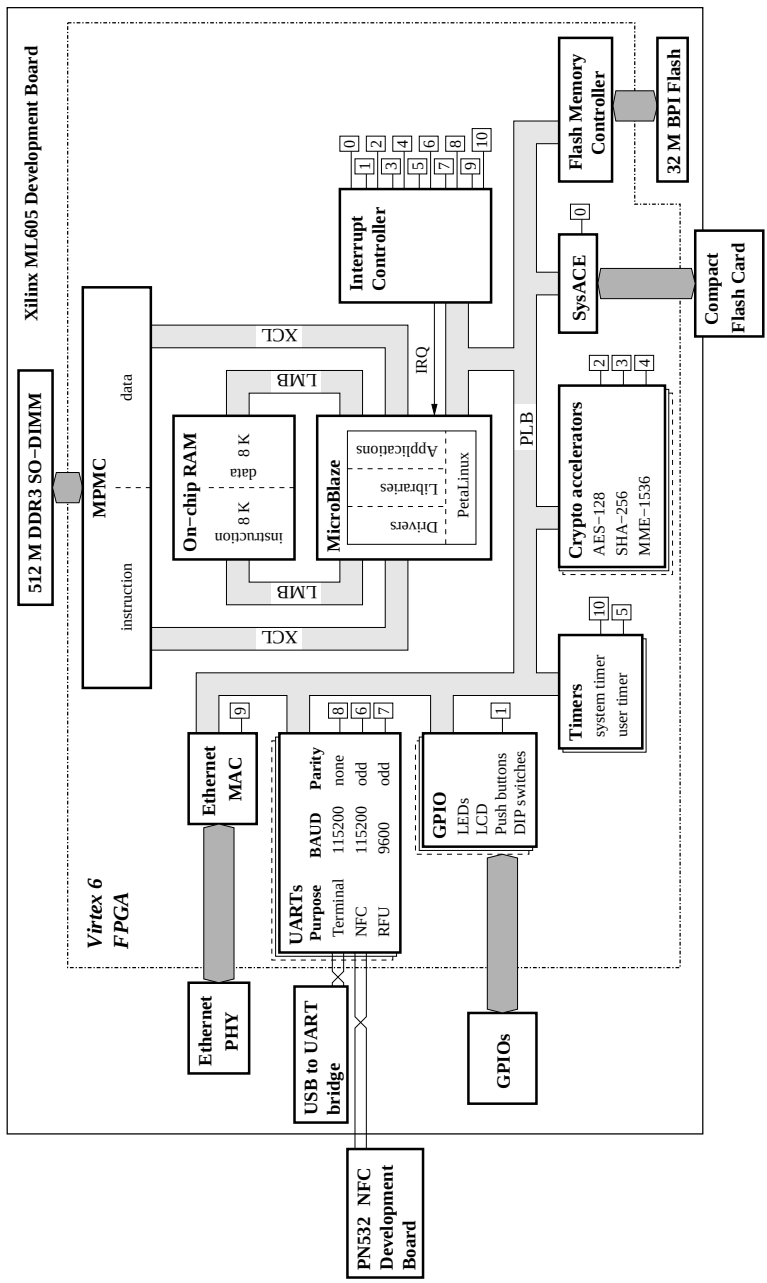


Figure 2.3: Block diagram of our test and development platform design

NTP. A necessity for many security applications is knowledge of the correct time, e.g., to determine if a certificate is still valid. With the *network time protocol* (NTP) computer systems can synchronize their clock with a time server over a packet-switched network with unknown latency; in practice using TCP/IP. We installed an NTP client application for this purpose.

RS-232. The ML605 platform provides a USB to UART bridge so a computer terminal can be used to monitor and/or control the system's behavior. In this particular setup, the USB/UART is used as a serial terminal to control the booting of the system as well as to issue shell commands (Busybox) to the OS. It also serves as the `stdin` and `stdout` for all applications.

We extended the ML605 with a hardware stack-up board so two extra RS-232 ports are available. One of these ports is being used to control the PN532 NFC communication module [32]. The other is reserved for future use (RFU). Note that the free-to-use UART IP core provided by Xilinx (`xps_uartlite`) requires a fixed baud rate, that needs to be set before design synthesis.

NFC. In Chapter 7 we discuss the selection of `libnfc` and `libnfc-llcp` to serve as NFCIP-1 and LLCP layers for the NFC P2P communication stack. We opted to keep these libraries separated on the platform as well and not to offer a single NFC P2P library. The main reason is that in case we want to communicate with a phone in card emulation mode, we only require `libnfc`.

Because `libnfc` has a lot of overhead i.e., not only support for the PN532 and not only support for UART, we reduced this library to the bare minimum. Currently the only optional feature that has been left in, is a small number of debug messages, though not the whole set.

As we mention in Chapter 7, the support for SNEP and NDEF has to be written by the application designer, but we provide an example on how to make a basic implementation (Appendix B).

2.3.3 System Timer and Interrupt Control

The operating system's scheduler requires some way to measure time. This is provided by the `system timer`. This 32-bit timer/counter generates an interrupt after a certain amount of time, predefined in software.

Because the MicroBlaze has only one interrupt input, an interrupt controller is required to capture and distinguish between interrupts from different sources. The priority of the interrupts is fixed in hardware and shown on the block diagram. The highest number corresponds with the highest priority. The OS can distinguish

Table 2.1: List of implemented general purpose devices.

Device description	UIO device	PLB base address
8-bit DIP switches	`/dev/uio0`	`0x81460000`
LCD 2x16[6]	`/dev/uio1`	`0x814a0000`
8-bit LEDs	`/dev/uio2`	`0x81440000`
5-bit LEDs	`/dev/uio3`	`0x81420000`
5-bit push buttons	`/dev/uio4`	`0x81400000`
user timer	`/dev/uio8`	`0x83a00000`

between the different interrupt sources by reading from the interrupt controller's status registers (over the PLB).

The interrupts from e.g., the system timer and the Ethernet MAC are processed automatically by the kernel subsystems. Interrupts from e.g., the cryptographic accelerators are caught by the UIO driver and processed by the user space driver.

2.3.4 General Purpose Devices

Support for typical hardware found in embedded applications is also provided. All these devices have been implemented as UIO devices. The push buttons and user timer interrupts can be captured by the operating system. Table 2.1 lists all the supported general purpose devices.

2.3.5 Memory

The FPGA's on-chip memory is used to create an 8 K instruction memory and an 8 K data memory. Currently, this is only used to store the First-Stage Bootloader (see paragraphs on booting Sect. 2.3.7), so 16 K is more than sufficient. The 512 M DDR3 RAM forms the main working memory of the operating system. It is accessible through the MPMC, but the OS in combination with the MMU, ensures that application developers don't need to worry about the implementation of the underlying memory architecture.

Using the System Advanced Configuration Environment (SysACE) IP core [39], the system can use a Compact Flash card as large non-volatile memory. Although originally designed for FPGA configuration, we only use the SysACE controller for

[6]HD44780-based LCD operated in 4-bit mode.

Table 2.2: List of implemented cryptographic accelerators.

Device description	UIO device	PLB base address
AES-128	`/dev/uio5`	`0xa1000000`
SHA-256	`/dev/uio7`	`0xa2000000`
MME-1536	`/dev/uio6`	`0xa0000000`[8]

Compact Flash storage. To be able to work with SysACE, the card needs to be FAT16-formatted.[7]

The ML605 also houses several non-volatile memory devices, both FLASH and EEPROM. We use the 32 M BPI Flash memory as configuration and second stage boot loader memory. It is, however, also accessible at run time, both from the second stage boot loader and operating system. This is to support in-the-field configuration updates, which are much faster than the Xilinx tools to program the Flash. The main reason is that for in-the-field updates the data is downloaded over TCP/IP, while the Xilinx tools use JTAG, which is much slower.

2.3.6 Hardware for Cryptographic Operations

To be able to support several cryptographic protocols in hardware, we added three hardware cores; one for symmetric key cryptography (AES-128 [23]) one for hashing (SHA-256 [25]) and one for asymmetric key cryptography (MME-1536). All cores have been implemented as UIO devices and are connected as memory-mapped devices to the PLB (see Table 2.2 for the details).

The AES-128 core is based on the design of Gaj [GC03]. It only implements the AES encryption/decryption [DR02]. Modes of operation have to be implemented by the controlling software.[9] This core has mainly been implemented to investigate how easy it is to add existing hardware designs to the platform. It is not currently being used in any case studies.

The SHA-256 core has been designed by Rykx and Thielen [RT09]. Only the compression function is implemented in hardware. The padding of the message has to be performed by the controlling software. A library containing the driver source (UIO-based) and a user API is added to the OS. The core is used in the

[7] To be able to read the FAT16 file system, we compiled the kernel including the required code page (`cp437`) and iocharset (`iso8859-1` [6]). This is not the standard PetaLinux kernel configuration and hence support needs to be enabled explicitly.

[8] The control register space starts from address `0xa0006000`.

[9] An up-to-date list of approved modes for approved block ciphers can be found on the NIST *Current Modes*: `http://csrc.nist.gov/groups/ST/toolkit/BCM/current_modes.html`

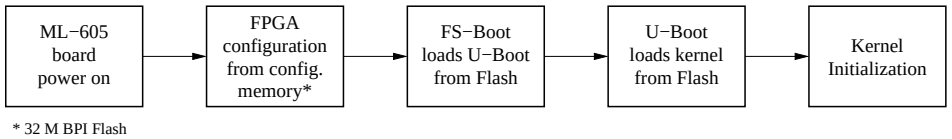


Figure 2.4: Typical platform boot sequence.

case study (Chapter 9) to compute the hash in the CL-based credential verification (see Chapter 8).

One part of this PhD focuses on the design of a highly configurable hardware accelerator for modular multi-base exponentiations for use with this embedded test platform. The MME-1536 core is an instantiation of this core, which supports modulus and operand lengths of 512 bits, 1024 bits and 1536 bits. The details of this core's design are described in Chapters 3 to 6.

2.3.7 Advanced Booting

Aside from configuring the FPGA and MicroBlaze processor from the JTAG debug chain, it is also possible to automatically configure the system when the board power is switched on. This automatic configuration is much quicker than the JTAG-based configuration, which is interesting when only developing embedded applications i.e., the hardware platform doesn't need to be changed.

To that end, we created an advanced boot setup that offers both speed and flexibility. We made use of two existing boot loaders and the on-board Flash memory. In addition, we also made changes to the kernel initialization.

Boot loaders. To get the operating system running at board power-up, a boot sequence as shown in Fig. 2.4 is required. At board power-on, the FPGA image (stored on the boot Flash memory) is used to configure the FPGA. After configuration, the on-chip RAM contains the FS-Boot (First Stage Boot loader) provided by PetaLinux. The FS-Boot takes the U-Boot boot loader[10] from the Flash memory and loads it into the DDR3 memory.

When not interrupted, U-Boot will automatically load the kernel image from flash to the DDR3 memory. It then transfers control to the kernel which assumes its normal operation after the initialization is complete. However, when a workstation (serial terminal) is connected to the board, a developer can interrupt the U-Boot automatic boot. With the (limited) U-Boot shell he can e.g., make changes to

[10] *Das U-Boot – the Universal Boot Loader* project website: `http://www.denx.de/wiki/U-Boot`

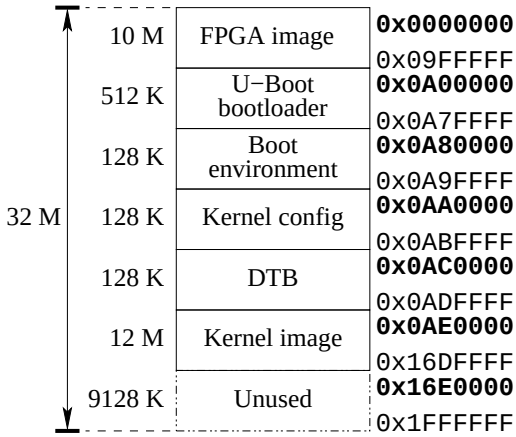


Figure 2.5: Organization of the 32 M Linear BPI boot flash memory.

the TCP/IP configuration, request a new kernel image over the network, boot the new image or update the Flash contents. The "net-boot" feature is particularly interesting when developing libraries, because they need to be built into the kernel. As long as changes are being made to the libraries, net-booting the newly built kernel is quicker than re-writing the Flash and performing a standard boot.

Boot Flash memory. The boot Flash memory organization we use for the boot sequence, is shown in Fig. 2.5. The PetaLinux tools provide a utility to configure the Flash contents. The user, however, needs to ensure (manually) that the reserved memory spaces are sufficiently large. A requirement for automatic FPGA configuration is that the FPGA image is located at address `0x0`. The FS-Boot also needs to be compiled with the correct start address of U-Boot. The boot environment contains environment variables required by U-Boot e.g., network configuration data or the net-boot. The DTB is a compiled version of the DTS.

The boot Flash can also be updated by the OS itself. This is a utility that is mainly interesting for in-the-field firmware updates.

Modifications to the kernel initialization. Next to the standard kernel initialization i.e., populating the device tree, loading kernel modules, setting up network interfaces, users (developers) can also add their own init scripts. These scripts need to be located at `/etc/rc.d/`. To be executed, the script name needs to be of the form `Sxx_my_custom_init_script_name` , where `xx` is a number. The number determines the order in the init sequence, where the lower numbers are executed first.

We added two init scripts to automatically enable some extra features in the system:

1. To automatically mount the workstation's NFS share as well as the Compact Flash card, the script `S50custom_mount` is executed during OS initialization:

```
#!/bin/sh
echo "Mounting extra's:"
# create mount points
mkdir /mnt/nfs
mkdir /mnt/cf
# mount NFS share
WORKSTATION=10.26.8.2
WORKDIR=/home/geoffrey/Petalinux/petalinux-v2.1-final-
    full/software/user-apps
mount -t nfs -o tcp $WORKSTATION:$WORKDIR /mnt/nfs
# mount Compact Flash
mount -t vfat /dev/xsa1 /mnt/cf
```

2. To make sure the system clock has the correct time, the NTP client is used to synchronize with a time server (in this case `be.pool.ntp.org`) at system start-up. This is done by the script: `S51ntp_starter` :

```
#!/bin/sh
NTPHOST=be.pool.ntp.org
echo "Getting time from timeserver: $NTPHOST"
ntpclient -s -h $NTPHOST
```

2.4 Design Portability

As mentioned in Sect. 2.2, the design is a typical SoC prototype design. It is not only applicable for the ML605, but every recent Xilinx FPGA (starting from the Spartan/Virtex 5 series) can be used to create a similar setup. Moreover, when migrating from PLB to AXI, this design can also be implemented on e.g., a Xilinx Zynq SoC [46]. These devices integrate a complete dual-core ARM Cortex-A9 combined with FPGA fabric, hence offering an more powerful CPU than the MicroBlaze.

Altera also offers an embedded microprocessor system with embedded Linux. The Nios II processor [7] is comparable to the MicroBlaze processor when it comes to processing power. However, it uses the Avalon switch fabric [8] as peripheral interconnect. This means that IP cores developed for AXI or PLB will have to be redesigned, or that an AXI/PLB to Avalon bridge will have to be used.

2.5 Conclusions

In this chapter we have detailed the design of test, prototyping and evaluation platform for embedded SoCs. This platform offers the features of both embedded hardware and wireless communication development boards. We have explained our design choices and provided practical details and discussed advantages and difficulties.

With regard to supporting cryptographic operations, we have added several custom hardware IP cores. The bus structure with memory-mapped peripherals in combination with the UIO driver offered by the operating system makes adding these peripherals and managing the system's complexity almost child's play in the test and development platform we have set up.

Currently, only one wireless interface has been implemented (i.e., NFC). With regard to future work, supporting other much-used embedded wireless standards e.g., Bluetooth, Bluetooth 4.0, WiFi, should be a priority. For offline platforms, it would be advised to add a *real-time clock* IC e.g., connected with I^2C or SPI, for keeping track of the time.

Chapter 3

Introduction to Hardware Support for Modular Multi-Base Exponentiations

3.1 Motivation

As we have pointed out in the introduction (Chapter 1), it is a challenge for today's design engineers to counter the security issues in newly developed applications on embedded wireless platforms. One of the issues is how to preserve the privacy of the users, especially in an embedded context where computational power is limited i.e., on the side of a stand-alone wireless terminal such as a vending machine or an electronic locker.

A privacy-friendly solution to access electronic services is offered by attribute-based credential systems. A more detailed overview of these systems is given in Chapter 8. However, we will briefly outline the difficulties with using attribute-based credential systems in an embedded context. The main disadvantage of these technologies is their poor performance, both in terms of processing power and in terms of memory footprint, compared to classic certificate technology. The `Identity Mixer` [17] for instance, uses zero-knowledge proofs that require multiple exponentiations by both the prover and the verifier (see Chapter 8). This operation is shown in Eqn. (3.1), where the length of the base operands g_i, and the modulus m, is at least 1024 bits. The length of the exponents e_i can range from a few bits to a few thousands of

bits.

$$\prod_{i=0}^{l-1} g_i^{e_i} \bmod m \ . \tag{3.1}$$

Especially these computations currently lead to unacceptable response times in many application domains. The same problems arise for other protocols like DSA signature verification or Direct Anonymous Attestation (DAA).[1] In the case of attribute-based credentials, the existing research has mainly focused on optimizing the prover operations and exploring the usability boundaries on mobile platforms such as smart cards and mobile phones. That approach assumes that the verification of credential proofs occurs at a powerful back end. An overview of this research is given in Chapter 8.

However, selective disclosure or anonymous authentication is also very relevant in settings where a user authenticates to a terminal that is not connected with a powerful back end. For example, a stand-alone cigarette or beverage vending machine wants to verify if the user is older than 18. Similarly, a local waste disposal center wants to permit access only to citizens of the area. Therefore, users need to prove that they live in a certain city before they are granted access to that location. Another application of attribute-based credentials could be in a self-storage facility where users only require a valid credential to access the storage. The credential, however, is only valid for a predetermined time.

In all the above-mentioned situations, users can discourage extensive profiling by releasing no more information than is strictly necessary. Another similarity is that hardware that performs the credential verification is most likely to be an embedded platform and hence limited in processing power.

Example of the problem

Devices like stand-alone terminals or access points are typically run by a (low-resource) microcontroller. An implementation on a 16 MHz 8-bit AVR microcontroller with 1024-bit base operands and 24-bit exponents needs 1.72 seconds for one dual-base exponentiation [Bal08]. In comparison, a SHA1 computation on the same system only requires 6 ms.

To enable developers to evaluate different security protocols that require multiple modular exponentiations (e.g., attribute-based credentials, DSA signature verification, DAA, or secret sharing protocols), we have designed a hardware accelerator – also referred to as (hardware) IP core – to be used specifically in an embedded context. Because often several exponentiations are required, we opted to

[1] DAA is a cryptographic protocol for remote authentication of a trusted platform comparable to attribute-based credentials: i.e., it preserves the user's privacy.

create a hardware accelerator that is capable of performing these exponentiations simultaneously.

Many designs of hardware for modular exponentiations have been proposed in the literature. However, they are always targeted to optimize a certain design goal, whether that is the footprint, speed, throughput, energy or a product of one of these. Furthermore, very few exist that are capable of carrying out exponentiations simultaneously. The same holds for commercially available products. In this chapter we will give an overview of this research and point out the findings that are most relevant for our design.

Because we want to be able to test our hardware accelerator with different scenarios, in a practical setup, our main design goal is a customizable IP core (see outline for examples of several scenarios). With the hardware accelerator connected to our embedded test platform (Chapter 2), these different scenarios can be easily evaluated and in addition, we can state the boundaries for certain design approaches: e.g., what is the computational speedup by using simultaneous multi-base exponentiations in this kind of applications.

Need for flexibility

Take the setting where a user rents a locker, for instance, in a train station. If the user has a previously purchased credential on his smartphone, he can open and close the locker with it. The credential only holds an expiry date (and of course a master secret).

In the case of a car rental or sharing application like, for example, *Cambio*, a user should only have to prove that he has a valid driver's license and a car insurance.

A purchase of cigarettes or liquor at a vending machine using a mobile phone requires a range check of the age e.g., older than 18.

Also the level of security (determined by the length of the modulus) is an important factor. It is realistic that for the the vending machines (relatively small amounts of money), a modulus length of 1536 bits suffices, while for car rental a longer modulus (at least 2048 bits) is required.

After this introductory chapter, we detail the design of our own IP core in (Chapter 4). We explain our design choices, how we made the design customizable, and how that translates into hardware. We also explain the Linux device driver code for interfacing with the hardware from the embedded OS.

In Chapter 5 we evaluate how the multiplier can be "tuned" in terms of resource usage and operational frequency. This allows us to somewhat compare our hardware

with existing material. Chapter 6 continues with including the complete IP core in the platform as described in Chapter 2. We state some theoretical run times and verify these in practice. The speedup by doing simultaneous multi-base exponentiations instead of single-base exponentiations will also be evaluated.

3.2 Related Academic Work

Efficient implementations of finite field arithmetic have long been investigated. They already received considerable attention due to their extensive use in the theory of error correction codes. However, research on this topic exploded after Diffie and Hellman [DH76] introduced the concept of public-key cryptography (PKC).

Since then, several well-known public-key cryptosystems have been proposed e.g., RSA [RSA78], DSA [24], ElGamal [Elg85], Schnorr [Sch90], `Identity Mixer` [17], `U-Prove` [35], ECC [Mil86] and HECC [Kob87]. They all rely on factoring and the discrete logarithm problem, either in a large prime field or on an elliptic curve over a finite field. The European Network of Excellence in Cryptology II publishes a yearly report on Algorithms and Keysizes [Eur12]. With more powerful devices coming up (cf. Moore's Law), larger keys are required to maintain a certain level of security. With the design of a test platform in mind, scalability of hardware implementation is a key factor.

As was also stated in the motivation (Sect. 3.1), the efficiency of an implementation of a cryptosystem depends greatly on the efficiency of the implementation of the algorithms. More specifically, in the case of the aforementioned PKCs, the modular arithmetic will play a decisive role. That is why a great deal of the concerning research has focused on modular multiplication. It is a key operation in the finite field arithmetic and forms the basis for a modular exponentiation, also commonly used.

An excellent overview of some techniques for efficient implementation of both integer and modular arithmetic can be found in The Handbook of Applied Cryptography [MVO96]. How finite field arithmetic is implemented efficiently in hardware and specifically applied to cryptography, can be found in both Guajardo et al. [GGK+06], and in Sakiyama and Batina [SB10]. Knežević' work [Kne11], continues to give a thorough overview of the research on how to improve the algorithmic efficiency of several hardware implementations.

The research on improving the efficiency always starts with improving the algorithms; by rearranging steps, by analyzing the requirements, and simplifying the algorithm or by coming up with new algorithms altogether. The next challenge is to implement them as efficiently as possible, either to reduce the footprint (fully sequential design) or to increase the speed (fully parallel design).

In the next section we give an overview of the evolution and interesting milestones in the design of modular multipliers, and additionally exponentiators. This will help us make some well-funded choices in the design of our own hardware implementation. Some of the commercially available products are discussed in Sect. 3.3. Section 3.4 concludes this chapter.

3.2.1 Modular Multiplication Hardware

The basis: Barrett and Montgomery multiplication. From an algorithmic point of view, two distinctive implementation types are used: *Bit-parallel* and *Digit-serial*. Bit-parallel algorithms are focused on reducing the run time of a multiplication. High speeds are achieved by expensive operations like integer division.[2] The trade-off is an increased area utilization. In contrast, digit-serial algorithms reduce the area by sequentially interleaving multiplication with reduction. Also, less complex operations like addition and shifting are used. The sequential character of these algorithms obviously has an impact on the execution speed. Mainly due to their reduced area (less-expensive), digit-serial algorithms are used in embedded applications.

Two algorithms have contributed to efficient implementation of modular arithmetic. In 1985, *Montgomery* came up with an idea to carry out a modular multiplication without the need of a trial divivision [Mon85]. His algorithm omits the classical reduction step. The result is given by Eqn. (3.2), that defines the Montgomery multiplication ($*$) as follows:

$$x * y = xyR^{-1} \bmod m \ . \tag{3.2}$$

Where, $R = b^n$ with b the base and n the number of digits of m. The algorithm is highly suited for use in modular exponentiations as the factor R can be easily removed at the end by (Montgomery) multiplying with R^2, which can be precomputed and is constant as long as the length of the modulus m does not change.

The approach of *Barrett* [Bar87] is different in the sense that it uses an estimate for the intermediate quotient. As a consequence, a number of correction steps might be needed at the end. However, Dhem [Dhe98] shows that at most one correction step is needed, provided that the correct values for some algorithm parameters are chosen. Just like with Montgomery multiplication, the efficiency of Barrett multiplication increases with the number of multiplications under the same modulus as the precomputation cost is fixed. Bosselaers et al. provide a thorough comparison of Barrett and Montgomery multiplication [BGV94]. They state that for general modular exponentiations, Montgomery's algorithm performs best.

[2] Expensive on embedded hardware.

Speeding up Barret en Montgomery multiplication. In the case of ECC, the modular reduction can be performed more quickly than in the generalized algorithms of Barrett and Montgomery. This is due to the properties of the Mersenne prime moduli used in such cryptosystems. This computational advantage is the reason behind the use of such moduli in standards like ANSI X9.62 [9], FIPS 186-3 [24] and SEC2 [34]. Quisquater [Qui92] came up with the first practical algorithm to simplify the reduction step. Later, Lenstra [Len98] proposes a method to generate RSA moduli that also simplify the reduction step while maintaining the security level. This method has later been enhanced by Joye [Joy08]. A high-performance FPGA implementation that combines the advantages of Mersenne prime moduli with the features offered by the DSP slices is presented by Güneysu and Paar [GP08].

Specifically for high-speed hardware implementations of ECC, Knežević et al. [KBV09] propose some sets of moduli that make it possible to implement both Barrett and Montgomery multiplication without the precomputational phase. Additionally, their results also show substantial improvement in case of a small number of reductions.

In an attempt to achieve higher execution speeds, an algorithm is proposed that combines Montgomery and classical multiplications. The first version of Potgieter [PvD02] only works with finite fields of characteristic 2. This was extended by Kaihara and Takagi to the ring of integers [KT05]. The so-called bipartite modular multiplication (BMM) efficiently combines Montgomery's modular multiplication algorithm with a classical modular multiplication method. Higher execution speeds are achieved by splitting the operand multiplier into two parts which enables parallel processing. Similarly to previously discussed techniques, the calculations can be sped up further by using special sets of moduli [KVV10]. The k-partition method by Néto et al. [NTR11] is an extension to the bipartite algorithm. In this method, k partitions operate in radix 2^k, each computing a part of the total result. The fastest multiplication would execute in n/k cycles. The complexity of the partitions, however, is higher than for a standard Montgomery multiplier, but often the FPGA's on-board multipliers are used to implement the partitions.

Further parallelization is achieved by Sakiyama et al. with the *tripartite modular multiplication* method [SKF+11]. As its name suggests, this algorithm combines three algorithms: the classic modular multiplication, the Montgomery multiplication and Karatsuba's algorithm [KO63]. The latter allows for even more parallelism by reusing intermediate partial products. While the tripartite algorithm provides ample parallelism and thus the possibility to fully speed up multiplication, it also allows for a speed/area trade-off.

Efficient implementation. In spite of the existence of several alternative algorithms (e.g., Barrett, bipartite, ...), the Montgomery algorithm is one of the most popular modular multiplication algorithms. This is mainly due to the

fact that it easily translates into hardware. However, the original Montgomery algorithm's main drawback is its carry propagation in the addition; especially for long multiplicands.

Several architectures have been proposed that combine Montgomery multiplication with either a redundant radix number system [SKN08, HC09] or the Residue Number System [BDK97, Phi01] to cope with this problem. However, these implementations have several other drawbacks. The most remarkable may be the fact that a lot of preprocessing of the operands is required.

Montgomery multipliers can use Booth encoding [Boo51] to replace the use of precomputed hard multiples in the processing elements. In [PH07], this is combined with left-shifting of the input operands –instead of right-shifting the result– to reduce the critical path. In [APH08, PH08], more flexibility is added to the design by varying the length of the processing elements and by implementing the Booth encoding in hardware rather than requiring precomputed hard multiples of the input operands.

In 1996, Koç et al. [cKKAJ96] made a first classification of implementations of Montgomery multiplication. Their classification is based on two features. The first feature is whether multiplication and reduction were interleaved or not. The interleaving could happen on a *coarse-grained* or *fine-grained* basis, depending on how often there is a switch between multiplication and reduction. The second feature is the formal structure of the algorithm; *operand scanning* refers to an outer loop that moves through words of one of the multiplicands, whether *product scanning* loops through words of the product. Their conclusion was that coarse-grained integrated (interleaved) operand scanning is best-suited for a general purpose processor. However, they suggested that for DSPs or dedicated hardware a fine-grained approach would perform better.

Inarguably, the best-known class of hardware Montgomery multipliers is the systolic array architecture. A first design milestone in that multiplier architecture was the work of Blum and Paar [BP99]. Nedja and Mourelle [Nd06, dMMN05, NdMM03] have performed a lot of work concerning practical implementations of systolic array implementations. They have shown that for operands larger than 512 bit, the systolic array implementation improves the *time* $\times$ *area product* over other implementations.

Systolic array Montgomery multipliers have been implemented in 2-dimensional (e.g., [Nd06, Nd02]) and 1-dimensional designs (e.g., [OBPV03, TcKK07]). The 1-dimensional variant requires less silicon to implement, but is slower than the 2-dimensional implementation. In 1999, Tenca and Koç [TcKK99] came up with a pipelined design where the circuit of the Montgomery multiplier is split into "word size" processing elements. This approach made the design scalable in terms of both processing speed and multiplicand length. Since then, several improvements have been made to their original design [TcKK03, ZW10]. One of the fastest

Montgomery multiplier designs combines pipelining and the FPGA's hardware multipliers, further increasing the execution speed compared to the previously available designs [MSPV07].

Practical concerns. While a lot of the aforementioned (pioneering) research put its emphasis on either optimizing for speed or area, there are also practical considerations to make. Consider, for instance, the work of Fournaris and Koufopavlou [FK07]. They propose a reusable ECC arithmetic unit that supports basic EC point operations and can thus be used to create a fully functional EC coprocessor. However, data flow to and from this processor also needs to be taken into consideration and is (again) highly dependent on the application requirements. For instance, in the case of high-throughput applications, it may be interesting to integrate the hardware accelerator in such a way that it is part of the processor data path.[3] In contrast, for applications that only require hardware offload in bursts, an accelerator designed as memory-mapped peripheral might be more appropriate. Fan et al. [FBV11] illustrate this point with a unified HECC multiplier and inverter design for both high-performance and lightweight applications.

Another practical aspect is the resistance of a design against side-channel attacks [Koc96, KJJ08]. By the way a design is implemented, it can leak information to the outside world over unintended channels, making them prone to physical and electrical attacks [HiHA13]. In his CHES 2007 paper [Joy07], Joye presents algorithms for multiplication that are protected against SPA-type attacks and safe-error attacks. A comprehensive text on side-channel analysis is written by Örs et al. [OPV07]. It gives insight in how the attacks are staged and what countermeasures can be taken.

For many embedded implementations, the footprint of a design is a key feature. Here, ECC can offer an alternative to RSA-based PKCs. One of the smallest ECC processors on FPGA is presented by Vliegen et al. [VMG+10]

It may be clear that making an efficient implementation of a cryptographic protocol is never a story of only algorithmic optimization or only hardware optimization. It requires a view on "the bigger picture". Or as Verbauwhede states [Ver12]: ... *on embedded devices a holistic design approach needs to be applied. This design approach has to look at each abstraction level during design. It includes protocol, algorithm, architecture, arithmetic and circuit level optimizations.*

3.2.2 Modular Exponentiation Hardware

The most straightforward way of performing a modular exponentiation is by repeated squarings and multiplications to get the final result [BP01, SDI11]. That

[3] For a MicroBlaze processor this can be achieved at some level by using an FSL connection [40].

is why most of the research effort went to the design of efficient multipliers. The square and multiply step can be performed either in parallel [Nd06] or sequentially [dlPTC11]. Again, this has repercussions on the footprint of the design. Where square and multiply are performed in parallel, two multipliers will be required. Blum also states in his thesis [Blu99] that squaring and multiplying in parallel does not fully utilize the complete design. This is because a multiplication will only be executed when an exponent bit is one; i.e., on average half the time.

This fact can also be exploited to reduce the execution time (of sequential implementations). Chang and Lai use Booth's method [Boo51] to increase the number of exponent 0-bits [LC03]. This reduces the number of multiplications and thus results in a speedup. However, more memory is required to store precomputed values; e.g., a dual-base exponentiation with a classic simultaneous exponentiation algorithm requires 4 memory locations while Chang's algorithm requires twice as many.[4] Chang and Lai also proposed a parallel modular exponentiation scheme (also using recoded exponents) that is specifically targeted for multi-processor systems [CL05].

To speed up simultaneous exponentiations even more, new exponentiations using complex arithmetic have also been proposed. In 1999 already, Dimitrov et al. came up with a first algorithm using complex arithmetic [DJM97]. In 2007, Wu et al. proposed a new algorithm based on modified complex arithmetic [WLLC07]. Both algorithms, however, have not been implemented in hardware, mainly due to their complexity.

Since Montgomery-based implementations are susceptible to timing attacks, Walter [Wal99] proposes a constant time implementation of a Montgomery Exponentiation. By leaving out the conditional final reduction, Hachez and Quisquater [HQ00] have proposed an improved version of this implementation.

3.3 Commercial Products

3.3.1 Trusted Platform Modules

... The heart of a trusted computer system is the Trusted Computing Base (TCB) which contains all the elements of the system responsible for supporting the security policy (...) Thus the TCB includes hardware, firmware and software critical to protection and must be designed and implemented such that system elements excluded from it need not to be trusted to maintain protection. ... [33]

[4] The memory locations are all of the same size as the modulus.

This means that, in order to build a secure architecture, you need at least one piece of secure hardware that you can build on. The Trusted Platform Module (TPM) specification [14] by the Trusted Computing Group (TCG) specifies such a piece of hardware. So in fact the TPM can do more than just some finite field arithmetic. It can be used to generate a hash value or an RSA key. It also facilitates remote attestation and sealed storage of keys.

Several manufacturers offer TPM devices. We discuss some devices that are widely used, also by other manufacturers like Intel or Broadcom. They all implement the current version (1.2) of the TPM specification.

The Atmel® AT97SC3204 [11] is a fully integrated security module for both computer systems and embedded systems. An LPC interface is used for communication between the TPM and the rest of the system. The TPM is internally controlled by an AVR® RISC microprocessor. Aside from secure key storage and hashing and signing operations, it also provides a random number generator (RNG) compliant with the FIPS 140-2 specification [22].

STMicroelectronics® offers two TPM devices. The low-cost ST19NP18-TPM [36] is based on the ST19N smart card platform which has an 8-bit architecture. The 1088-bit modular arithmetic processor allows for squaring, multiplication and addition of up to 2176-bit operands. A CRC module, hash generator and RNG are also included. The ST19NP18-TPM is designed specifically for PC platforms and BIOS security.

The high-performance ST33TPM12SPI [37] is ARM-based (32-bit @ 30 MHz) and designed for embedded security; hence the SPI interface. In addition it also has an AES-128 (CTR mode) core and an DES accelerator.

Infineon's SLB9635TT description does not contain any information about the execution speed. We have sent a request for information and are currently waiting for a reply.

More details about these devices can be found in Table 3.1. Note that the execution times for hashing and signature generation do not include communication with the controlling software application.

3.3.2 Cryptographic Accelerators

Hardware security modules (HSMs) are comparable with TPMs when it comes to functionality and architecture. However, where TPMs are mostly used for bootstrapping security, HSMs are used to accelerate cryptographic operations in high-speed or high-throughput applications.

The IBM® 4765 PCIe Cryptographic Coprocessor [18] is a such an HSM

Table 3.1: Comparison of different TPMs and HSMs.

	Interface	RSA signature		HASH
AT97SC3204	LPC 33 MHz	2048 bits	1024 bits	SHA-1
(Atmel)		200 ms	40 ms	20 µs [a]
ST19NP18	LPC 33 MHz	2048 bits	1024 bits	SHA-1
(STMicroelectronics)		382 ms	189 ms	unknown speed
ST33TPM12SPI	SPI 10 MHz	2048 bits	1024 bits	SHA-1, SHA-256
(STMicroelectronics)		150 ms	30 ms	155 µs [4]
SLB9635TT	LPC 33 MHs	?	?	?
(Infineon)				
4765 PCIe	PCIe x4	2048 bits	1024 bits	SHA-1, SHA-256, ...
(IBM)		348.87 [b]	1067.62 [4]	
		922.63 [c]	1631.30 [4]	
SafeXcel^{TM} PK IP	32-bit bus IF	?	1024 bits	Not Available
(Authentec)			0.95 ms[d]	
			16.3 ms[e]	

[a] For a 64-byte block
[b] calls per second, single-threaded
[c] calls per second, multi-threaded (7 threads)
[d] 33 PEs
[e] 4 PEs

suitable for high-security and high-speed cryptographic operations. Typical applications include financial applications such as PIN generation and verification in automated teller and point-of-sale transaction servers, but also Internet business, secure Web-serving applications and any PKI application in general.

Designed as a PCIe plug-in card, the "4765" can be used to offload computationally intensive cryptographic processes from a hosting server, and to perform sensitive tasks unsuitable for less secure general-purpose processors. The tamper-responding programmable secure hardware includes sensors to protect against attacks involving probe penetration, power sequencing, and temperature manipulation. As such it is certified under the FIPS 140-2 standard (level 4).

Aside from large number modular math functions for RSA (up to 4096-bit), ECC Prime Curve and other public-key cryptographic algorithms, it supports symmetric-key algorithms like AES, DES. Together with hardware for hashing and random number generation, this forms a complete secure computing environment.

The Authentec SafeXcel^{TM} IP Public Key Accelerators [12] are sold as a

design only (so-called Intellectual Property). The accelerators contain a multiplier-based Public Key Crypto Processor (PKCP) and, in most configurations, a Large Number(Montgomery) Multiplier and Exponentiator (LNME). This LNME consists of a selectable number of Processing Elements (PEs), that operate simultaneously in a pipelined manner. The number of PEs can be varied, allowing customers to make a trade-off between power consumption, gate count, and performance. The core can run at maximum frequencies of 230 to 250 MHz, provided that the design is implemented on high-speed 0.13 µm technology. What is remarkable is that the number of PEs only seems to influence the gate count and the number of cycles required for an operation and not the maximum clock frequency.

Authentec claims that the architecture and implementation of the LNME offers resistance against power and timing attacks. According to the product brief schematic, the IP core has a 32-bit memory-mapped bus interface with separate input and output.

3.4 Conclusions

Modular arithmetic, and specifically modular multiplication, has seen a long history of algorithmic optimizations and hardware designs. A lot of the optimizations are targeted to speeding up the design and make use of assumptions on e.g., the structure of the modulus. Doing so, however, limits the range of applications for which the multiplier design can be used. Other optimizations involve recoding or combination of algorithms to achieve higher execution speeds. However, this recoding increases the complexity of the implementation and consequently results in a higher area cost.

It is remarkable, but not surprising, that designs that are targeted for light-weight PKC, prefer (H)ECC for the cryptographic operations. This allows for smaller designs (shorter key lengths) and the use of e.g., special moduli. In contrast, most multipliers for non-ECC-based applications are optimized for speed. If we want our design to be applicable for a broad rang of protocols, it is clear that we cannot make any assumptions on the properties of the input data.

Many practical designs for large-number arithmetic use a systolic Montgomery multiplier of some sort. The main reason is the low complexity, which is essential for scalability. For large multiplicands (larger than 1024 bits), a 2-dimensional structure becomes highly impractical. However, if speed is a key factor, parallelism is mostly introduced in the exponentiation by executing the square and multiply steps in parallel. This again comes with an increased area cost which is not preferable in an embedded context.

Regarding simultaneous exponentiation, the proposed algorithms (academic)

require complex hardware or a great amount of memory (in comparison to classic designs). Commercially, there is no hardware available that is able to carry out exponentiations simultaneously. To avoid a time-consuming redesign of both hardware and software, we prefer an architecture that requires the same features and has the same complexity for simultaneous and single exponentiations. This way, we are able to test and compare both computations under the same circumstances.

Chapter 4

Design of an Open-Source IP Core for Modular Simultaneous Exponentiation

4.1 Introduction and Design Goals

As seen in Chapter 3, the computations required for attribute-based credentials systems vary in complexity depending on the kind of application. Sometimes it suffices to do a verification of a credential with a single attribute, while other applications require a range proof or a verification of credentials with several attributes, where some are revealed and others remain hidden. The main computation, however, is always a product of a series of exponentiations, the number of which is e.g., dependent on the number of attributes.

This chapter describes the design of a hardware accelerator (a hardware IP core) for modular simultaneous exponentiations. In practice, the IP core will compute a dual-base exponentiation (4.1) simultaneously:

$$g_0^{e_0} \cdot g_1^{e_1} \bmod m \ . \tag{4.1}$$

This IP core is designed with several requirements in mind. First of all, the circuit should be fit for operation on an embedded platform; i.e., it is controlled by a microcontroller or embedded processor.

A second goal is to provide the design as an open-source IP core. To ensure

portability to other platforms[1], we have specifically opted not to use any device-specific FPGA primitives like multipliers. The reason for this is twofold. First of all it allows other researchers to proceed in our work, both for the hardware design as for the applications. Secondly, a generic hardware description makes it easier to take the hardware design to a next level i.e., an ASIC or full-custom design. As a consequence our design might be out-performed by device-tailored designs (whether they are optimized for speed or area). However, the VHDL is written in a modular and structural way; any module can be easily replaced with a device-specific primitive or user-defined module. Well-documented code aides in customizing the design.

Finally, we want the design to be adaptable to the user's needs. To this end, we have written the VHDL in such a way that several parameters can be chosen before synthesis. The reason for this is twofold: on the one hand the open-source nature of the project and on the other hand it enables us to easily examine the design's resource usage and execution times for different parameter values. This will enable us to formulate some general guidelines for tuning the design parameters to the desired performance.

Aside from tuning the operational conditions, flexibility at run time should also be guaranteed. Varying the modulus length and exponent length without the need to do a time-consuming redesign or synthesis allows for practical tests in a comparable setting. Since the number of exponentiations depends on the application, support for single-base exponentiations and multiplication is also a must.

The developed modular exponentiation accelerator is designed to be part of a larger embedded system. To that end we need a software interface through which applications can use the IP core. The library containing this interface and low-level driver source code will be discussed in Section 4.5.

Publications. The first design has been presented as a poster on *the 8th Worshop on RFID Security and Privacy 2012* in Nijmegen, the Netherlands [Ott12]. An updated design has been detailed in an article and presented at *the 9th International Symposium on Applied Reconfigurable Computing* [OPGS13]. An article that describes the first tests of the IP core as part of our embedded test platform is published online at *EDN* [OPS+13]. The full technical documentation of the latest version of the IP core design is published on *OpenCores* [OC].

[1] Different devices and different manufacturers.

4.2 Simultaneous Multi-Base Exponentiation

An efficient way of performing a modular exponentiation is the *Montgomery exponentiation algorithm* which uses the *Montgomery multiplication* as the main computation step. An advantage of Montgomery multiplication over other techniques is that it requires no division, which simplifies the hardware considerably. The exponentiation can be extended to an efficient simultaneous exponentiation algorithm. The case with 2 bases is presented in Algorithm 1, but this can be easily generalized.

The modulus m and the bases g_0 and g_1 all have a length of n bits, whereas the length of the exponents e_0 and e_1 is w bits. The algorithm also requires $R^2 \bmod m$ which is $2^{2n} \bmod m$. We assume that R^2 can be provided i.e., precomputed in software.

Algorithm 1 Montgomery simultaneous exponentiation

Input: g_0, g_1, $e_0 = (e_{0_{w-1}} \cdots e_{0_0})_2$, $e_1 = (e_{0_{w-1}} \cdots e_{0_0})_2$, $R^2 \bmod m$, m
Output: $g_0^{e_0} \cdot g_1^{e_1} \bmod m$

1: $\tilde{g}_0 := \text{Mont}(g_0, R^2)$, $\tilde{g}_1 := \text{Mont}(g_1, R^2)$, $\tilde{g}_{01} := \text{Mont}(\tilde{g}_0, \tilde{g}_1)$
2: $a := \text{Mont}(R^2, 1)$ ▷ This is the same as $a := R \bmod m$.
3: **for** $i \leftarrow (w-1)$ **downto** 0 **do**
4: $\quad a := \text{Mont}(a, a)$
5: $\quad$ **switch** e_{1_i}, e_{0_i}
6: $\quad\quad 0, 1 :\ a := \text{Mont}(a, \tilde{g}_0)$
7: $\quad\quad 1, 0 :\ a := \text{Mont}(a, \tilde{g}_1)$
8: $\quad\quad 1, 1 :\ a := \text{Mont}(a, \tilde{g}_{01})$
9: $a := \text{Mont}(a, 1)$
10: **return** a

What draws attention is that, apart from the main loop (line 3), we need several multiplications for the precomputation of $\tilde{g}_0$, $\tilde{g}_1$ and $\tilde{g}_{01}$ (lines 1, 2) and one final multiplication (line 9). A logical design choice would be to only implement a multiplier and to implement some control logic in such a way that it can either run a single multiplication or run the main loop automatically.

Table 4.1: Example of the required factors in case of simultaneous exponentiation with 3 bases.

Name	Value	Exponent bits when used ($e_{0_i}e_{1_i}e_{2_i}$)
a	intermediate result	000
$\tilde{g}_2$	$(g_2) \cdot R$	001
$\tilde{g}_1$	$(g_1) \cdot R$	010
$\tilde{g}_{12}$	$(g_1 \cdot g_2) \cdot R$	011
$\tilde{g}_0$	$(g_0) \cdot R$	100
$\tilde{g}_{02}$	$(g_0 \cdot g_2) \cdot R$	101
$\tilde{g}_{01}$	$(g_0 \cdot g_1) \cdot R$	110
$\tilde{g}_{012}$	$(g_0 \cdot g_1 \cdot g_2) \cdot R$	111

Some remarks on precomputation, postcomputation and required memory

To counter the disadvantage of Montgomery multiplication, (i.e., the factor R^{-1} in the final result), every factor used in the simultaneous square-and-multiply algorithm is multiplied with R (or Montgomery multiplied with R^2). This ensures that all the intermediate results will also carry this constant factor R. To get rid of this constant, it suffices to (Montgomery) multiply the final intermediate result with 1.

To ensure the quickest operation and/or to reduce bus traffic, the required factors should be stored locally in the core's dedicated memory. These values are the bases, all combinations of products between the bases and the intermediate result. The amount of memory, however, increases exponentially with the number of simultaneous exponentiations (called l). In Table 4.1 all required factors are listed in the case of $l = 3$ simultaneous exponentiations. It is easy to see that in general one requires 2^l n-bit memory locations to store all necessary values.

4.3 Implementation Strategy

Following a standard design method, we implemented the IP core as a memory-mapped peripheral. A bus interface is provided so the processor can control the hardware as well as write data to and read data from the hardware core. The bus interface itself is split into two parts. A generic part with a 32-bit data input and output, an address line and control signals to the core; this part is fixed. On top of that interface is the actual bus interface e.g., PLB [41], AXI [10] or Whishbone [16].

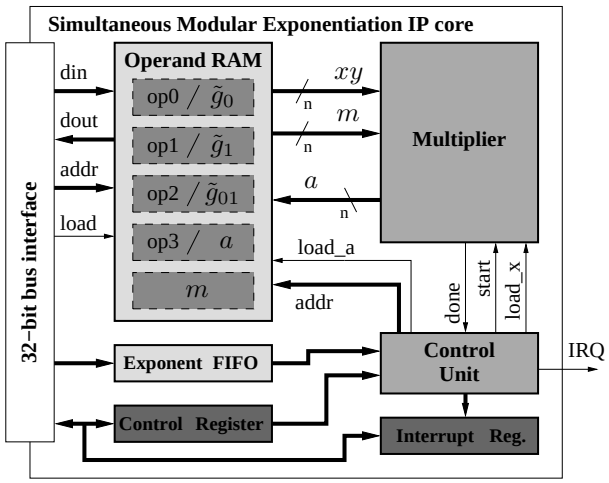


Figure 4.1: Block diagram of the simultaneous modular exponentiation IP core with all the main components.

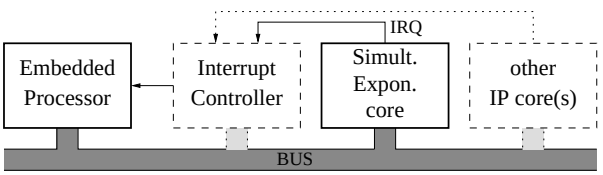


Figure 4.2: Embedded processing system setup: The IP core interfaces with the processor over a bus and it can generate interrupts.

The hardware can signal the processor of certain events using an interrupt line (IRQ) – optionally through an interrupt controller. This basic setup is shown in Fig. 4.2.

The kernel of the IP core is a Montgomery multiplier. As stated in Sect. 3.2, a systolic array multiplier is preferable when the base operands are larger than 512 bits; this is the case in most of the current RSA-based PKCs. Because resources increase proportionally with the length of the base operands, a 1-dimensional array is preferable over a 2-dimensional array, where resources increase quadratically. To further increase execution speeds, we applied pipelining. The design of this multiplier is elaborated in Sect. 4.4.

The operand length n and the number of pipeline stages k (see Sect. 4.4) can be chosen before synthesis. However, the hardware is designed in such a way that choosing different operand lengths at run time is also possible. This is done providing logic that allows to split the multiplier pipeline into an upper and a lower

part. At run time, users can choose to either use the upper or lower part, or the entire multiplier. This is an interesting feature when different operand lengths are needed in the application (e.g., different levels of security), because the execution time of a multiplication (and hence the exponentiation) is determined by n. It also facilitates extensive testing because there is no need to change the hardware configuration every time another operand length is used. It needs to be noted that calculations can only take place with operands of equal length.

To reduce bus data traffic between the core and the system controller, we provide RAM (Random Access Memory) to store several operands and the modulus. This RAM is implemented as dual-port memory. One port provides a 32-bit interface for reading and writing data over the 32-bit bus. The second port has a full n-bit interface to the multiplier. The control logic selects the multiplicands x and y from the operands and determines the destination operand of multiplier result a. When performing a single multiplication, the controlling software on the embedded processor (Fig. 4.2) has full control over the operation of the core. Any operand can be multiplied by another operand (even by itself) and stored in any location.

For maximum flexibility at run time, the exponent length w is determined only by the controlling software.[2] This is supported in hardware by the use of a FIFO. The exponent FIFO should be filled (at least partially) before starting the exponentiation. The exponents are split into 16-bit parts and stored in the FIFO's 32-bit entries, where the 16 most significant bits are e_1-bits and the 16 least significant bits are the corresponding bits of e_0. The IP core can be used for a single-base exponentiation by setting the e_1 or the e_0 bits to '0' and ignoring operands 1 and 2 or operands 0 and 2 respectively.

The software on the embedded processor can manage the core's behavior by writing to a control register (Fig. 4.1). When using the core to perform a (simultaneous) exponentiation, the control logic looks at the exponent bits stored in the Exponent FIFO, and selects the multiplicands and destination accordingly (main loop in Algorithm 2). This means that the controlling software needs to precompute the required values and store them in the correct locations before starting the exponentiation. Operand 0 should contain $\tilde{g}_0$, operand 1 should contain $\tilde{g}_1$ and operand 2 should contain $\tilde{g}_{01}$. Operand 3 is used to store the (intermediate) result a, but before starting the exponentiation, it should contain $R \bmod m$.

The core generates an interrupt when a multiplication is completed. In automatic exponentiation mode, it generates an interrupt only when the last multiplication is completed *i.e.* when the FIFO is empty. Other events like a full FIFO buffer or a RAM access conflict, also generate an interrupt. The controlling software can distinguish between the different interrupt sources by checking the interrupt register.

[2]This is a practical necessity. For instance, a revealed attribute can be 1 bit long (e.g., the sex of a person), while a hidden attribute will be several hundreds of bits in length.

Open-source design

The complete VHDL design, including full technical documentation and test benches for multiplication and exponentiation, is published online at the open-source website *OpenCores*. The source is freely downloadable under the GNU Lesser General Public License (LGPL).

Project website: `http://opencores.org/project,mod_sim_exp`

4.4 Pipelined Montgomery Multiplier

4.4.1 Montgomery Multiplication

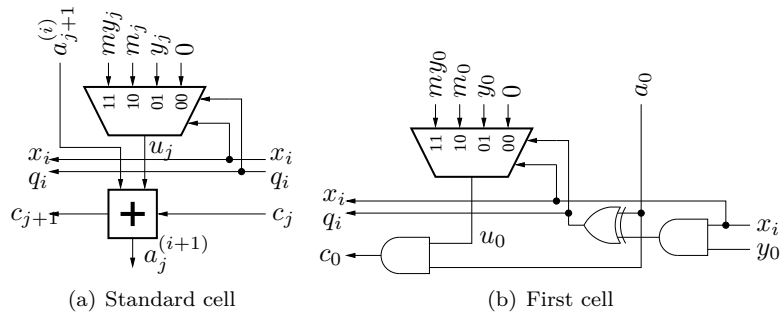
The kernel of our hardware design is the Montgomery multiplier. For our implementation we take the work of *Nedja and Mourelle* [Nd06] as a starting point. They designed a systolic array multiplier by modifying the Montgomery multiplication algorithm (Algorithm 2). The resulting algorithm easily translates into hardware. Every bit of the (intermediate) result a is driven by just a multiplexer (line 6) and an adder (line 12). The resulting circuit is shown in Fig. 4.3(a).

Algorithm 2 Systolic Montgomery Multiplication

Input: $x = (x_{n-1} \cdots x_0)_2$, $y = (y_{n-1} \cdots y_0)_2$, $m = (m_{n-1} \cdots m_0)_2$
Output: $x \cdot y \cdot R^{-1} \bmod m$ with $R = 2^n$

1: $my = (my_n \cdots my_0)_2 := m + y$
2: $a^{(0)} = (a^{(0)}_{n-1} \cdots a^{(0)}_0)_2 := 0$
3: **for** $i \leftarrow 0$ **to** $n-1$ **do**
4: $\quad q_i := a^{(i)}_0 \oplus x_i \cdot y_0$
5: $\quad$ **for** $j \leftarrow 0$ **to** n **do**
6: $\quad\quad$ **switch** x_i, q_i
7: $\quad\quad\quad 1, 1 :\ u_j := my_j$
8: $\quad\quad\quad 1, 0 :\ u_j := y_j$
9: $\quad\quad\quad 0, 1 :\ u_j := m_j$
10: $\quad\quad\quad 0, 0 :\ u_j := 0$
11: $\quad\quad c_j := a^{(i)}_{j+1} \cdot u_j + a^{(i)}_{j+1} \cdot c_{j-1} + u_j \cdot c_{j-1}$ ▷ $c_{-1} = 0$
12: $\quad\quad a^{(i+1)}_j := a^{(i)}_{j+1} \oplus u_j \oplus c_j$
13: **if** $a > m$ **then**
14: $\quad a := a - m$
15: **return** a

For every bit x_i of x, a new version $a^{(i)}$ of a is computed. A right shift operation – inherent to the Montgomery algorithm – ensures that a is never larger than $n + 2$ bit (final carry included). The other multiplexer selection line q_i also needs to be updated. This is done in the first cell (Fig. 4.3(b)).



(a) Standard cell (b) First cell

Figure 4.3: A systolic array cell computes 1 bit of the (intermediate) result.

In our design, systolic array cells are grouped into stages. This has two advantages. First of all it breaks up the long carry-chain in the adders, so we can achieve higher clock frequencies. Secondly, this approach allows for pipelining, further speeding up the design. A drawback is that, by increasing the number of stages, more flip flops will be used. The pipeline design and operation will be explained in the following paragraphs.

4.4.2 Pipeline Logic Blocks

The resulting pipeline consists of three main computing blocks (shown in Figure 4.4). The first cell logic computes (for each new bit x_i of x) the first carry-in for all the adders as well as the mux selection input q. The last cell logic computes the final carry (of the two's complement addition), which is used to select either the result or the reduced result. The systolic array stages (k in total) perform the main computation.

Note that the critical timing path is clearly the path to the final carry bit (line 11 of Algorithm 1). By grouping the systolic array cells in k stages this carry-chain is broken. To that end some registers are provided at the end of each stage. These registers store bits x_i, q_i and the carry bits that go to the next stage. Each stage has the same length s where $s = n/k$.

The `start` signal of each stage serves as an enable line to store the intermediate result and bits. The stage's `done` signal is just a delayed version of `start`. In this case the time required for a stage to complete one step is 1 clock cycle. The `done` signal of a stage acts as `start` signal for the next stage. The resulting structure allows for pipelining.

Because of the right-shift operation (lines 11, 12) the most significant bit of the previous result `a_msb` is provided by the next stage, so a stage can only start the

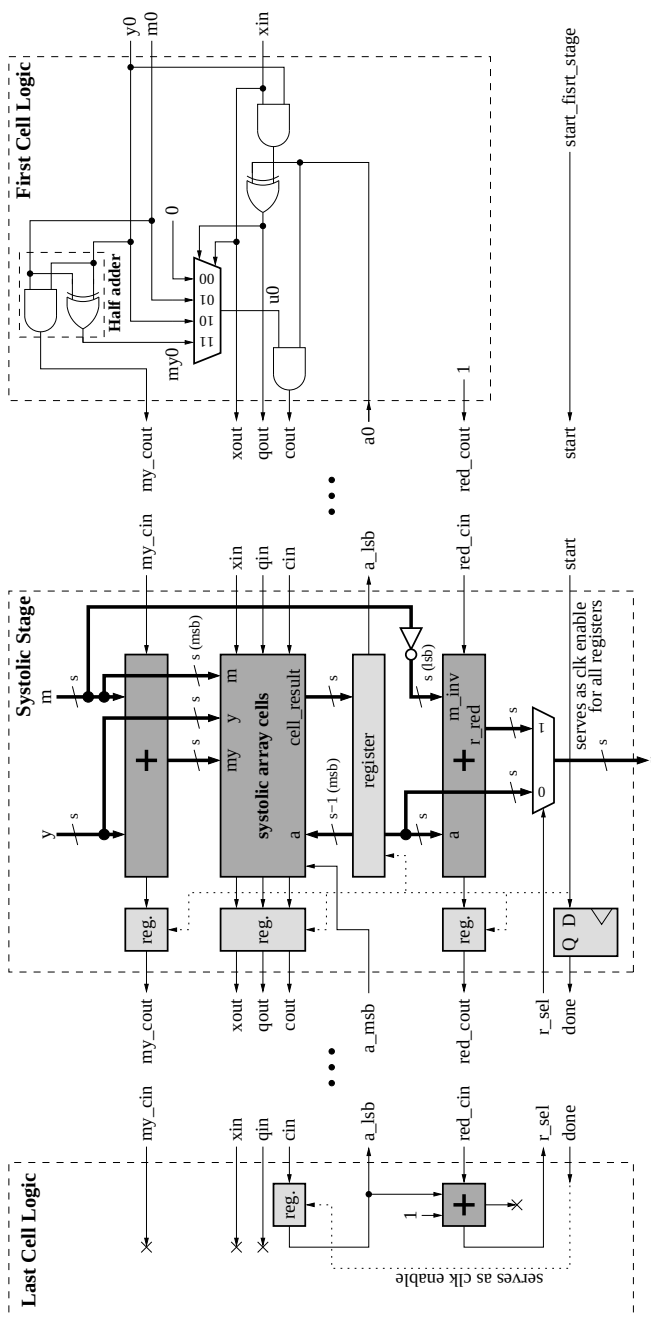


Figure 4.4: Pipeline logic blocks: k stages (width $s = \frac{n}{k}$) perform the main computation. The first cell logic computes (for each new bit x_i of x) the first carry-in for all the adders as well as the mux selection input q. The last cell logic computes the final carry (of the two's complement addition), which is used to select either the result or the reduced result.

next step once the next stage has finished the previous step and `a_msb` is valid.

In Algorithm 2 the value $my = m + y$ is required; two approaches are possible. One possibility is to let the controlling software send this value over the bus whenever it is needed. However, this value changes for almost each multiplication when doing a simultaneous exponentiation, which would lead to a lot of bus traffic and possibly slow down operation. In our opinion, this is not preferable.

Therefor we use a hardware adder that computes the required my value before the pipeline starts. An extra advantage is that we don't need to provide storage for my. We should note that the length of the carry chain in this adder is no issue here because the required part of the sum will be available before the corresponding stage starts its first step.

The final subtraction (Algorithm 2, line 14) is performed by converting the modulus to its two's complement representation and using a standard adder to compute the difference. The carry-out computed by the last cell logic serves to select the desired result.

4.4.3 Pipeline Operation

Fig. 4.5 shows the operation of the pipeline. A stage can only compute a step every 2 τ_s, where τ_s is the time it takes a stage to actually complete a step. In this case τ_s is 1 clock cycle, but the index $_s$ illustrates that this is influenced by the length of s.

Based on Stallings' notation on pipeline run time [Sta06], we can state that for one multiplication, the total computation time $T_{k,n}^{(\times)}$ for an n-bit operand with a k-stage pipeline is given by (4.2). Since there are no conditional paths in the pipeline design,[3] $T_{k,n}^{(\times)}$ is only dependent on n and k.

$$T_{k,n}^{(\times)} = [k + 2(n-1)]\,\tau_s \ . \tag{4.2}$$

In the next chapter we will examine the effect of the stage length on τ_s and what speedup can be achieved by the pipelining.

4.4.4 Complete Multiplier

The complete multiplier design is shown in Fig. 4.6.

[3] at least not any that have an influence on the run time...

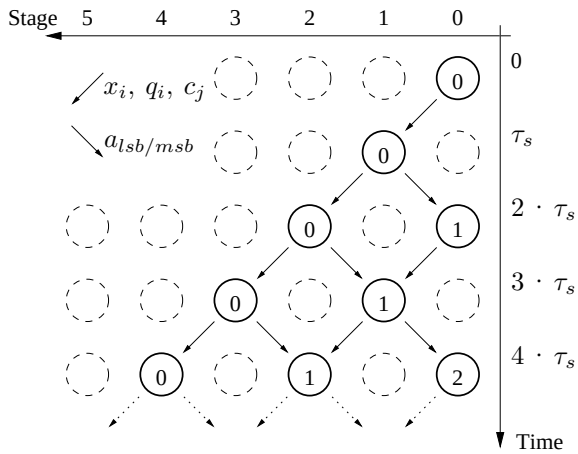


Figure 4.5: Pipeline operation: each circle represents a stage. The number inside the circle represents loop counter i (Algorithm 1, line 3) indicating the step that is executed by the stage at that time. Inactive stages have dotted-line contours.

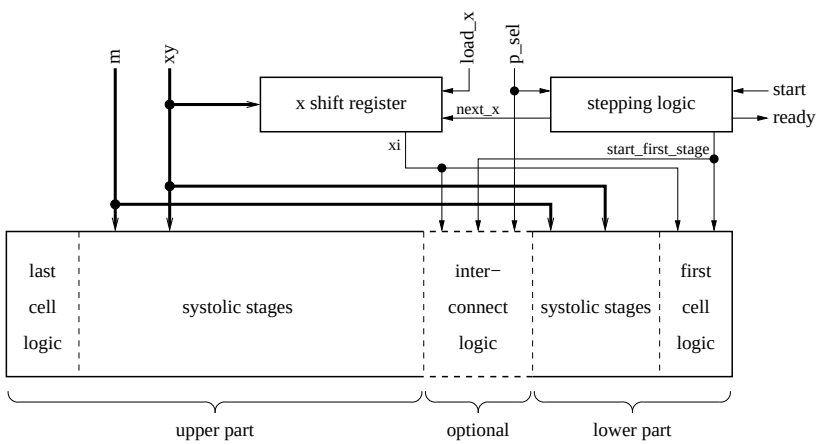


Figure 4.6: Complete multiplier design.

Prior to a new multiplication, the x operand is put on the `xy` line and clocked into the x shift register using the `load_x` line. Note that this is done in only one clock cycle.

The stepping logic makes sure that the first stage in the line starts at the right moment. It also signals the x shift register to provide a new bit of the x operand. Finally, it keeps track of the status of the multiplier and drives the `ready` line when the last stage finishes the last step. The stepping logic mainly consists of some counters.

As mentioned before, the design supports different operand lengths at run time.[4] This can be achieved by splitting the pipeline into two parts.[5] Fig. 4.6 shows the resulting structure with the upper and lower part and the optional interconnect logic in between. One can either choose to use the upper part, the lower part or the total pipeline using the line `p_sel`. The pipeline interconnect logic makes sure the correct stages are used. It is however not possible to use both the upper and lower part at the same time (for a different calculation).

Two more detailed example configurations of the systolic pipeline can be found in Appendix A.

4.4.5 Resistance Against Side-Channel Analysis

It should be noted that this design has not been tested for its resistance against side-channel attacks. In [Wal08, SST04] the frequency of occurrence of the final subtraction (Algorithm 2, line 14) is used to gain knowledge of the inputs. In this design, however, the final subtraction is always carried out and there is no difference in timing between the two cases. Still the resistance against any type of side channel attack has not been investigated.

Also the occurrence of zeros in corresponding bits of the exponents will have an influence on the execution time of the exponentiation. This means precautions should be taken to guarantee a constant timing, e.g., adding dummy multiplications or padding the exponents with preceding zero-bits. As a consequence, this would mean a drop in performance because the constant timing can only be achieved if the worst-case timing (i.e., all bits of all exponents are 1) is taken. In the current version of this design however, no such measures are taken.

This unknown side-channel resistance limits the current range of practical applications i.e., all applications that require exponentiations on secret information can be unsafe if this hardware accelerator is used. An example of such an application

[4] The choice of which operand lengths are supported, however, needs to be made before synthesis.

[5] Preferably each with a different length; for instance in our practical setup $n_l = 512$ and $n_h = 1024$, so $n_{tot} = 1536$.

is the building of a credential proof (see Chapter 8). A credential verification, on the other hand, is safe, provided that there's no harm in an attacker learning revealed attributes.

4.5 IP Core Software Interface

As seen in the previous sections, the IP core is designed to be part of a larger embedded system. The hardware is connected with a bus and IRQ line to an embedded processor. The core's internal memory and control registers are mapped in the controlling software's (virtual) memory. To ensure portability to other platforms we assume that the embedded CPU will be running Linux and that the hardware will be accessible as a UIO device (see Section 2.2.3). This also simplifies the design of the driver software.

All the source code for working with the hardware IP core is bundled in a library called `libmme1536`. This code consists of low-level driver functions and a high-level API. Users only require this API to work with the hardware accelerator. The library relies on the `UIO driver model` [15] and `GMP` [38]. In the following paragraphs we will highlight the most important functionality of the API.

Open-source driver library

Like the VHDL code of the hardware design, the driver source is also available online. The code is is hosted on *Google Code*. The source is freely downloadable under the GNU Lesser General Public License (LGPL).

Project website: `https://code.google.com/p/libmme`

4.5.1 Low-Level Driver and Hardware Control API

Memory organization. As the IP core is developed as a memory-mapped peripheral, all operations with the core involve reading from and writing to certain memory locations. The memory organization of the IP core is shown in Fig. 4.7. One can see that the total 28 Kbyte memory space consists of 7 blocks of each 4 Kbyte.

The control registers are accessed with the UIO driver. Although there are only a 32-bit control register and an interrupt control register, the UIO kernel driver expects that at least 4 Kbyte is mapped. This is because the kernel address translation table of the embedded OS works with 4 Kbyte memory pages. The great advantage of the UIO kernel driver, however, is that it catches the interrupts

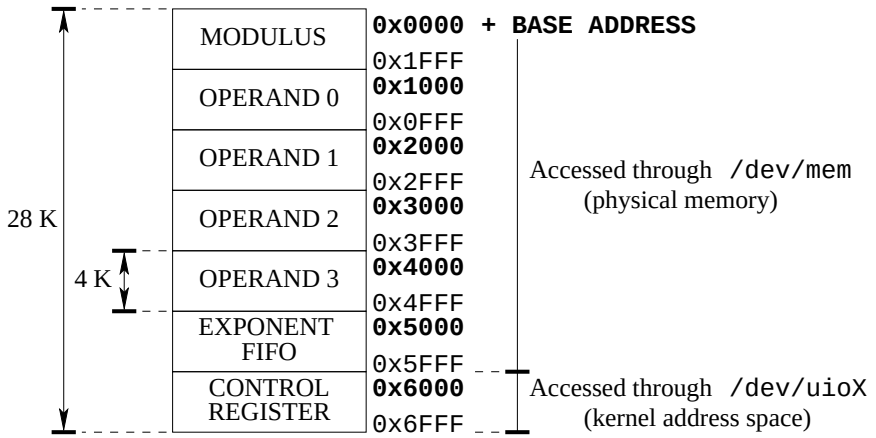


Figure 4.7: IP core memory organization.

generated by the hardware. By reading from the UIO device, the software can determine if there were any interrupts and how many.

All other memory locations are directly mapped. Four (4) Kbyte is provided for the modulus and each operand. When the operand length is smaller than the reserved memory, only the lowest memory locations are used (see example). Since the exponents are stored in a FIFO, only address `0x5000` is used.

Example

If operands of 1536 bits are being used, they are treated in software as a 32-bit array with $\frac{1536}{32} = 48$ elements each. The modulus is then stored in hardware in an address range `0x0000` to `0x0030` (offset to the `BASE_ADDRESS`), `OPERAND 0` in range `0x1000` to `0x1030` and so on.

Keep in mind that all other addresses, e.g., `0x0031` to `0x0FFF` are in fact not addressable in hardware so the allocated space of 28 Kbyte does not reflect the actual available hardware memory of the IP core.

Operation control. Running a computation can be done by calling either `MME1536_StartSingle()`, which starts a single multiplication, or `MME1536_StartAuto()`, which starts the main square-and-multiply loop. The function `MME1536_WaitUntilReady()` will wait until an interrupt is received, signalling that the computation has finished. A time-out value can be specified to cope with missed interrupts or other blocking errors.

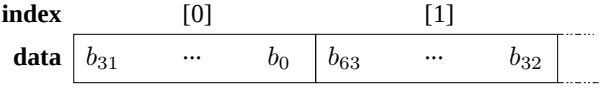


Figure 4.8: IP core data alignment requirement.

Initializing the hardware for use on the embedded platform can easily be done with `MME1536_Initialize()`. A user only needs to provide the correct UIO device. As a result the data and control memory pointers will be initialized as well as the UIO interrupt. With `MME1536_Clean()` the reserved memory space can be freed.

Reading and writing data. To write data to a certain memory location in the IP core, the user can use the `MME1536_SetOperand()`. Analogously, the function `MME1536_GetOperand()` can be used for reading data. The only important thing is the alignment of the data provided to the functions. This alignment is shown in Fig. 4.8. Both functions take care of the addressing, and if only a part of the pipeline is used, they will access the corresponding memory locations.

Similarly, for writing the exponent, the user can use `MME1536_SetExponent()`. The same data alignment applies for the exponents. When only one exponent is provided, the software will automatically write 0-bits in the FIFO entries. The only restriction for the exponents is that they need to be at least 32 bits wide.

In practice, a user only requires the functions `MME1536_Initialize()` and `MME1536_Clean()` and the arithmetic API to use the IP core. But the functions described above can (after the core has been initialized) be used to test the operation of the hardware e.g., a start/stop or read/write test.

4.5.2 Arithmetic API

To easily use the hardware accelerator for what it is designed for, three functions are provided:

- `MME1536_Multiply()`: perform a modular multiplication.[6]
- `MME1536_MME()`: perform a dual-base exponentiation. This performs the precomputations and stores the results in the designated memory location. Then the exponents are pushed in the FIFO and the automatic operation of the main loop is started. After the main loop completes, the postcomputation is executed and the result is read back.

[6]This is an actual modular multiplication and not a Montgomery multiplication.

- `MME1536_Exp()`: perform a single-base exponentiation. This is basically the same as the dual-base exponentiation but with one of the exponents set to zero.

To reduce redundant operations, there is also a function `MME1536_UpdateModulus()`. This will write a modulus to the hardware and compute R^2 in software. These two operations need to be done only once, as long as the modulus does not change.

4.6 Ongoing Development

As this is an open-source design, the source code is being maintained and has been subject to changes. The basic functionality and design concept, however, have been kept as described in the previous sections. Below are the most important changes that have been made to further increase performance, functionality and flexibility:

- *Generic RAM description.* To further increase the portability to platforms others than Xilinx FPGAs, the design of the RAM has been changed to a more generic description. The current RAM design is now synthesizable on both Xilinx and Altera FPGAs and effectively uses the available dual-port RAM. For devices that don't support asymmetric dual-port RAM, we have provided an option that uses an alternative RAM design that mimics an asymmetric RAM by using symmetric RAM and additional FFs. Of course this design uses more resources.
- *AXI(lite) bus.* Support has been added for the AXi4(lite) bus. Users can now choose between AXI or PLB.
- *Internal core clock.* In the original design, the complete IP core operated at the same frequency as the bus. For instance on Spartan devices this could result in a performance loss, where MicroBlaze systems often operate at frequencies of 66 or 75 MHz. Because the multiplier can work at higher frequencies, we added a separate input for the multiplier clock. The RAM operates as the main buffer between the two clock domains.
- *Tested on Zynq.* The IP core has been tested on a Xilinx Zynq device. The core was connected with the AXI bus to the Cortex A9 processor, which ran embedded Linux. Both the core as the developed driver library proved to be working on this platform as well.

4.7 Conclusions and Future Work

In this chapter we have presented the design of a flexible VHDL design for accelerating modular exponentiations in hardware. Both (simultaneous) dual-base and single-base exponentiations are supported as well as single multiplications. The IP core is designed for implementation in an embedded processor system on configurable hardware and requires controlling software to be operated properly.

Flexibility is available in terms of configurable operand length and variable exponent length. Furthermore, the length of the pipeline stages can be changed to cater to the user's needs. How this affects resources and clock frequency will be investigated in Chapter 5.

The complete design is available as open-source hardware and is maintained as such. The generic VHDL code does not instantiate any device-specific primitives e.g., multipliers, RAM, making it suitable for use across different platforms. Future updates could include other bus interfaces (e.g., Whishbone), side-channel resistance measures or support for different kinds of adders (e.g., carry-select to find a better trade-off between resources and speed). Also a JTAG interface [19] or some kind of self-test ability could be a useful addition.

In the following two chapters we will evaluate the performance of this design in terms of speed and resource usage. In Chapter 5 we verify the multiplier operation and the influence of different pipeline configurations on speed and resources. Chapter 6 then looks at the complete design in a practical setup and which speedup can be gained by deploying simultaneous exponentiations.

Chapter 5

Multiplier Performance

5.1 Introduction

In this section we will cover the performance of the multiplier of the developed IP core (described in Chapter 4) in terms of resource usage and maximum clock frequency. Those data come as a result of the synthesis process. Because the multiplier is the main part of the design, this will give as a good (first) idea about the performance of the complete IP core.

The synthesis process analyses the VHDL hardware description and turns it into an implementation consisting of logic gates available in the hardware. The main features of the synthesis process are presented in Table 5.1. The synthesis process has been set to optimize the design for area with a high effort. The reason for this is that, by optimizing for area, the synthesized circuit will be closely linked to what we designed e.g., no extra gates to reduce critical paths. This will give us a better understanding of how the variation of certain design parameters will influence the operational frequency or FPGA resource usage.

We will compare the synthesized multiplier with several designs found in the literature. As a last step, the multiplier timing will be verified in a practical setup.

Table 5.1: Important features for the synthesis process.

Feature	**Value**	**Remarks**
Device	XC6VLX240T-1FFG1156	Xilinx FPGA
Synthesis tool	XST	
Goal	area	
Effort	high	
Technology	40 nm	

5.2 Timing

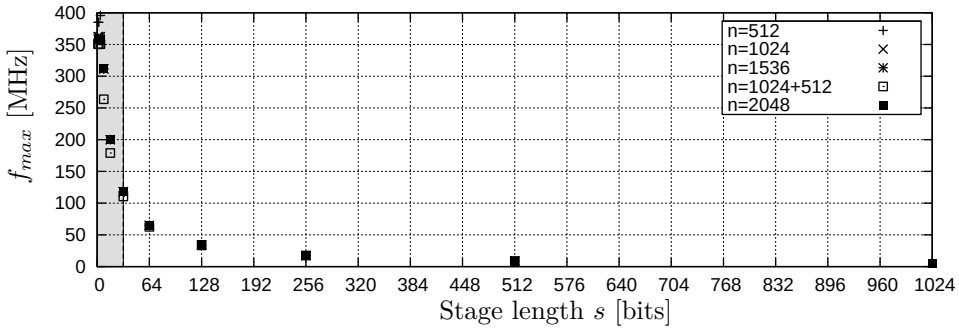
5.2.1 Operational Frequency

Since the maximum frequency is highly influenced by the latency in the critical path, we can expect to achieve higher frequencies for shorter stage lengths. We obtained this figure from the static timing analysis during the synthesis step in the design process (see Fig. 5.1).

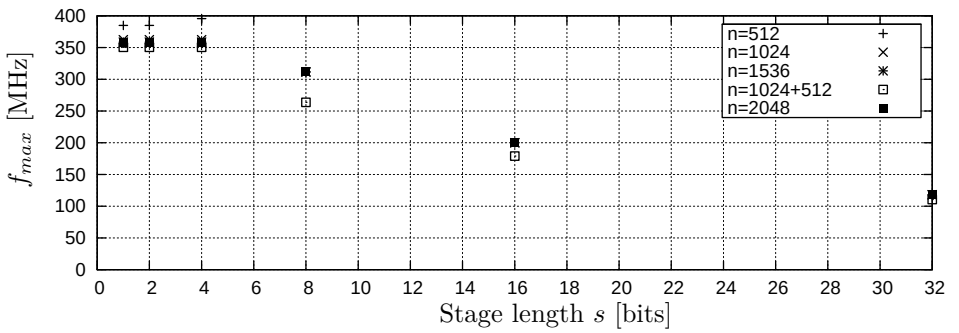
We can see that f_{max} indeed increases when the stage length decreases. For $s \leq 4$, we see that f_{max} saturates to a maximum (i.e., 350 MHz). A likely explanation is that this depends on the slice architecture which, for the Virtex 6, contains 4 LUT-FF pairs (see outline).

It is also notable that the maximum frequency is almost independent of n. We can see, however, that f_{max} is slightly lower for a design with a split pipeline ($n = 1024 + 512$ versus $n = 1536$). This is due to longer critical paths in the interconnect logic.

What is striking is that for stages of 512 bits and more, the operational frequency is less than 10 MHz (Fig. 5.1(a)). This is less than 5% of the maximum attainable frequency (for this design on this FPGA).



(a)



(b) Detail

Figure 5.1: Maximum clock frequency of the pipelined multiplier for different values of s and n.

Influence of the slice architecture

As we have observed, the clock frequency reaches a maximum when the stage length is equal to the number of LUT-FF pairs in a slice, designated v (for our FPGA this is $v = 4$). Or in other words, the latency of a stage (τ_s) is determined by v.
A slice can be used to compute 4 bits of the (intermediate) result in τ_s. If we would increase the stage length to 6, it would not take $1.5\tau_s$ but $2\tau_s$, because now 2 slices are required. This means that τ_s (and hence f) shows some granularity, determined by v: $\tau_s = \alpha\lceil\frac{s}{v}\rceil$. Because from $s > 4$, s progresses in multiples of 4, we cannot observe this behavior.
Similarly, Blum observed in his thesis [Blu99] that matching the length of the processing elements to the slice architecture yields the highest speed.

5.2.2 Execution Time

Now that the effect of the pipeline stage length on the operational frequency is known, we can see what this does to the multiplication execution time. We recall equation (4.2), which gives the time required for one multiplication:

$$T_{k,n}^{(\times)} = [k + 2(n-1)]\,\tau_s \ .$$

Or as a function of stage length:

$$T_{s,n}^{(\times)} = \left[\frac{n}{s} + 2(n-1)\right]\tau_s \ . \tag{5.1}$$

We defined τ_s as the time required to compute one stage. In our design τ_s is one clock cycle, or $\tau_s = \frac{1}{f}$. If we assume the multiplier operates at its maximum frequency, we can set out $T_{k,n}^{(\times)}$ as a function of the number of stages. For clarity, however, we set out $T_{s,n}^{(\times)}$ (Fig. 5.2).

One might expect that $T_{s,n}^{(\times)}$ is proportional to $\frac{1}{s}$. However, τ_s is proportional to s (see outline). After substitution we get:

$$T_{s,n}^{(\times)} = \left[\frac{n}{s} + 2(n-1)\right]\alpha\left\lceil\frac{s}{v}\right\rceil \ .$$

Or in case s is a multiple of v:

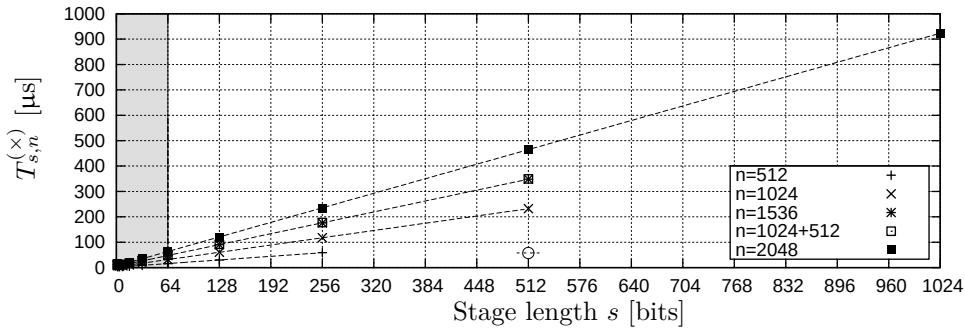
$$T_{s,n}^{(\times)} = \frac{\alpha n}{v} + \frac{2\alpha(n-1)}{v}s \ . \tag{5.2}$$

And when $s \leq v$ (or $\lceil\frac{s}{v}\rceil = 1$):

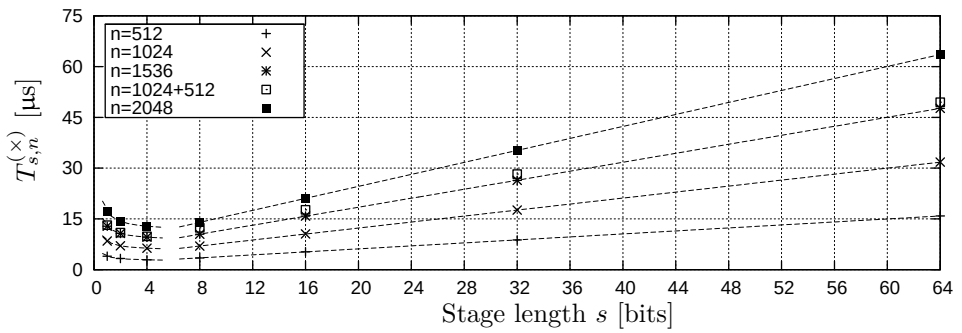
$$T_{s,n}^{(\times)} = \alpha n\frac{1}{s} + 2\alpha(n-1) \ . \tag{5.3}$$

As we can see in Eqn. (5.2), when $s > v$, the execution time is proportional with s. For $s \leq v$ there is a $\frac{1}{s}$-behavior (Eqn. (5.3)). This is set out in Fig. 5.2(b). Because the maximum frequency is slightly lower for the split pipeline, the run time will hence be slightly higher.

In any case, the minimal execution speed is achieved when $s = v$. Then $T_{s,n}^{(\times)} = \alpha\left[\frac{n}{v} + 2(n-1)\right]$, where α is dependent on the technology of the FPGA and both α and v are dependent on the slice architecture. This is also valid when the multiplier operates at a certain fraction of the maximum frequency.



(a)



(b) Detail

Figure 5.2: Multiplication execution time of the pipelined multiplier operating at the maximum frequency for different values of s and n.

5.2.3 Speedup

If no pipelining were to be used, the multiplier would complete one step in one clock period τ_n and one multiplication in $n \cdot \tau_n$. This time is also shown in Fig. 5.2(a) in the case of $n = 512$ (the ∘-mark). Now that the execution speed for different stage lengths is known, we can set out the speedup factor as a function of the stage length.

Stallings [Sta06] defines the speedup factor S_k as in Eqn. (5.4). Here, $T_{1,n}^{(\times)}$ is the run time if no pipelining is used.

$$S_k = \frac{T_{1,n}^{(\times)}}{T_{k,n}^{(\times)}} = \frac{n\tau_n}{[k + 2(n-1)]\,\tau_s} \quad . \tag{5.4}$$

Fig. 5.3 shows the speedup for $n = 512$ and $n = 1024$ as a function of s. It is clear

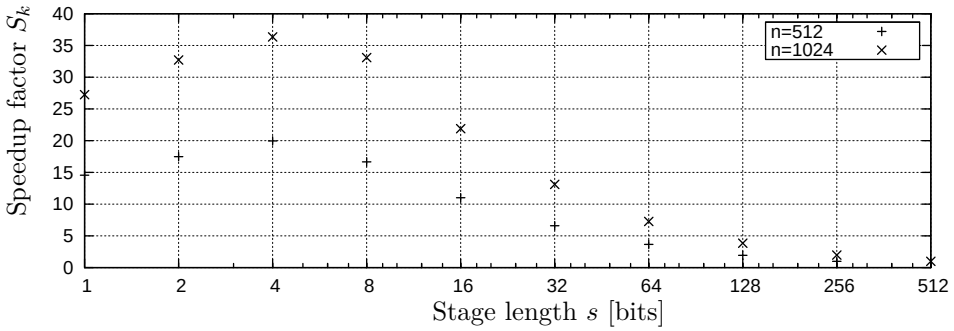


Figure 5.3: Speedup factor in function of the stage length for $n = 512$ and $n = 1024$.

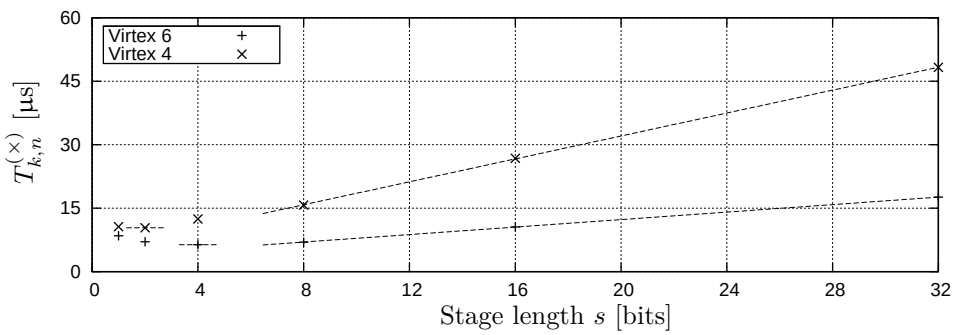


Figure 5.4: Execution time in function of s for $n = 1024$ on Virtex 6 and Virtex 4.

that pipelining results in a practical speedup which is proportional with the length of the operands. Of course, the largest speedup is achieved for $s = v = 4$.

5.2.4 Timing on Different FPGAs

To check if working with a different type of FPGA yields comparable results, we did the synthesis for the Virtex 4. The two important differences between the Virtex 6 and Virtex 4 are the technology and the slice architecture. The Virtex 4 series is fabricated with 65 nm technology resulting in slower clock speeds and has two LUT-FF pairs per slice instead of four. We have set out the execution times for $n = 1024$ in Fig. 5.4, for the Virtex 4 as well as the Virtex 6.

It is clear that for $s \leq 8$ the execution time has the same linear behavior. Due to the lower clock frequencies attainable on the Virtex 4, however, execution times are

longer compare to the Virtex 6. The minimum execution time, however, which lies for both devices in the non-linear part, is situated at $s = 2$ for the Virtex 4. This further corroborates our assumption that the number of LUT-FF pairs determines the stage length to get a minimum execution time on FPGA devices.

5.3 Resource Usage

5.3.1 Registers

Based on our design we can predict the number of registers required for the multiplier. Both the x-shift register and the (intermediate) result register are n-bit registers. For each stage a total of 6 registers is required: 3 for carry bits[1] and 1 for the `start/done` signal, `qi` and `xi` signals. To keep track of the operation progress two counters are being used: one for the number of steps and one for the stages. That results in Eqn. (5.5).

$$\#\ \text{FFs} = C + 2n + 6k + \lceil \log_2(n) \rceil + \lceil \log_2(k) \rceil \ . \tag{5.5}$$

The term C is a constant number of flip flops, independent of k and n. We will determine C experimentally. We synthesized the IP core with different values of n and k. The results are set out in Fig. 5.5. We also plotted Eqn. (5.5). With regression, C was determined as 5.

It is clear that with Eqn. (5.5) the number of registers in the synthesized design can be determined accurately. Also note that there is no difference between the split and non-split pipeline. Furthermore, synthesis of this design on a Virtex 4 results in exactly the same number of flip flops being used (see Fig. 5.6).

5.3.2 Combinatorial Logic

The number of look-up tables used is also a result of the synthesis process. It gives an idea about how many gates are used in the design.[2]

We have set out the number of LUTs in the multiplier for different values of n and s in Fig. 5.7. In Fig. 5.8 one can see the number of look-up tables in a 1024-bit multiplier for different values of k on Virtex 6 and Virtex 4.

It is clear that the number of look-up tables is only dependent on n and the type of LUT; i.e., the Virtex 4 has 4-input LUTs while the Virtex 4 houses 6-input LUTs.

[1] One carry bit in the stage itself and 2 in the adders (reduction adder and *my* adder).

[2] It is difficult to translate the number of LUTs into a number of GEs. For example, a 6-input LUT can be used as e.g., a single inverter but also as a 4-to-1 multiplexer etc.

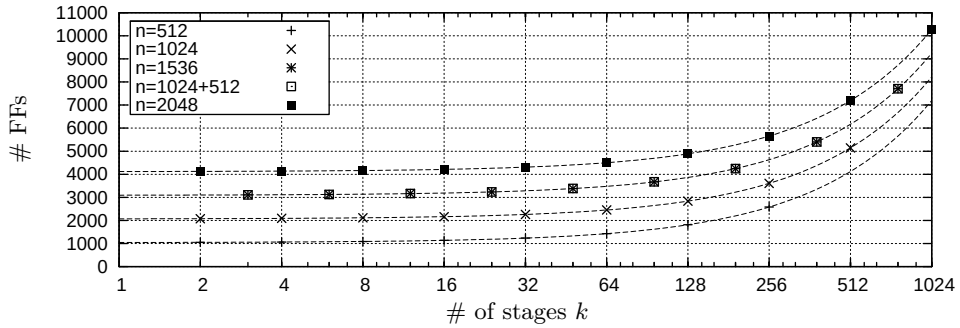


Figure 5.5: Number of flip flops in the multiplier for different values of n and k.

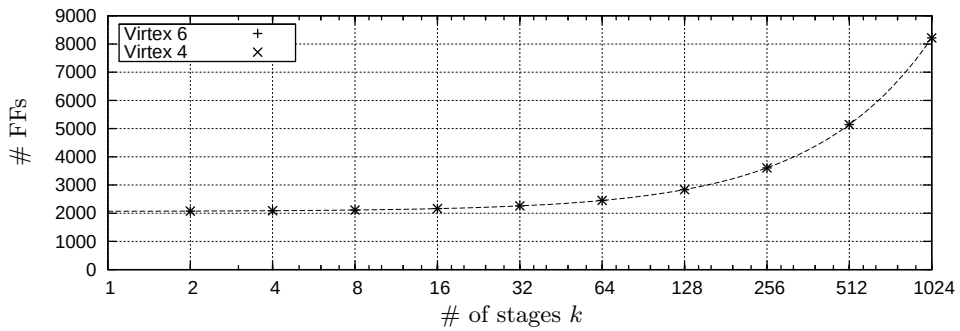


Figure 5.6: Number of flip flops in a 1024-bit multiplier for different values of k on Virtex 6 and Virtex 4.

Equation (5.6) can be used to estimate the number of LUTs in a multiplier with length n.

$$\#\ \text{LUTs} = \begin{cases} 8 \cdot n & \text{for 4-input LUTs} \\ 6 \cdot n & \text{for 6-input LUTs} \end{cases} \tag{5.6}$$

5.3.3 Resource Usage on an Altera FPGA

To verify if our design produces a comparable resource usage on FPGAs from another manufacturer, we have synthesized the multiplier on an Altera Stratix FPGA. The number of FFs as well as the number of LUTs match with the expected values; resp. Eqn. (5.5) and Eqn. (5.6).

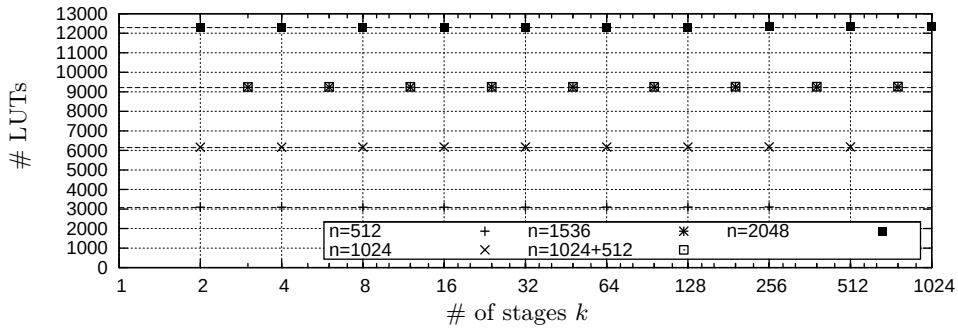


Figure 5.7: Number of look-up tables in the multiplier for different values of n and k.

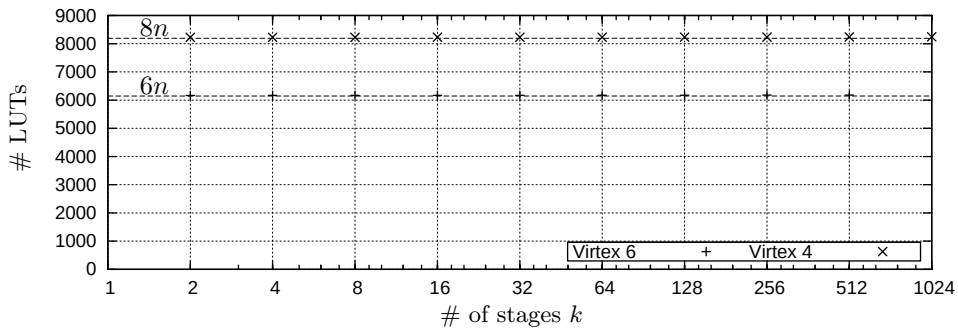


Figure 5.8: Number of look-up tables in a 1024-bit multiplier for different values of k on Virtex 6 and Virtex 4.

Also interesting, is that the Altera design tools give the number of logic gates that is being used. This allows us to state a first indication of the gate count this design would need when implemented on ASIC. This way we can also make a cautious comparison with other designs. The number of gates given by the Altera synthesis tool, is listed in Table 5.2.

Table 5.2: Required amount of gates for the multiplier (Altera synthesis).

Gate	**OR**	**XOR**	**AND**	**MUX**	**Total comb.**
amount	$3n$	$6n$	$6n$	$4n$	$19n$

If we take a gate equivalent (GE) of 1.5 for the combinational gates and a GE of 5 for a flip flop, then a multiplier with $n = 1024$ and $s = 16$ takes a total of about 32 kGE.[3] This is of course a rough indication and real gate counts can only be obtained by actually making an ASIC or full custom design. Keep in mind that for the complete IP core, also $5n$ RAM cells are required, which would add at least an extra 7.7 kGE.

5.4 Comparison with Existing Designs

In this section we compare our multiplier design with some designs found in literature. We have looked at the resource usage and time required to perform an exponentiation. This comparison is not an easy task as some designs use, for instance, the DSP slices or an FPGA with a different slice architecture. Some articles list only the number of slices or CLBs their design uses, while other designs are only synthesized for ASIC.

Nevertheless, we have tried to make an honest comparison. To that end, we have taken a commonly used configuration i.e., a 1024-bit multiplier with 16-bit processing elements, as benchmark. Also, most designs list the time of an exponentiation with the F_4 exponent so we have taken this as a reference.[4] However, for our design we only list the time required for the main loop.

Table 5.3 shows our comparison. Where necessary, we have listed the number of LUTs and FFs in the CLB/slice of the FPGA used in a particular design.

Our design has a comparable performance (same order of magnitude) in terms of resources. Faster designs will either use more (complex) logic, require more complex pre-computations, or only work under certain assumptions.

The main strength of our design is that it can be tuned to a desired operational frequency or exponentiation run time independent of the device, while maintaining an acceptable amount of logic. Moreover, changing the operand length or pipeline stage length does not require a complete redesign of hardware and software. This is especially interesting in the case of fast prototyping, where developers are more interested in the performance gain by using a hardware accelerator with certain specifications and what effect these specifications have on the overall performance of the application, rather than selecting a new IP core every time the specifications need to be changed.

[3] We take a GE as being the number of transistors in a 2-input NAND gate. For CMOS this is four transistors, which means that a 32 kGE corresponds with 128 k transistors.

[4] The Fermat prime $2^{16} + 1 = 65537$ is an exponent used in RSA, designated F_4.

Table 5.3: Comparison with existing work $n = 1024$. Exponentiation times are marked with $^{(e)}$, multiplication times with $^{(m)}$.

Description	**Ref.**	**Technology**	**LUTs**	**FFs**	**area**[a]	**Clock speed**[b]	**Time**
Pipelined systolic array Montgomery (16-bit stages)	This work	Xilinx Virtex 6 XC6VLX240T	6171 and 65 BRAMs (complete IP core)	2453	-	199.96	0.20 ms $^{(e)}$ 10.55 µs $^{(m)}$
Systolic array Montgomery	[BP99]	Xilinx XC4000	*2/CLB (LUT4)*	*2/CLB*	6633 CLB	45.66	0.22 ms$^{(e)}$
k-partition parallel	[NTR11]	90 nm cell	-	-	1.179 mm^2	87	2.98 µs $^{(m)}$
redundant radix-64k	[SKN08]	Xilinx Virtex-II Pro	*2/slice (LUT4)* and 64 18-bit multipliers and 29 BRAMs	*2/slice*	7886 slices	52.59	1.23 µs $^{(m)}$
Radix-4 Booth enc.	[ZW10]	Altera Stratix III	2836	2932	-	-	10.18 ms$^{(e)}$
Parallel radix-4	[PH08]	Xilinx Virtex II	8428	7543	-	248	5.4 ms$^{(e)}$
Authentec	[12]	IP core 130 nm benchmark	-	-	254 kGates	230	0.95 ms$^{(e)}$ 1024-bit exp.

[a] If no LUTs and FFs were specified. Can be a number of slices, CLBs or an actual area.
[b] [MHz]

Table 5.4: Test setup features.

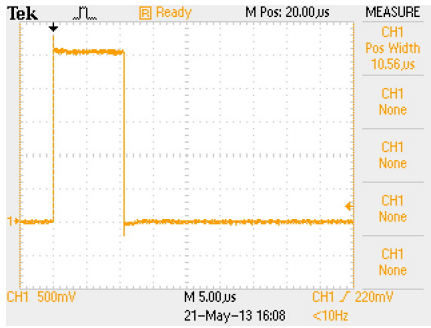
Feature	**Value**			**Remarks**
Device	XC6VLX240T-1FFG1156			Xilinx FPGA
CPU	MicroBlaze			MMU, barrel shifter, multiplier
OS	Linux			Petalinux with BusyBox shell
f_{clk}	100 MHz			both CPU and IP core
n	512	1024	1536	[bits] – split pipeline
k	32	64	96	for the supported operand lengths resp.
$T_{k,n}^{(\times)}$	10.54	21.10	31.66	[µs] – no OS overhead included

5.5 Practical Multiplication Timing

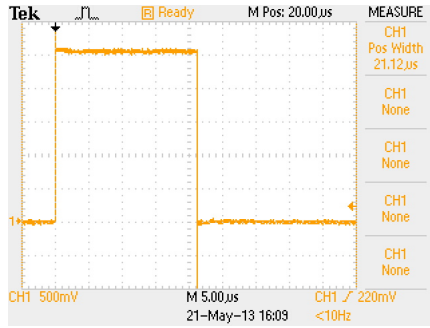
The developed IP core has been added to the embedded test platform described in Chapter 2. All practical results are obtained with the same configuration. The most important features of the test setup listed in Table 5.4.

To verify the run time of a single Montgomery multiplication in the IP core itself, we added an output to the IP core which is high during computations. By sampling this output with e.g., an oscilloscope we can measure the hardware run time. These measurements can be seen in Fig. 5.9(a), (b) and (c). Note that the timing corresponds with the expected values (Table 5.4). However, there is a constant difference of 0.02 µs, which corresponds with 2 clock cycles.

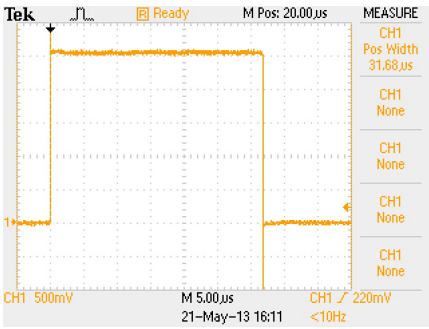
We also compared the hardware run times with the application run time. The latter includes writing the operands to the IP core memory, starting the multiplication, waiting for an interrupt and reading back the result. The average run times (over 40 measurements) are shown in Table 5.5. Fig. 5.9(d) shows a snapshot of the time between two consecutive multiplications. It is clear that bus communication and latency introduced by the operating system makes a software-controlled version of the square-and-multiply algorithm highly inefficient. This certainly justifies the design choice of implementing a slightly more complex control logic and exponent FIFO to enable automatic squaring and multiplying in a multi-base exponentiation hardware accelerator.



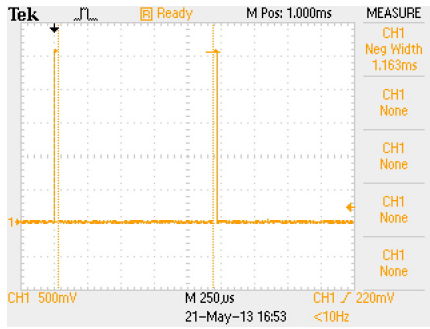
(a) $n = 512$



(b) $n = 1024$



(c) $n = 1536$



(d) Two subsequent multiplications.

Figure 5.9: Hardware run times for a Montgomery multiplication. The positive pulse width corresponds with the time that the IP core is performing computations.

Table 5.5: Montgomery multiplication application run time versus hardware run time.

n	**Expected**	**Measured**		
	Eqn. (4.2)	(hardware)	(application)	
512	10.54	10.56	1182.50	[µs]
1024	21.10	21.12	1400.50	[µs]
1536	31.66	31.68	1623.25	[µs]

5.6 Conclusions

In this chapter we have evaluated the performance of the multiplier used in our IP core design (Chapter 4) in terms of resources and timing.

As expected, shorter pipeline stage lengths result in a higher attainable frequency. When it comes to minimum execution times for multiplications, we can state that they are proportional to the stage length. However, the absolute value is determined by the technology and slice architecture of the device for which the design is synthesized.

In practice the multiplier operates as expected with a timing given by Eqn. (4.2). However, because of the overhead introduced by the OS and the bus communication, the actual multiplication timing as seen by an application is orders of magnitude higher. In the simultaneous exponentiation algorithm, getting data to the multiplier on time is ensured by the use of the IP core's operand RAM and control logic. This approach also omits the bus communication during the main loop of the algorithm.

For the resource usage, we have observed that the amount of combinatorial logic is only determined by n, where the number of register is dependent on n and the number of stages. To be able to assess the design's resource usage before synthesis, we provide the user with two expressions that state (for an FPGA) the required number of LUTs and FFs; Eqn. (5.6) resp. (5.5). The resource usage is of the same order of magnitude as other multiplier designs with same length for operands and processing elements.

For the complete IP core, only the control logic and operand memory needs to be added. For large n, the resource usage of the control logic is negligible. Because of the construction of the asymmetric memory, the current design is quite lavish in the use of Block RAM, e.g., 96 BRAMs are required to create the 1536-bit operand RAM. This is, however, a distorted picture as only a fraction of a BRAM cell is being used. A full custom design would only require $5n$ bits of RAM instead of the current 96×18 K.

Chapter 6

IP Core Performance

6.1 Introduction

The developed modular exponentiation accelerator is designed to be part of a larger embedded system. To verify the operation and to evaluate the operational performance of the hardware accelerator in such an embedded system, we add the core to our embedded test platform (Chapter 2). We use the developed software API to create several test applications.

First of all, we will compare the theoretical run times for multiplication and (dual-base) exponentiation with the practical run times achieved on the embedded platform (Sect. 6.2).

Secondly, we will evaluate the performance when the exponents differ in length. As the hardware is developed to work with exponents of the same length, we can expect length differences to have an influence on the run time. We will derive some general expressions for the run time of an l-base simultaneous exponentiation and the maximum attainable speedup compared to an l-base exponentiation using single exponentiations and multiplications (Sect. 6.3).

The conclusions from this evaluation are stated in Sect. 6.5.

All practical results are obtained with the same configuration listed in Table 5.4.

6.2 Simultaneous Exponentiation Run Times

Hardware run time. If we look at Algorithm 1, we see that a multiplication step (lines 5-8) is only carried out when at least one of the exponent bits is one ('1'). Consequently, it is not carried out when all of the exponent bits are zero ('0'). Looking at practical applications (Chapter 8), we can see that the exponents contain a great deal of randomness. That means that the chance of having a one or a zero is equally large i.e., $\frac{1}{2}$. For a dual-base exponentiation, that means that there is a chance of $\frac{3}{4}$ that at least one of the exponent bits will be one. So on average Algorithm 1 requires $\frac{7}{4}$ multiplications per exponent bit. The run time of a dual base exponentiation (main loop) is hence given by (6.1), where w is the length of the exponents.

$$T_{2,w}^{(\wedge)} = \frac{7}{4} \cdot w \cdot T_{k,n}^{(\times)} \quad . \tag{6.1}$$

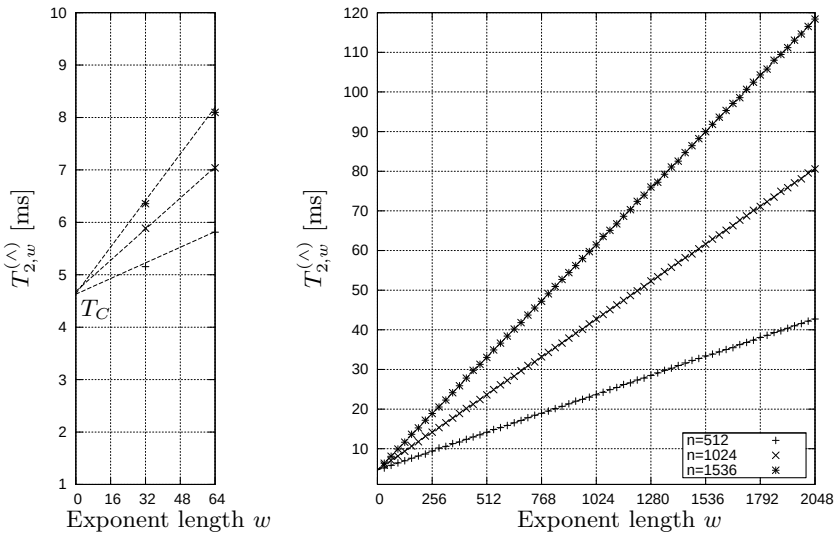
Application run time. To get the practical run time in an application, we need to add the time required for bus communication with the IP core, the latency introduced by the OS[1] and the time for the precomputation and postcomputation; both with their own bus communication and OS latency as seen in the previous paragraphs. We can assume that for a given operand length, this extra time is independent of w, so Eqn. (6.1) becomes:

$$T_{2,w}^{(\wedge)} = \frac{7}{4} \cdot w \cdot T_{k,n}^{(\times)} + T_C \quad . \tag{6.2}$$

To verify Eqn. (6.2) and to determine T_C we have performed a series of simultaneous exponentiations for different exponent lengths (Fig. 6.1). Every data point on the graph is the average of 20 measurements, with new random exponents for each measurement. The slope and intercept obtained from the least-squares linear regression are shown in Table 6.1. It is clear that $T_{2,w}$ follows the predicted linear function and that the slope corresponds with what we expected. Note that T_C is determined by the time required for bus communication (dependent on n) and the OS latency (independent of n). The latter is clearly dominant and hence we can state that T_C is independent of n.

To illustrate the operation of hardware during execution of the function `MME1536_MME()`, which performs a dual-base exponentiation, we can sample the hardware timing output of our IP core. This measurement is shown in Fig. 6.2. Note the spikes at the start (precomputation), the main loop and the spike at the end (postcomputation).

[1] The time between the interrupt signal and the interrupt being handled by the UIO driver and the application.



(a) detail (b)

Figure 6.1: Application run times of a modular simultaneous exponentiation for different values of w and n.

Table 6.1: Linear regression results for $T_{2,w}^{(\wedge)}$.

n	slope (expect.)	slope (regres.)	intercept	correlation coef.
512	0.0184	0.0186	4.629	0.9999743
1024	0.0369	0.0370	4.682	0.9999918
1536	0.0554	0.0555	4.639	0.9999930

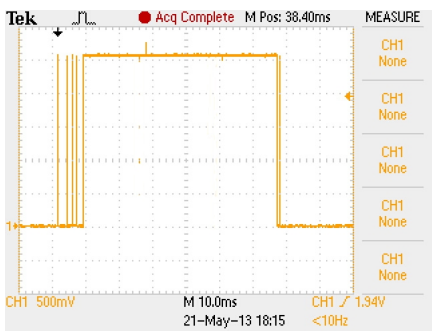


Figure 6.2: Hardware run time for `MME1536_MME()`.

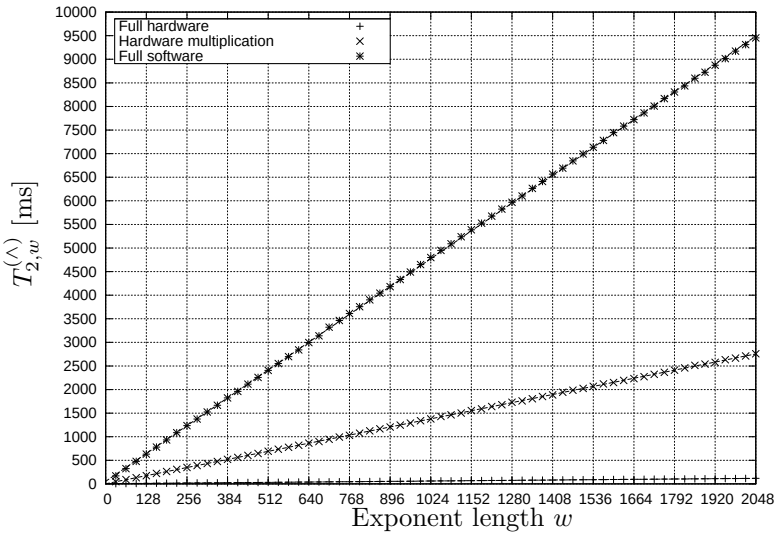


Figure 6.3: Comparison of three embedded implementations of a modular simultaneous exponentiation for different exponent lengths and $n = 1536$.

Comparison with alternative embedded implementations. To illustrate the value of our IP core in embedded applications, we compare the execution time of our full-hardware simultaneous exponentiation with two alternative implementations:

- An embedded software implementation that uses GMP [38] for the arithmetic, running completely on the MicroBlaze processor.
- An implementation where only the Montgomery multiplier is used and the rest of Algorithm 1 runs on the CPU.

The results are shown in Fig. 6.3. It is clear that the full-hardware exponentiation outperforms an embedded software implementation. And although it is faster than the embedded software implementation, the software-controlled multiplier approach is also an order of magnitude slower than a full-hardware exponentiation, as was to be expected. Moreover, the full-hardware approach requires only a small amount of extra resources.

6.3 Influence of the Exponent Length Difference

The previous results only illustrate the value of the design compared to approaches that are destined to be slower. However, as already mentioned in previous chapters,

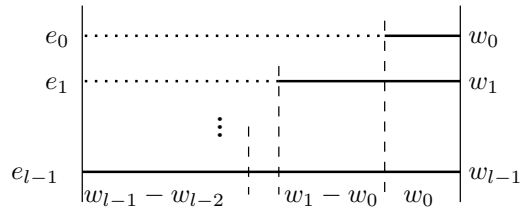


Figure 6.4: Arrangement of the exponents in a simultaneous exponentiation.

a trade-off between speed and area/cost needs to be made for every application separately. If applications only require a few exponentiations, it might not be worth it to implement simultaneous exponentiation hardware (due to the increased memory requirements). Moreover, previous results have been obtained with equal lengths of the exponents e_0 and e_1. However, as will be shown later (Chapter 9), exponents can have different lengths in a practical application.

To be able to make educated decisions e.g., whether to use a multi-base or a single-base exponentiation accelerator in a certain application, we require a general expression which gives the time to compute a simultaneous multi-base exponentiation in function of the exponent lengths and the gain we get in comparison to a single-base implementation.

6.3.1 A General Expression for Simultaneous Exponentiation Timing

Let l be the number of factors in the multi-base exponentiation Eqn. (6.3):

$$\prod_{i=0}^{l-1} g_i^{e_i} \bmod m \ . \tag{6.3}$$

The length of an exponent e_i is designated w_i. We also assume that the exponents are ordered in such a way that:

$$w_0 \leq w_1 \leq ... \leq w_{l-1} \ .$$

Because the exponentiation hardware requires exponents that all have the same length, it is required that exponents are padded with preceding zeros to match the length of the longest exponent. This is also represented graphically in Fig. 6.4.

To generalize the timing for a dual-base exponentiation (6.1) to a multi-base exponentiation with l exponents, we can state that the chance that for a certain

position all bits are zero is $\frac{1}{2^l} = P_0$. So the average time for a simultaneous exponentiation where all exponents have the same length w is given by:

$$T_{l,w}^{(\wedge)} = [1 + (1 - P_0)] \cdot w \cdot T_{k,n}^{(\times)} \ .$$

$$T_{l,w}^{(\wedge)} = \left[1 + \left(1 - \frac{1}{2^l}\right)\right] \cdot w \cdot T_{k,n}^{(\times)} \ .$$

$$T_{l,w}^{(\wedge)} = \frac{2^{l+1} - 1}{2^l} \cdot w \cdot T_{k,n}^{(\times)} \ . \tag{6.4}$$

Or generally for different exponent lengths:

$$T_{l,\{w_0..w_{l-1}\}}^{(\wedge)} = \sum_{i=0}^{l-1} \frac{2^{l-i+1} - 1}{2^{l-i}} (w_i - w_{i-1}) T_{k,n}^{(\times)} \ . \tag{6.5}$$

Where we define $w_{-1} = 0$. If all exponents have the same length, Eqn. (6.5) can be simplified, resulting in Eqn. (6.4).

It must be noted that this is an average timing, achieved when random exponents are being used. If a constant timing must be maintained (e.g., to counter timing analysis attacks) the worst-case timing must be taken. It is clear to see that this will be $2 \cdot w \cdot T_{k,n}^{(\times)}$. However, this is independent of the number of exponents.

6.3.2 An Expression for the Speedup when Using Simultaneous Exponentiations Instead of Single-Base Exponentiations

We start by assuming that the time for a multiplication is negligible with respect to the time for a single exponentiation. Even with the overhead (i.e., bus traffic and OS latency) introduced in a practical setup, this is a justifiable assumption when the exponents are large enough. In that case $T^{(\wedge)}$ will be orders of magnitude larger than $T^{(\times)}$.

With that assumption we can state that the time to compute a multi-base exponentiation with the use of single-base exponentiations is:

$$\sum_{i=0}^{l-1} T_{1,w_i}^{(\wedge)} \ . \tag{6.6}$$

We then define the speedup factor $S_{l,\{w_0..w_{l-1}\}}$ as:

$$S_{l,\{w_0..w_{l-1}\}} = \frac{\sum_{l=0}^{l-1} T^{(\wedge)}_{1,w_i}}{T^{(\wedge)}_{l,\{w_0..w_{l-1}\}}} \quad . \tag{6.7}$$

The index l designates the number of simultaneous exponentiations and $\{w_0..w_{l-1}\}$ designate the respective lengths of the exponents.

Determining the conditions for maximum speedup in case of two exponents with different length. We will derive the conditions for a maximum speedup in the case of $l = 2$. For $l > 2$, a similar approach can be followed. Equation (6.7) in the case of $l = 2$ becomes:

$$S_{2,\{w_0,w_1\}} = \frac{\frac{3}{2}w_0 T^{(\times)}_{k,n} + \frac{3}{2}w_1 T^{(\times)}_{k,n}}{\frac{7}{4}w_0 T^{(\times)}_{k,n} + \frac{3}{2}(w_1 - w_0)T^{(\times)}_{k,n}} \quad . \tag{6.8}$$

It is clear that $T^{(\times)}_{k,n}$ has no influence on the speedup, which could be expected. As stated before, we require that $w_0 \leq w_1$. To be able to see what happens with the speedup when w_0 becomes larger than w_1, however, we reformulate (6.8) in both cases. After simplification we then get:

$$S_{2,\{w_0,w_1\}} = \begin{cases} \frac{6w_0+6w_1}{w_0+6w_1} & \text{when } w_0 \leq w_1 \\ \frac{6w_0+6w_1}{6w_0+w_1} & \text{when } w_1 \leq w_0 \end{cases} \quad . \tag{6.9}$$

To represent this graphically, we define v as $\frac{w_0}{w_1}$. Doing so, we can write (6.9) as:

$$S_{2,\{w_0,w_1\}} = \begin{cases} \frac{6v+6}{v+6} & \text{when } v \leq 1 \\ \frac{6v+6}{6v+1} & \text{when } 1 \leq v \end{cases} \quad . \tag{6.10}$$

This is set out in Fig. 6.5. We can clearly see that a maximum speedup of 1.7 is achieved when $v = 1$ or in other words, when both exponents have the same length. Also, when an exponent is less than twice as long as the other ($0.5 < v < 2$), a speedup of at least 1.4 can be achieved.

Experimental verification of the speedup for $l = 2$. With our embedded test setup, we were able to verify the correctness of Eqn. (6.8). We varied w_0 and w_1 between 256 and 2048 bits. The expected speedup has been set out in Fig. 6.6(a). The gray scale represents the speedup in steps of 0.5. We then used the times

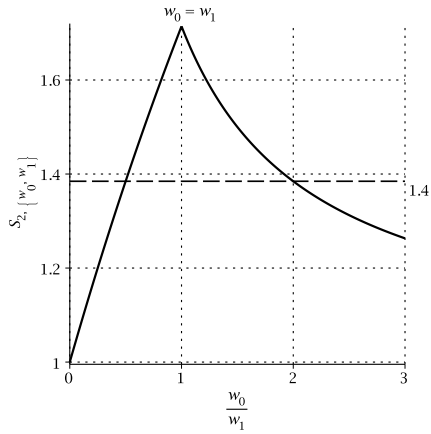
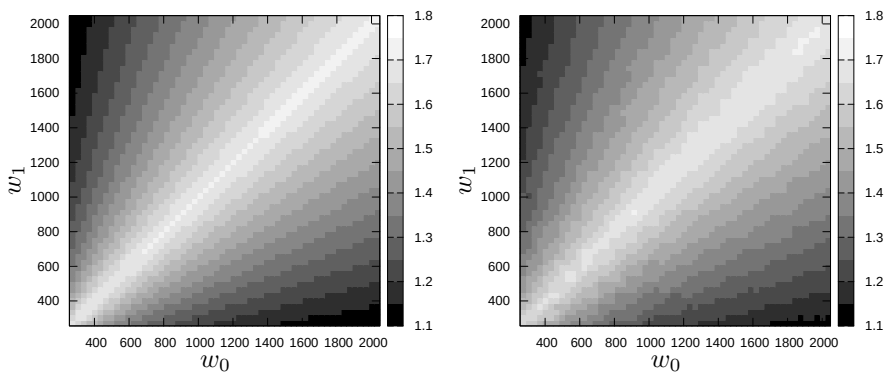


Figure 6.5: Speedup factor when using dual-base instead of single-base exponentiation as a function of the exponent length (ratio).

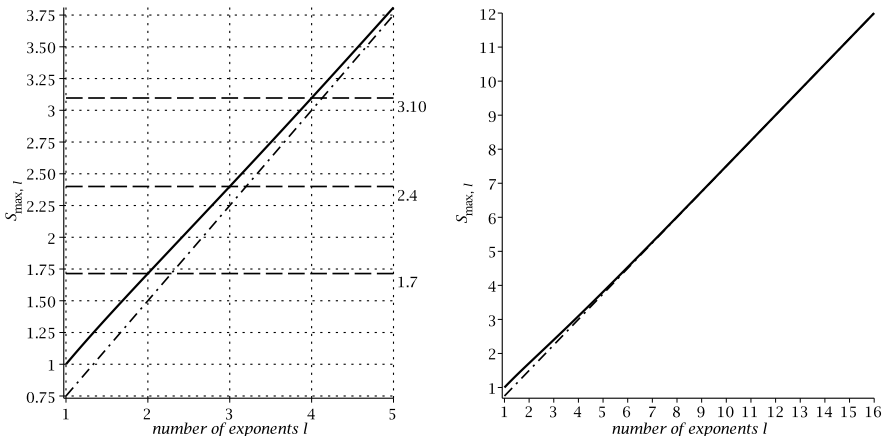


(a) $S_{2,\{w_0,w_1\}}$ as predicted from (6.8)

(b) $S_{2,\{w_0,w_1\}}$ as measured with the embedded test setup.

Figure 6.6: $S_{2,\{w_0,w_1\}}$: the speedup factor when using dual-base simultaneous exponentiation compared to single-base exponentiation.

measured with our embedded test platform to generate Fig. 6.6(b). We can clearly see that especially for larger values of w, the measured results follow the predicted values. The reason that there is a difference for smaller exponents is presumably the fact that Eqn. (6.7) doesn't take the overhead into account. This overhead is smaller for single-base exponentiations because only one multiplication is required in the precomputation step. Since the overhead was independent of w its *relative share* of the total computation time will also be lower in this case, and thus higher in case of a simultaneous exponentiation.



(a) $S_{max,l}$ for several numbers of exponents. (b) $S_{max,l}$ evolving to $\frac{3}{4}l$ (dash-dot line).

Figure 6.7: $S_{max,l}$: the maximum speedup factor when using l-base simultaneous exponentiation compared to single-base exponentiation.

General expression for l exponents with equal length w. Knowing that a maximum speedup can be achieved when the exponents are all of the same length, we can express this speedup as a function of the number of exponents:

$$S_{max,l} = \frac{lT_{1,w}^{(\wedge)}}{T_{l,w}^{(\wedge)}} \quad . \tag{6.11}$$

Again, this is independent of $T_{k,n}^{(\times)}$. $S_{max,l}$ further simplifies to:

$$S_{max,l} = \frac{3}{2}\frac{l4^l}{2^{1+l}-1} \quad . \tag{6.12}$$

$S_{max,l}$ is presented in Fig. 6.7(a) for different values of l. Note that Eqn. (6.12) is only dependent on the number of exponents and not on their length.[2] Furthermore, we can see that for larger values of l the trend for $S_{max,l}$ becomes $\frac{3}{4}l$. Fig. 6.7(b) shows how $S_{max,l}$ evolves towards this trend. Note that for $l = 2$ we get the same speedup factor of 1.7. Keep in mind, however, that the required memory evolves exponentially with l (see Chapter 4).

[2]Of course, to have a maximum speedup, the exponents need to have an equal length.

A realistic example

For an `Identity Mixer` credential verification (only master secret) a 4-base exponentiation needs to be carried out (see Chapter 9). With a modulus length of 1536 bits, the exponents have the following lengths: $\{w_0 = 256, w_1 = 608, w_2 = 864, w_3 = 2592\}$. Mind that some of these lengths have been increased to meet the constraint of the IP core; i.e., all exponent lengths should be a multiple of 32.

In the case of single exponentiations (T_1) the total computation time is given by Eqn. (6.13). Dual-exponentiation (T_2) and 4-base exponentiation (T_4) run times are given by Eqn. (6.14) and Eqn. (6.15) respectively.

$$T_1 = \left[\frac{3}{2}256 + \frac{3}{2}608 + \frac{3}{2}864 + \frac{3}{2}2592\right] T_{k,n}^{(\times)} \ . \tag{6.13}$$

$$= 6480 \cdot T_{k,n}^{(\times)} \ .$$

$$T_2 = \left[\frac{7}{4}(256) + \frac{3}{2}(608 - 256) + \frac{7}{4}864 + \frac{3}{2}(2592 - 864)\right] T_{k,n}^{(\times)} \ . \tag{6.14}$$

$$= 5080 \cdot T_{k,n}^{(\times)} \ .$$

$$T_4 = \left[\frac{31}{16}(256) + \frac{15}{8}(608 - 256) + \frac{7}{4}(864 - 608) + \frac{3}{2}(2592 - 864)\right] T_{k,n}^{(\times)} \ . \tag{6.15}$$

$$= 4142 \cdot T_{k,n}^{(\times)} \ .$$

The speedup for using dual-base exponentiations is 1.28; this is significantly lower than the 1.7 maximum. A 4-base simultaneous exponentiation has a 1.56 speedup, but for this to work it requires 4 times the amount of memory as for a dual-base exponentiation.

Alternatives for T_2

In Eqn. (6.14), the exponents are combined in such a way that the differences in length are minimal. However, two other options exist:

$$T_2(B) = \left[\frac{7}{4}(256) + \frac{3}{2}(864-256) + \frac{7}{4}608 + \frac{3}{2}(2592-608)\right] T_{k,n}^{(\times)} \quad . \tag{6.16}$$

$$T_2(C) = \left[\frac{7}{4}(256) + \frac{3}{2}(2592-256) + \frac{7}{4}608 + \frac{3}{2}(864-608)\right] T_{k,n}^{(\times)} \quad . \tag{6.17}$$

Both alternatives result in a run time of $5400 \cdot T_{k,n}^{(\times)}$, which is significantly larger than the proposed approach.

6.4 Proposal for a New Operand Memory Architecture

We have observed that with the current design method, we require 2^l memory locations for an l-base simultaneous exponentiation. This rapidly becomes infeasible, but we have seen that it is imperative that the required operands are immediately available to the multiplier when doing exponentiations. To cope with this exponential memory increase, we propose an alternative IP core design, that at the same time makes the design even more customizable.

We start with the assumption that the main memory used by the OS is shared and that it can be accessed by peripherals. Even more, peripherals should be able to read to and write from this memory without intervention of the central processor, i.e., Direct Memory Access (DMA). This requires either a DMA controller or a master/slave bus with a bus arbiter.

If that is the case, the core only needs a local operand cache and an address table. The local cache is always the same size, i.e., one location for the modulus, one for the intermediate result and at least one location to store an operand (all n bits). The address table holds the shared memory addresses where all the required operands are stored. Of course, also $2^l - 1$ entries need to be stored, but that requires considerably less space.[3]

When executing the main loop of the simultaneous exponentiation algorithm, the first step is squaring the intermediate result. During this step, the control logic

[3]The reason that it is $2^l - 1$ instead of 2^l because no address entry is needed for the intermediate result.

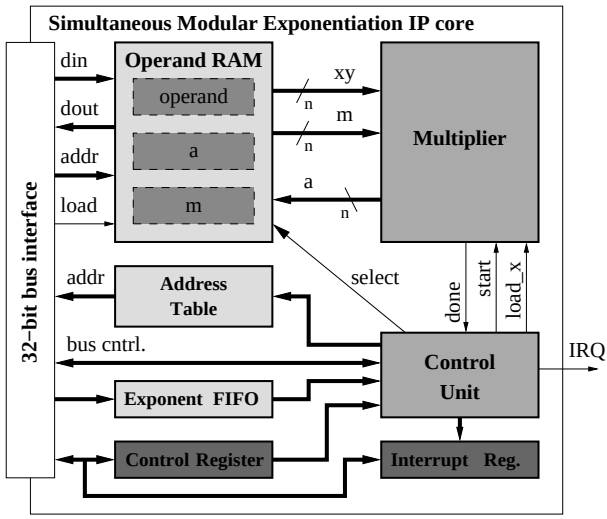


Figure 6.8: Provisional block diagram of the IP core redesign.

can analyse the exponent bits and with the corresponding the address from the address table, retrieve the required operand.[4] This is then stored in (one of) the operand memory cache(s), that is not being used during squaring. This approach is only useful under the condition that the time to fetch an operand is (ideally) lower than $T^{(\times)}_{k,n}$. In any case, extra logic is required to ensure that the multiply step is only executed when the operand is present in the cache.

Fig. 6.8 shows a provisional block diagram of the proposed design. The main difference with the previous design is the smaller RAM, the address table and more complex control logic and bus interface. Fig. 6.9 shows an example of the address table usage for $l = 4$. In this example, only a triple-base simultaneous exponentiation is executed, so one of the exponents is zero.

[4]When the exponent bits are all zero, no operand needs to be fetched, of course.

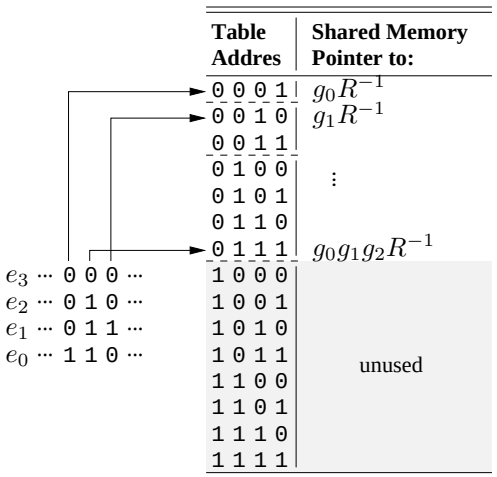


Figure 6.9: Example of the address table for $l = 4$ in case of a triple-base simultaneous exponentiation.

6.5 Conclusion

In this chapter we have verified some expressions that allow us to determine in advance computation times for multiplication, single-base exponentiation and l-base simultaneous exponentiation. We found that in practice there is an overhead introduced by the latency in the OS and the communication with the IP core. This overhead is constant regardless of w and n, but it is dependent on the system setup e.g., clock frequency, CPU tics,...

It is clear that there is a speedup in using simultaneous exponentiations. We have stated some expressions that allow us to determine the maximum speedup, as well as the effective speedup in a practical case. For this practical case, we have also shown that the fastest run times for dual-base exponentiations are achieved when the exponent length difference is minimal.

Together with the known memory usage, the general expressions allow us to make an educated trade-off between speed and area for a given case; i.e., whether to use l-base simultaneous exponentiations or not. Moreover, the influence of the overhead on the speedup decreases when the exponents are longer. In practice, for a dual-base exponentiation $2n$-bit of extra operand memory is required, resulting in a maximum speedup of 1.7.

In addition, the analysis shows that the speedup is independent of $T_{l,w}^{(\wedge)}$. This means that, when Algorithm 1 is used and the multiplication timing is constant for a given operand length, this analysis is generally applicable for any multiplier

implementation.

We have proposed a new IP core memory architecture that can be used when shared memory is available on the embedded system: its advantage is that we can benefit from simultaneous exponentiations without increasing the operand memory.

We have also evaluated the performance of the IP core in some realistic scenarios i.e., attribute-based credential verification. This has taught us the optimal way of combining multiple exponentiations, as well as the practical speedup we can expect. We will evaluate this in practice in Chapter 9.

Chapter 7

NFC Peer-to-Peer Communication

7.1 Introduction

As a consequence of the mobile revolution, smartphones and tablet computers will increasingly be used to gain access to electronic services, e.g., mobile payment to a vending machine, access control, ... When the user is (required to be) near the verifying device, Near-Field Communication (NFC) could be useful as a communication medium.

In this chapter, we will investigate which features NFC offers and how it relates to other wireless technologies (Section 7.2). Because two-way communication is required by most of the classic challenge-response protocols as well as by the more recently developed attribute-based credential protocols, the standard for *peer-to-peer* (P2P) communication with NFC is presented in Section 7.3. More specifically, we will focus on the communication between a smartphone and an embedded terminal.

To that end we will use an Android phone. Android OS is one of the most-used operating systems for smartphones.[1] Moreover, developing apps for Android is easier compared to iOS or Windows 8 because no developer account is required. Also, when needed, root access to the phone can be enabled. For the bidirectional communication between an Android smartphone and an embedded terminal, we

[1] According to IDC, Android and iOS Combine for 92.3% of all smartphone operating system shipments in the first quarter of 2013: `http://www.idc.com/getdoc.jsp?containerId=prUS24108913`. Android takes a market share of 75%. This share only includes the smartphone market and not, e.g., tablets.

will look at the current state of the art for NFC P2P and where needed select a modus operandi (Section 7.4 for Android OS and Section 7.5 for the embedded terminal).

Our conclusions are stated in Section 7.6.

7.2 Near Field Communication

7.2.1 Applications

NFC is a relatively new wireless communication standard. Unlike Bluetooth or WiFi, it only works over short distances, i.e., a few centimeters maximum. On the other hand, connection setup times are considerably lower. Bringing two NFC devices close to each other, is enough to set up a connection and exchange data i.e., the so-called *touch*.

The NFC Forum[2] regulates the design of all NFC specifications and norms. Started in 2004, it brings together manufacturers and service providers. Among its members are NXP, Nokia, VISA, Samsung, etc. The main goal of the NFC Forum is to ensure interoperability of devices and protocols and thus help in building the so-called *NFC ecosystem*.

In ticketing applications, NFC is gradually replacing paper tickets. This is, for instance, the case in Dutch public transportation or in the London Metro, with respectively the OV-chipkaart[3] and Oyster Card.[4] A first advantage is that NFC cards can be reused over and over again, which saves on ink, paper and storage [NFC11]. But tickets are also increasingly being stored on NFC-enabled smartphones, which are less prone to loss. In Paris' and London's public transportation, commuters can use their smartphone as a ticket [Cla12, Hea11]. NFC ticketing systems are furthermore assumed to increase the throughput of the public transportation [NFC11].

Predictions state that in 2016, 13% of US and Western Europe citizens will use their smartphones as a ticket [The12]. Manufacturers such as Asus, HTC, Nokia and Samsung have all released NFC-enabled devices. It is expected that 46% of all the smartphones will support NFC by 2016. The only uncertainty is Apple, which has to date not launched any NFC phones.

[2]NFC Forum website: `http://www.nfc-forum.org`

[3]OV-chipkaart information website: `https://www.ov-chipkaart.nl/`

[4]What is Oyster? Transport for London – Information website: `http://www.tfl.gov.uk/tickets/14836.aspx`

7.2.2 Devices and Communication Modes

NFC can be seen as an extension to Radio Frequency Identification (RFID) operation at 13.56 MHz defined by the ISO 14443 standard [1, 2, 3, 4]. This implies that for NFC two types of devices are defined:

- *Passive devices.* These devices –often referred to as *tags*– don't carry a battery and typically have very limited processing power and memory. The functionality ranges of storage of a simple ID (e.g., a classic RFID tag), over storage of (secured) data (e.g., a Mifare card, smart posters), to devices with a cryptographic co-processor (e.g., Java cards). Passive devices draw their power from an RF field generated by an active device.
- *Active devices.* An active device –also called *NFC-enabled* device– has its own power source. This can be either a net supply or a battery. It typically has more processing power than a passive device. Examples of NFC-enabled devices are (some) smartphones and access points.

One of the main differences between RFID and NFC is that with NFC, active devices are also able to communicate with each other. In this case both devices can alternately generate their own RF field when sending, or one of the devices can decide to act as a passive device. Because of the difference in communication between passive and active devices on the one hand and communication between two active devices on the other, the NFC forum has defined three different modes of communication:

- *Reader/writer mode.* This is the "classic" RFID communication. The active device acts as reader to read tags of the ISO/IEC 14443 A/B, Felica, etc.
- *Peer-to-peer mode.* In this mode, two NFC-enabled devices can exchange data with one another, e.g., digital business cards, an interesting URL, ... The data rate, however, is small. For this reason, a connection handover mechanism is used when large amounts of data have to be transferred. With this mechanism, all necessary parameters to set up a connection over, e.g., WiFi or Bluetooth, are transferred over NFC P2P. After the connection has been established, the NFC communication is terminated.[5]
- *Card emulation mode.* In this alternative to P2P, an NFC-enabled device will act as a traditional RFID tag. The main advantage is that it allows NFC smart phones to easily blend into the market, without the need to change the existing infrastructure (i.e., the existing card readers). However, in this mode the NFC chip of the smartphone communicates directly with a secure element on the phone (typically a SIM card), which eliminates intervention

[5]The connection handover mechanism is standardized in the ISO/IEC 18092 spec. [5]

of the processor and the OS altogether. This is e.g., the case with *Google Wallet*.

7.2.3 The Need for P2P Communication

Smartphones are used increasingly for mobile payment and the use of NFC as a medium is also growing. Several applications already exist where the credentials are stored on a secure element (SE) e.g., *Google Wallet*,[6] *Isis*,[7] ... and where the phone operates in card emulation mode. However, this approach has several drawbacks.

Because the communication from the NFC chip is immediately routed to the SE, this element forms a bottleneck. First of all, payment apps are limited to the memory size of the SE. Second, the processing and access times to SEs are higher. A last drawback is that provisioning credentials to a SE is a complex process. In practice, this implies that every *e-wallet app* requires its own SE.

An alternative to using card emulation with SEs is the so-called *software card emulation*. Instead of passing the NFC communication to the SE, it is captured by an NFC service in the OS. This approach breaks the dependency on the SE i.e., credentials can be stored anywhere (e.g., in the phone's user memory, within a trusted execution environment (TEE), even in "the cloud"), but it also allows for several payment applications to use the NFC functionality.

Roland has written a comprehensive article on this subject [Rol12]. His conclusion is that the main reason to go for software card emulation is that it does not require any changes to the existing payment infrastructure (i.e., the vendor terminals and network infrastructure). However, he also states that the most logical choice for payment and ticketing applications over NFC is to go for P2P because it was designed for easy communication between NFC devices. Following this conclusion, we will add support for NFC P2P communication to the embedded platform. We will look at the specification later (Section. 7.3) and how we implemented it on an embedded terminal (Section 7.5).

With regard to software card emulation, BlackBerry is the only company that supports this approach. There have been patches submitted for Android,[8] but none have been merged with the main Android branch. The only way this can be achieved now on Android is by rooting the phone and installing a custom ROM like CyanogenMod;[9] something that would void your phone's warranty.

[6] `http://www.google.com/wallet/`

[7] `https://www.paywithisis.com/`

[8] The patch by SimplyTapp adds software card emulation to the Android OS: `https://github.com/CyanogenMod/android_external_libnfc-nxp`

[9] http://www.cyanogenmod.org/

7.2.4 Alternative Short-Range Wireless Communication Technologies

In this section we will try to give a short overview of wireless technologies that can be an alternative to NFC. The main focus will be on data rates, the time to set up a connection and the range. In Table 7.1, these parameters, together with some extra features, have been set out for the considered technologies.[10]

Classic RFID. RFID defined by the ISO/IEC 14443 standard [1, 2, 3, 4] operates at 13.56 MHz and was originally designed as an automatic identification technology to replace barcodes. The main difference with NFC is in the distinctive roles of active and passive devices. RFID readers are only used to read out a passive tag, they cannot communicate with one another. Because of the use in tracking and theft detection, the range of RFID is also larger than is the case with NFC, where "touching" two devices is part of the concept.

Bluetooth. Bluetooth defines three classes of devices, each with a different transmission power and thus a different range. Class 3 devices have a maximum transmission power of 1 mW, where class 1 devices can go to 100 mW. This corresponds to respective ranges of 1 m to 100 m. Operating at 2.4 GHz with frequency hopping over 79 channels, the pairing of two devices can take some time. This pairing involves a device discovery and connection key agreement.

The recently released Bluetooth 4.0 standard is a collection of three separate (and not inter-operable) standards:

- *Bluetooth 2.1*, which operates as described above.
- *Bluetooth low-energy* uses direct-sequence spread spectrum instead of frequency-hopping. The long connection set-up of Bluetooth 2.1 is countered with a guaranteed set-up time of under 3 ms. Data rates, however, are much lower.
- *Enhanced Data Rate* is designed to achieve higher transmission speeds. This is accomplished by using a connection handover to another technology with a higher bandwidth. This is comparable to connection handover in NFC.

ZigBee. Designed for low-power wireless sensor networks (WSNs), ZigBee operates at 2.4 GHz. Intended for sensing and actuator control (e.g., home automation), the data rate is rather low. The inter-node distance will be around

[10] The listed data rates are in fact the maximum raw data rates; practical data rates (as seen by the application) can be an order of magnitude lower.

10 m (indoor), but the WSN itself –with thousands of nodes– can obviously span a large area. A node can connect to the network in a few seconds, key agreement and service discovery excluded.

WLAN. Wireless LAN is designed to build computer networks without the need of cables. WLAN is described by the IEEE 802.11 standard of which versions *a*, *b*, *g* and *n* are amongst the best-known. The latter supports data rates up to 600 Mbit/s. WLAN typically requires access points to connect to the existing (cable) LAN. WLAN can operate at both 2.4 GHz and 5 GHz with ranges up to a few tens of meters. NFC is not a competitor for WLAN, but more of a complementary technology. Especially for the *Internet of Things*, it can be interesting to use NFC to commission devices that have no other input options, a so-called *headless device*. Via the NFC connection, the required WLAN network parameters can be sent to the headless device i.e., NFC connection handover.

QR codes. A QR code is a 2-dimensional bar code. It can, however, store more data than a regular bar code and also offers more error correction. To scan a bar code, the user has to start the correct application, and point the camera at the QR code. Bidirectional communication is possible between two smartphones if they alternately generate and scan QR codes.

Despite the limited capabilities, QR codes have been used in several mobile security applications. In his PhD thesis, Lapon describes an application that uses QR codes for mobile authentication towards a terminal [Lap12]. The recently released *Bancontact/Mister Cash App* also uses QR codes for mobile payments between two smart phones or tablets.[11]

7.2.5 Security Issues

Bringing two devices into each other's vicinity is enough to start data exchange with NFC. This consequently introduces some security threats [Pau07]. We will give an overview of the attacks that are most common i.e., they can be staged in practice.

Eavesdropping. Almost all wireless communications can be subject to eavesdropping. All that is required is an antenna and some analysis equipment. Hancke has demonstrated this attack for RFID tags operating at 13.56 MHz [Han08]. However, it is widely accepted that NFC is harder to eavesdrop because of the lower transmit power. Also the exact difficulty can not be determined easily, because it depends on

[11] `http://www.bancontact.com/en/cardholders`

Table 7.1: Comparison of different short-range wireless technologies.

Feature	**NFC**	**Bluetooth** (2.1 + EDR)	**ZigBee**	**WLAN** (802.11n)	**RFID**	**QR code**
range	10 cm (max.)	100 m (class 3)	10 m (indoor)	100 m	> 1 m	depends on camera and size
frequency band	13.56 MHZ	2.4 GHz	2.4 GHz	2.4 GHz 5 GHz	13.56 MHz but also in LF and UHF	/
data rate	424 kbit/s	3 Mbit/s	250 kbit/s	600 Mbit/s	848 kbit/s	/
setup time	< 0.1 s	seconds	seconds	seconds	< 0.1 s	/
passive devices?	yes	no	no	no	yes	yes
availability	low	high	low	high	average	average
applications	smart posters, mobile payment, ticketing, ...	audio streaming, file exchange, ...	home control, distributed sensing, ...	wireless LAN	product tracing, identification, ...	smart posters

a broad range of parameters e.g., the transmit power, signal noise, the environment (like metallic objects), the sensitivity of the attacker's equipment,.

Haselsteiner and Breitfuß [HB06] also state that it is more difficult to eavesdrop on a passive device, than on an active device. They also proclaim a maximum range for eavesdropping on NFCIP-1 and ISO/IEC 14443 devices. An attacker can listen to a passive device at a distance of 1 m, while for active devices this is 10 m. Eavesdropping can be countered by encrypting the data.

Data modification. Probably the easiest attack to perform is a *denial of service* (DoS) attack. By sending a sufficiently powerful sine wave at 13.56 MHz, the data modulation will be altered in such a way that no meaningful communication is possible. Two communicating devices cannot shield themselves from such an attack, they can only detect it.

Altering the data in a meaningful way is a lot harder to do. All RFID standards at 13.56 MHz use Amplitude Shift Keying (ASK) to modulate the carrier. Haselsteiner and Breitfuß [HB06] have shown that for an attacker, it is feasible to change a low amplitude into a higher amplitude, but that it is impossible the other way around. Because for bit rates higher than 106 kbit/s, a 10% modulation depth is used (instead of 100% for 106 kbit/s), Van Damme and Wouters state that it is easier to perform data modification at higher bit rates [DW09]. The most straightforward way to counter data modification is to use message authentication.

Data insertion. This attack is possible if a tag needs a lot of time to generate an answer and if the attacker can generate an answer more quickly. If the fraudulent and the real answer overlap, the result is a collision and hence data corruption occurs. Data insertion can be countered by authentication.

Man-in-the-Middle-Attack. In this case, the communication happens between three parties, one of which is the attacker. In order to execute a successful attack, the two genuine parties must be unaware the communication is passing through a third party. This attack is practically impossible to execute without resulting in collisions [DW09].

Relay Attack. This attack is also relatively easy to stage [FHMM10, FHMM11, HMM09, KW05]. The main goal of this attack is to trick two NFC devices into "thinking" they are into each others vicinity, while in fact their traffic is routed over a larger distance using e.g., Bluetooth, WiFi or GPRS. The attacker does not alter the data traffic, but only wants to gain access to the service which is enabled by the communication. Possible targets are wireless smart cards used for

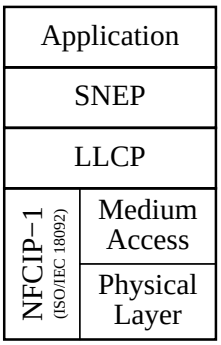


Figure 7.1: NFC P2P communication stack.

building access control or access to storage lockers. With that regard, Francis et al. [FHMM11] staged such an attack by using two NFC-enabled mobile phones. One phone operates in reader/writer mode and acts as a proxy between the smart card and the relay channel. The other phone acts as a card near the actual terminal.

One way to counter relay attacks is by employing distance bounding protocols [RTŠ+12, HPO13, PH12]. Current RFID communication protocols do not support this (at the physical layer), because these protocols typically require some rapid data exchange phase. Another way protecting honest users against relay attacks, is by asking for their permission before initiating communication.

7.3 NFC P2P Communication

For two active NFC devices to communicate with each other (e.g., a smartphone and a vending machine), the NFC Forum defines a *peer-to-peer* (P2P) communication mode. NFC P2P requires a stack of several protocol layers as shown in Fig. 7.1. The SNEP layer and LLCP layer are defined by the NFC Forum itself. The lower layers are manufacturer and hardware dependent. In the following paragraphs, we will provide a short overview of all the layers. Because we made our own basic implementation of SNEP on an embedded platform (Chapter 2), we will go into more detail when it comes to this layer.

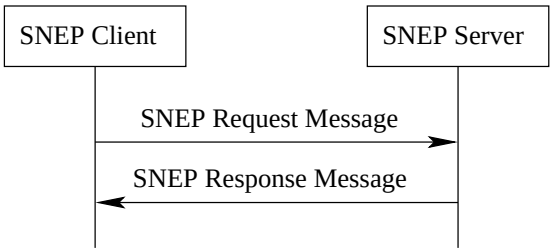


Figure 7.2: SNEP communication model.

7.3.1 Simple NDEF Exchange Protocol

The Simple NDEF Exchange Protocol (SNEP) is a request/response protocol for exchanging NDEF[12] messages between a client and a server [31]. To exchange these messages SNEP uses the stable logic connection between two devices, provided by the underlying LLCP layer.

SNEP message format. SNEP messages (Fig. 7.3) consist of four fields.

- The *version* field designates which version of the protocol is being used. The upper nibble represents the major version number, while the lower nibble represents the minor version number. At the time of writing, the most recent version is `1.0`.
- The *request/response* field indicates the type of the SNEP message. Currently, four request codes are defined:

  `0x00` *CONTINUE*: The client indicates that the server can send another fragment of a fragmented SNEP message.[13] This can only follow a previously sent *GET*. The information field in a *CONTINUE* message is empty.

  `0x01` *GET*: This code is used to request an NDEF message (see later) from the server. The type of NDEF message is determined by the NDEF message that is being sent in the information field of the *GET* request.

  `0x02` *PUT*: Used, when the client wants to sent an NDEF message. The actual message is sent in the information field.

[12] NDEF stands for NFC Data Exchange Format. This is discussed later on in this chapter (p. 99).

[13] This is the case when the information that is being sent, is too large to fit in the information field of a single message.

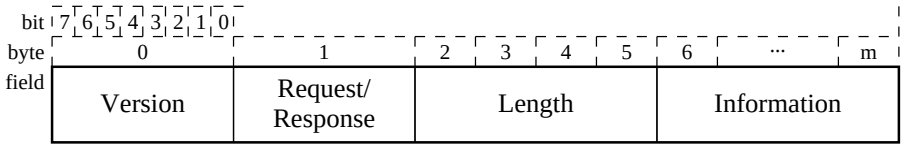


Figure 7.3: SNEP message format.

`0x7F` *REJECT*: When the client is not capable of receiving more messages (e.g., for a fragmented message), it sends a *REJECT*. This can imply that the logical connection will be terminated.

The possible response codes are defined between `0x80` and `0xFF`:

`0x80` *CONTINUE*: Analogous to the request.

`0x81` *SUCCESS*: Response to a *GET* or *PUT* request. In case of *GET*, the information field contains the requested NDEF message. Otherwise, the information field is empty.

`0xC0` *NOT FOUND*: When the server can not send the requested NDEF message (as response to a *GET* request).

`0xC1` *EXCESS DATA*: The response of the server is too large for one message.

`0xC2` *BAD REQUEST*: Response to a request that contains syntax errors.

`0xE0` *NOT IMPLEMENTED*: The specific request is not supported.

`0xE1` *UNSUPPORTED VERSION*: The version used by the client is not supported by the server.

`0xFF` *REJECT*: Analogous to the request.

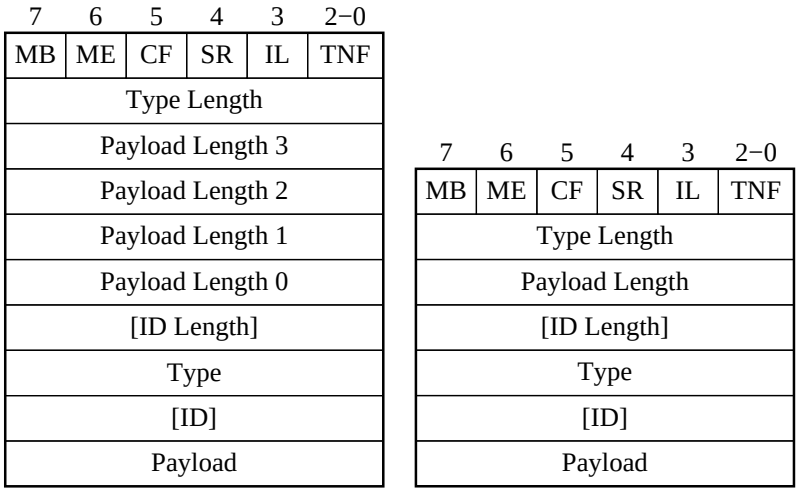
All undefined codes are reserved for future use.

- The *length* field (4 bytes) indicates the length of the information field (number of bytes).
- The *information* field is used to send the NDEF messages.

NDEF message format. The NFC Data Exchange Format (NDEF) defines the format of data packets for NFC communication [26]. For P2P communication, when one application sends some information to another, it must encapsulate this information in an NDEF message. An NDEF message consists of one or multiple NDEF records. This can be either a normal record or a short record (Fig. 7.4). Both have a comparable structure with the same fields:

- *Message Begin (MB)*: Set if it is the first record.

- *Message End (ME)*: Set if it is the last record. In case of an NDEF message consisting of a single record, both *MB* and *ME* are set.
- *Chunk Flag (CF)*: When data needs to be parsed over several records (chunking), the *CF* is set indicating that more records follow. Only in the last record is the *CF* cleared. For records following the initial record, *Type Length* and *IL* need to be zero, while *TNF* is set to `0x06` (unchanged).
- *Short Record (SR)*: Set to 1 if it concerns a short record. For a short record, the *Payload Length* field is only one byte (Fig. 7.4(b)), while for a normal record, it is four bytes (Fig. 7.4(a)).
- *ID present (IL)*: If set, then the *[ID]* and *[ID Length]* fields are present.
- *Type Name Format (TNF)*: These three bits indicate the structure of the *Type* field:
  - `000` Empty. Used when no payload is present. As a consequence, *Type Length*, *[ID Length]*, *Payload Length* are zero.
  - `001` NFC Forum well-known type (RTD). The *Type* field has a structure as defined in [27].
  - `010` Media-type. The *Type* field has a structure as defined in [21].
  - `011` Absolute URI. The *Type* field has a structure as defined in [28].
  - `100` NFC Forum external type (RTD). The *Type* field has a structure as defined in [27].
  - `101` Unknown.
  - `110` Unchanged; used in the case of chunking.
  - `111` Reserved.
- *Type Length*: The length of the *Type* field.
- *Payload Length*: The length of the *Payload* field.
- *[ID Length]*: An optional field, indicating the length of the *[ID]* field.
- *Type*: Describes the type of the data. This field should be consistent with the *TNF* bits.
- *[ID]*: An optional field. Can be used to identify data. The application is responsible for the implementation of this field.
- *Payload*: The actual data.



(a) Normal NDEF record (SR=0) (b) Short NDEF record (SR=1)

Figure 7.4: NDEF record format.

7.3.2 Logical Link Control Protocol

The Logical Link Control Protocol (LLCP) provides to higher layers a logical connection between two endpoints [30]. Every endpoint offers the same functionality. This eliminates the inherent asymmetry of the lower MAC layer with an initiator controlling the communication and a target device that only sends at a request of the initiator. LLCP creates a so-called *asynchronous balanced mode* (ABM) on top of this mechanism, through which each endpoint has the possibility to start a transmission.

Endpoints connect to the LLCP through *service access points* (SAPs). Typically, each protocol (endpoint) has its own SAP. For NFC, only SNEP has been assigned an SAP (`0x04`).[14]

Implementations on top of LLCP can either use connection-oriented or connectionless communication. Neither of these, however, have a guaranteed delivery time. This makes LLCP unusable for the streaming of audio or video. Other limitations are the fact that there is no possibility to send packets to different SAPs (multicast or broadcast) and that there is no support for encryption or authentication. This should be implemented by higher layers.

[14]The up-to-date list of assigned SAP addresses can be found at: `http://www.nfc-forum.org/specs/nfc_forum_assigned_numbers_register` .

7.3.3 NFCIP-1

All layers described above add a level of abstraction to the actual communication i.e., how bits are physically transmitted from one device to another. For NFC this is described in the ISO/IEC 18092 specification [5]. In [29], the NFC forum has tried to make a more concrete description of the protocol.

The NFCIP-1 protocol works according to an initiator/target model. The initiator sends commands to the target, to which the latter answers. A target can never send data at its own initiative. Because both devices will never transmit at the same time, NFC is a half-duplex protocol.

There is also a difference between passive targets and active targets, while this distinction is not made by higher layers. In active communication mode, a target will generate its own RF field when sending data to the initiator. A passive target relies on the RF field generated by the initiator. Through inductive coupling, the target can draw energy from the RF field. The target can be compared to the secondary winding and load of a transformer. By varying the load, the target can send data; the load variations can be detected by the initiator.

7.4 NFC P2P Implementation in Android OS

7.4.1 Android 4.1 – API 16 and Higher

The mechanism that handles NFC P2P communication in Android is called *Android Beam* (introduced with API 14). The API provides support for the exchange of NDEF messages, with support for different types of data. There are methods to create the NDEF records and combine them into a message.

To send an NDEF message, the API provides two different approaches. The first one will register a static message to be sent when another NFC-enabled device establishes communication. The second method allows to dynamically create an NDEF message at run time. This is interesting when context-sensitive information needs to be sent.

To receive an NDEF message an application needs to register the correct intent with the OS; in this case the `NDEF_DISCOVERED` intent. This is important because the *Tag Dispatch System* (Fig. 7.5) will try to immediately deliver all data received over NFC to the correct application. By using filters, the application can further specify if it is only interested in e.g., plain text NDEF messages. To prevent that a new instance of an application is started, every time an NDEF message is received, an active application can use the *Foreground Dispatch* to process the incoming intents.

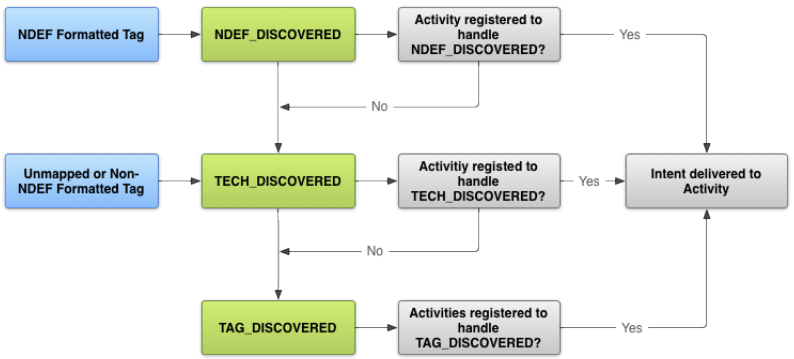


Figure 7.5: Android tag dispatch system – Taken from `http://developer.android.com/guide/topics/connectivity/nfc/nfc.html`

7.4.2 Internal Operation

If we look at the internal operation of the Android NFC service, one important flaw comes out, more specifically with respect to the NFC P2P specification. The procedure to send an NDEF message (whether it is a static message or composed at run time) is only called when the smart phone receives an *LLCP Link Activation* message. This message is part of the LLCP layer and is *only* sent when two devices are brought close enough to set up a connection. This means that, to be able to send multiple messages, the two devices have to be alternately brought together and moved away again [Ber13]. It is obvious that this is highly impractical. It also implies that currently there is no bidirectional communication possible (with more than one pass), as is required in security protocols.

A possible explanation is that before the release of the NFC P2P specification (by the NFC Forum), Google had its own protocol for exchanging NDEF messages, the NDEF Push Protocol (NPP) [13]. In NPP it is specified that a client has to sever the LLCP connection after sending a NDEF message. It is likely that SNEP was implemented without making any changes to the underlying framework, which explains why this feature is still present.

We note again that this is the case for all Android versions to date.

7.4.3 Solutions

As shown before, bidirectional P2P communication with Android is currently not possible. We will list some possible workarounds to establish P2P communication with the current API.

1. *Software card emulation.* As mentioned before, card emulation is another communication mode of NFC. When the SE element is replaced by a software application or service that acts like a SE, this is called software card emulation. However, this is not supported by the Android OS. Only custom builds like CyanogenMod support this mode of operation.

2. *External reader.* Another workaround could be in using an external reader which is connected to the smartphone with another interface e.g., Bluetooth or USB. In this case one could wonder for what reason, users would buy an extra NFC device while their smartphone supports NFC.

3. *Java Native Interface.* The Java Native Interface (JNI) is used to create a link from Java applications to code written in e.g., C or C++. This code is typically time-critical code, or low-level driver code. An option would be to create a replacement NFC service that implements the NFC P2P stack as defined by the NFC Forum. This would include the NFCIP-1 and LLCP (both written in C) coupled through JNI to a Java API that is usable by applications that require NFC P2P. The only problem with this approach, is the access to the NFC chip. The access is restricted to the root user or software that is signed by Google.

4. *Handover.* Currently the only solution that does not require tampering with the OS or external NFC device, is connection handover (standardized in ISO/IEC 18092 [5]). The drawback here is that this requires an extra wireless interface and that there is an increased connection setup time. The latter makes that this mechanism is only interesting when a large amount of data will be exchanged.

It is clear that all the proposed solutions are far from ideal, especially keeping in mind that there is a standard for NFC P2P. One could say that for the further evolution of NFC applications that require P2P communication, the ball is now in Google's court. On the other hand, we assume that eventually they will support the P2P standard. This is a reasonable assumption because 1) an NFC P2P standard is available and 2) applications, other than payment apps, could benefit from this as well.

Working JNI solution

In his master thesis, Van Den Berge demonstrates a solution based on JNI [Ber13]. He has created an application that serves as a proof-of-concept for NFC P2P communication (according to the specification) with an Android smartphone. His application bypasses the standard Android NFC service and is able to act as either an LLCP server or client. Only root access to the PN544 chip was required.

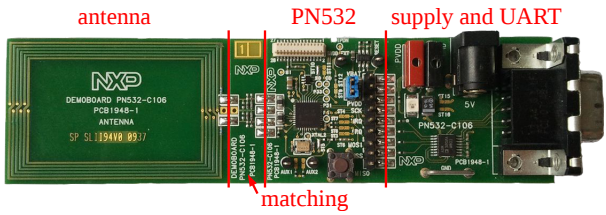


Figure 7.6: The NXP PN532C106 demo board.

7.5 NFC P2P Support on an Embedded Platform

Even if the current API of Android does not entirely follow the specification, we need to implement the NFC P2P stack on an embedded terminal (e.g., vending machine, point of sale, ...) as well. In the following paragraphs we will provide an overview of the libraries we selected for this purpose. We implemented these libraries on our embedded test platform (see Chapter 2). For the RF interface we use an NFC development board by NXP.

7.5.1 NFC Hardware

The NXP PN532C106 demo board is designed for prototyping NFC applications (Fig. 7.6). The heart of this board is formed by the PN532 NFC chip. An antenna and matching network are also provided as well as a power supply circuit and an RS-232 interface. The RS-232 high-speed UART (HSU) is one of the three interfaces through which a host can control the PN532 [32].

The PN532 and successors such as the PN544 are among the most-used NFC chips in smartphones. The PN532 supports four different modes of operation:

- support for ISO/IEC 14443A (Mifare) and FeliCa as reader
- card emulation as ISO/IEC 14443A and FeliCa tag
- support for ISO/IEC 14443B only as reader
- NFCIP-1, required for NFC P2P

The HSU operates at 119200 baud. A proprietary command/response instruction set is used to configure the PN532 and to exchange data.

7.5.2 Libraries for NFC P2P

Several open-source libraries exist that offer some functionality required for NFC P2P. One of the main requirements is that the libraries are able to run on an embedded platform. In particular we selected libraries that work with Linux and that support the PN532. We considered the following candidates:

- `open-nfc`[15]: This library is developed by the company *Inside Secure*. It supports the full range of modes of operation, P2P communication according to the specification as well as handover to Bluetooth and WiFi. It is available for Linux, Windows and Android, but a great disadvantage is the lack of hardware support (currently only Inside Secure's own hardware is supported). They offer a porting guide, but still this requires a great deal of knowledge of both the hardware and the `open-nfc` stack.
- `nfcpy`[16]: This library offers all the layers required for P2P communication as well as manipulation of several types of tags. It comes with several examples that aid in developing applications. The library is written completely in Python, which requires the presence of a Python interpreter on the embedded OS. Because this creates an extra level of overhead, this is obviously a disadvantage.
- `libnfc`[17]: A library with one of the largest communities; also one of the first. The source code is written entirely in C and supports a broad range of chips as well as a debug mode. It also offers a broad range of examples, illustrating the possibilities of the library. Although bidirectional communication is possible with this library, it does not support LLCP or SNEP.
- `libnfc-llcp`[18]: Also written in C, this library implements the LLCP layer on top of `libnfc`. There are some examples, but the source is poorly documented. Still, this library is popular amongst the `libnfc`-community.
- `Qt Mobility`[19]: The Qt Mobility module developed by Nokia, offers the LLCP layer and support for NDEF. However, it requires the Qt framework. Qt Mobility is written in C++.
- `nfctools`[20]: This library is written in Java. This ensures platform independence, but also requires a Java virtual machine (JVM) to operate. Currently `nfctools`, has been merged with `libnfc` and `libnfc-llcp` under the same project.

[15] Project website: `http://open-nfc.org/wp/`
[16] Project website: `http://nfcpy.readthedocs.org/en/latest/`
[17] Project website: `http://code.google.com/p/libnfc/`
[18] Project website: `http://code.google.com/p/libllcp/`
[19] Project website: `http://doc.qt.digia.com/qtmobility-1.2/index.html`
[20] Project website: `http://nfc-tools.org/`

- `ismb-snep-java`[21]: This library has evolved from the `ismb-npp-java` library, which was initially developed for indoor navigation. As its name suggests, the source is written in Java.

Keeping in mind the extra resources required by virtual machines, frameworks or interpreters, the only libraries that are fit for an embedded platform are those libraries that directly tap into the functionality provided by the OS (e.g., threading, memory management, device drivers), rather than doing this through an extra layer of abstraction. Note that on an Android platform, with applications running in a JVM, the NFCIP-1 and LLCP layers are also written in C.

This leaves only `libnfc` for NFCIP-1 and `libnfc-llcp` for the LLCP layer. That also means that a SNEP implementation and NDEF support has to be written by the developer. We have made such an implementation on our embedded test platform (Chapter 2). Appendix B shows a coding example on how to run a SNEP server application on an embedded terminal using our own SNEP implementation and the `libnfc` and `libnfc-llcp` libraries. The main drawback of the current implementation is that only short NDEF records are supported.

7.6 Conclusions

In this chapter we have given an overview of NFC as a communication medium between a mobile and an embedded device e.g., a smartphone and an NFC-enabled locker. More specifically, we have looked at the value of NFC for exchanging messages as part of a cryptographic protocol between an Android device and an embedded platform.

The NFC P2P specification, defined by the NFC forum, is the standard for bidirectional communication between two NFC devices. However, the current version of Android (i.e., API 16) implements this standard only partially. To that end, we have compared several alternatives. These are either complex (JNI, custom Android build, external reader) and/or require root access to the phone. Only communication handover to another wireless standard, that does support bidirectional communication, seems to be a valid alternative at the moment.

Of these alternatives, we implemented the JNI solution and the connection handover to WiFi. Both solutions were tested in combination with our embedded test platform (Chapter 2). Our findings are discussed in Chapter 10.

On the side of the embedded platform, we used two open-source libraries i.e., `libnfc` and `libnfc-llcp`, to implement the NFC P2P stack. Together with a basic own design of SNEP, we have implemented an example application for sending

[21] Project website: `http://code.google.com/p/ismb-snep-java/`

and receiving data over NFC P2P on an embedded platform. The main drawback of the current implementation is that only short NDEF records are supported and that the programmer is responsible for parsing long payloads over multiple records.

Chapter 8

Attribute-Based Credentials

8.1 Introduction

In a society that increasingly relies on electronic services, safeguarding a user's personal information is a daunting task. Service providers often collect more attributes than is required for running their service. While searching and browsing online stores and the Internet in general, users often reveal more information about themselves than they realize. This allows for extensive profiling of users, both for commercial and criminal purposes.

Classic certificate technology binds a public key to a certain identity. By using certificates both users and service providers can verify the identity of the other party. Certificates, however, contain a lot of information about the owner that is revealed during the authentication process. So while certificates have their usefulness in ensuring authenticated communication, they lack in offering privacy for their users. In addition, all actions of the same user i.e., with the same certificate, can be linked [Cha85].

An alternative to the classic certificate technology are the attribute-based credentials (ABC) or anonymous credentials cryptosystems [CL01, Cha85, Bra00, CL03]. This technology offers some interesting features:

- **Selective disclosure.** Like classic certificates, attribute based credentials contain a set of attributes, properties and values e.g., the user's name, gender and date of birth. A user can choose to only reveal a subset of these attributes, while keeping the remaining attributes hidden. This helps the user in maintaining his privacy.

- **Unlinkablity.** A user can use the credential several times but these actions cannot be linked.
- **Accountability.** In the case of abuse, a user's identity can be revealed.

Attribute-based credential systems can be divided into two major classes. The classification is based on the approach taken to prevent linking of interactions between a user and an issuer on one hand and between a user and relying parties on the other hand. To break the link between the user's credential and the issuer, the first class [Bra00] uses *blind signatures* [Cha85]. A blind signature means that the user's credential is signed without the issuer learning the resulting signature value. This way, when showing a credential, it cannot be used to link the user to the issuance phase, even when the relying party and the issuer share information. However, to make multiple transactions unlinkable, a user should use a different credential for every anonymous transaction. The second class, often referred to as Camenisch-Lysyanskaya (CL) based credentials [CL01, CL03], relies on *zero-knowledge proofs* to break the link between the issuance phase of the credential and its use. When a user wants to authenticate, he proves to the relying party that he possesses a genuine credential without revealing more than the fact that the credential is signed by a trusted issuer and that the user knows the corresponding private key (meaning he is the owner of the credential). The technique used is a so-called zero-knowledge proof-of-knowledge (ZKPK).

A practical implementation of the first class is `U-Prove` by Microsoft [35], while IBM has adopted the second class with the `Identity Mixer` [17]. Because of the computational complexity of the latter, it is challenging to implement the `Identity Mixer` in an embedded context. Testing the boundaries and offering solutions for embedded implementations of the `Identity Mixer`, will be a major topic in this thesis. We need to note that attribute-based credentials protocols should use anonymous communication [GRS96, RR98, DMS04, BFK01, KZG07] in order to fully provide their the privacy features. These measures could include e.g., hiding or randomizing the hardware and IP addresses. We provide a practical example to clarify the problem. Consider an e-voting application, where subscribers can anonymously participate in a weekly online poll regarding social and political themes. Even if attribute-based credentials are used, different votes of the same user can be linked, using his NIC hardware address, provided that he votes from the same computer. His IP address can be used to pin down a geographic location, or even find other personal information related to this address. In the case of NFC P2P, this unlinkability could be obtained by using a random NFC ID for each communication session. Setting a custom ID is supported by most NFC chips.

We start this chapter with an overview of some related work (Section 8.2). In Section 8.3, the key components and building blocks required for `Identity Mixer` attribute-based credentials system are explained. We will not go into detail and merely give the necessary concepts to understand the practical implementation.

Because the case study (Chapter 9) will focus on credential verification in an embedded context, we will detail the cryptographic operations needed for this in Section 8.4.

8.2 Related Work

Compared with traditional authentication technologies, attribute-based credentials based on CL signatures, require significantly more computation power for both proving and verifying. The main reason for this is the substantial amount of exponentiations during zero-knowledge proofs. If credentials are stored on mobile devices (e.g., mobile tokens, smart cards, ...) and the proving or verification is done in an embedded context (e.g., a vending machine), the performance might become a bottleneck. Several implementations of attribute-based credentials systems with such devices have been presented in the literature.

In 2009, Bichsel et al. [BCGS09] made a complete Java Card implementation of attribute-based credentials. In this case, the Java Card had to compute the proof, which took about 7.4 s for a 1280-bit modulus, and up to 16.5 s for a 1984-bit modulus.[1] To increase efficiency, both [BCD+12] and [SGPV09] present an approach where the computation of the proof is divided between a partially trusted host with more processing power (e.g., a smartphone) and a tamperproof smartcard. The drawback of this approach is that partial trust on the host is required. A practical implementation of this approach, targeted for ticketing applications over NFC has been demonstrated by Derler et al. [DPWD12]. They use a Javacard secure element inside the mobile phone to compute the proof. They show that credential verification takes significantly longer when less attributes are being revealed. For example, when revealing one attribute, the verification takes 7.7 s and 16.7 s for a 768-bit modulus and a 1984-bit modulus respectively. However, when revealing four attributes, the verification time is reduced to 2 s and 4.5 s respectively. Unfortunately, all the previous results do not differentiate between the time required for communication and the time required for arithmetic.

One of the most recent implementations of the CL-based prover protocol on smart card is the one of Vullers and Alpár [VA13], that uses the MULTOS card [FM05] which has better support for modular arithmetic. For a simple credential proof,

[1] The length of the modulus determines to a large extent the security level and computational effort of the algorithm. Based on Lenstra and Verheul's work [LV00] (updated in 2004 [Len04]) a modulus length of 1984 bits can provide security until the year 2038. The ECRYPT II recommendations [Eur12] describe the security level as a *Legacy standard level*. In contrast, a modulus of 768 bits is not secure as it only provided security until 1999, allowing for *attacks in "real-time" by individuals* according to ECRYPT. The meaning (and use) of the modulus is explained later in Section 8.3.

the computation takes about 1.1 s for a 1024-bit modulus. This result shows that CL-based credentials on smart cards may become practical in the near future.

The protocols themselves are also still evolving, mainly when it comes to the attributes. In 2008, Camenisch and Groß [CG08] extended the classic CL scheme in such a way that selective disclosure became more efficient. Another interesting concept in the area of attribute-based credentials is the use of *encrypted attributes*, introduced by Guajardo et al. [GMS10]. Encrypted credentials allow for the possibility that none of the involved parties, including the user, learns the values of the attributes.

Aside from the CL-based implementations, there are also prototypes using `U-Prove` [35] attribute-based credentials, which take about 5 s for showing a credential with two attributes and a 1280-bit modulus [TJ09]. A MULTOS card implementation of the same protocol by Mostowski and Vullers [MV12], resulted in computation times of about 0.5 s. This is faster than CL-based credentials but, for smart cards and memory-limited devices in general, the U-Prove system might become impractical. When it comes to unlinkability, a new credential has to be issued for each transaction, which may quickly exhaust the EEPROM of the card [BCGS09].

8.3 Concepts of CL-Based Credentials

In this section we will give an overview of the concepts used in CL-based credentials and more specifically in how they are used in the `Identity Mixer` specification. Because this thesis will mainly focus on credential verification, we will only discuss the components required to understand that mechanism. Because they are out of the scope of this text, we will not explain concepts such as anonymity or credential revocation, credential updates, or range proofs.

8.3.1 Mathematical Concepts

Groups. A set of elements together with an operation working on any two elements of that set, resulting in an element from that set, is called a group. All groups satisfy four conditions (or group axioms). In case of a group $\mathbb{G}$ with operation $*$ those conditions are stated as:

Closure: $\forall\, a, b \in \mathbb{G} : a * b \in \mathbb{G}$

Associativity: $\forall\, a, b, c \in \mathbb{G} : (a * b) * c = a * (b * c)$

Identity: $\exists\, e \in \mathbb{G} : a \in \mathbb{G} : a * e = e * a = a$ (e is the identity element)

Invertablity: $\forall\, a \in \mathbb{G} : \exists\, a^{-1} \in \mathbb{G} : a * a^{-1} = e$

A group can have several additional properties. If the operation $*$ is also commutative (i.e., $\forall\, a, b \in \mathbb{G} : a * b = b * a$) the group is called *abelian*. Groups can have a *finite* set of elements e.g., the multiplicative group of integers modulo n. The size $|\mathbb{G}|$ of a finite group is called the *order*.

A cyclic group means that there is an element $g \in \mathbb{G}$ such that for each $y \in \mathbb{G}$ there exists an integer x so that $y = g^x$. Here, g is called a generator of $\mathbb{G}$.

Subgroups. For $g \in \mathbb{G}$, the set of all powers of g is a cyclic subgroup of $\mathbb{G}$. This subgroup, denoted as $\langle g \rangle$, is called the subgroup generated by g.

Quadratic residue. Let $\mathbb{Z}_n^*$ be the multiplicative group of integers modulo n, then $a \in \mathbb{Z}_n^*$ is called a *quadratic residue* if there is an $x \in \mathbb{Z}_n^*$ such that $x^2 \equiv a \mod n$. Then $\mathbb{Q}_n$ denotes the set of all quadratic residues modulo n.

The Euler totient function. The number of integers in the interval $[1, n]$ (for $n \geq 1$) which are relatively prime to n, is denoted as the Euler totient function, represented as $\phi(n)$. When p is a prime number, then $\phi(p) = p - 1$. When the greatest common divisor of two integers p and q is 1, then $\phi(pq) = \phi(p)\phi(q)$.

RSA modulus. An RSA modulus n is generated by choosing two large random (and distinct) primes p and q, each roughly the same size. Then $n = pq$.

For the RSA algorithm, a random integer $e \in]1, \phi(n)[$ is selected, such that the greatest common divisor of e and $\phi(n)$ is 1. This is the public exponent and forms, together with n, the public key. The private exponent $d \in]1, \phi(n)[$ is computed from $ed \equiv 1 \mod \phi(n)$.

Notation. To make formulation of the protocol easier, we introduce some notations:

- A signed (binary) value x with a bit length ℓ is denoted as $x \in \pm\{0, 1\}^\ell$.
- The symbol $\|$ means a concatenation of two values.
- The function $H()$ represents a hash function. Throughout this work we will always use SHA-256 [25], unless specified otherwise.

Assumptions. An attacker with unlimited processing power can crack almost any security algorithm. Therefor we make the assumption that the attacker is also

restricted to the current state of the technology. We recall the assumptions that are used by the protocols presented in this chapter.

1. The *Discrete Logarithm problem* [DH76, McC90] requires that, for $g \in \mathbb{G}$ (multiplicative group) and $y \in \langle g \rangle$ it is hard to find an integer x so that $g^x = y$.
2. The *RSA assumption* [RSA78] requires that, given an RSA modulus n, a prime e and a random element $y \in \mathbb{Z}_n^*$, it is hard to compute $x \in \mathbb{Z}_n^*$ so that $x^e \equiv y \mod n$.
3. The *Strong RSA assumption* [FO97] requires that, for an RSA modulus n and a random element $y \in \mathbb{Z}_n^*$, it is hard to compute $x \in \mathbb{Z}_n^*$ and $e \in \mathbb{Z} > 1$ so that $x^e \equiv y \mod n$. The CL-signature scheme (explained later) is secure against adaptive chosen message attacks [GMR88] under the strong RSA assumption [CL03].

8.3.2 Cryptographic Concepts

Roles. The `Identity Mixer` specification defines several roles in an attribute-based credentials system. From a protocol perspective we can distinguish *issuers* (I), *recipients* (or *owners*), *provers* (P) and *verifiers* (V). The terms issuing party (IP), user (U) and relying party (RP) are mostly used from an application point of view. During the credential issuance phase an issuing party and a user collaborate to generate a credential. The user will act as prover to prove to a relying party that he is the owner of the credential. In this case the relying party acts as verifier of the proof.

Credentials and attributes. The attributes are a set of values associated with the owner of the credential. This can include features such as name, age, gender, nationality, etc. The attributes are all transformed into a numeric value that can be used in the cryptographic constructions. The attributes are bound to the owner by means of the owner's master secret in the issuance phase. The cryptographic information for this binding, together with the set of attributes form the credential. It is assumed that binding the attributes to the user's master secret will prevent sharing of the credentials as this would result in the user revealing his master secret. When the user wants to use his credential to access a certain service, he generates a proof of possession (of the credential). During this proof, he can decide (be asked) to reveal certain attributes (e.g., age and nationality). The set of revealed attributes is denoted as A_r, while the set of hidden attributes is A_h ($A_{\overline{r}}$ is also used). Note that the master secret is always contained in A_h.

The CL Signature Scheme. When an issuing party issues a credential to a user, he generates a signature on the user's set of attributes. This allows the relying party to (1) verify that the credential has not been altered and (2) that the credential has been issued by the issuer. During the issuance, the user verifies the correctness of the signature. The CL-signature scheme, between issuer I and verifier V (the owner of the credential), for blocks of L messages (the attributes) as implemented in [17] and based on [CL03], goes as follows:

I : $(pk_{Sig}, sk_{Sig}) \leftarrow \mathsf{setup}_{\mathsf{CL}}(l_n)$
Choose a special RSA modulus $n = pq$ of length l_n with $p = 2p' + 1, q = 2q' + 1$ where p, q, p' and q' are prime. Then choose, uniformly at random $S \in \mathbb{Q}_n$ and $Z, R_0, ..., R_{L-1} \in \langle S \rangle$. Then the public key $pk_{Sig} = (n, R_0, \ldots, R_{L-1}, S, Z)$ and secret key $sk_{Sig} = (p)$.

I : $(\sigma) \leftarrow \mathsf{sign}_{\mathsf{CL}}(m_0, \ldots, m_{L-1}, sk_{Sig})$
Let l_m be a parameter defining the message space as $m_i \in \pm\{0,1\}^{l_m}$ for $0 < i < L$. Choose a random prime e of length $l_e > l_m + 2$ and a random number $v \in_R \pm\{0,1\}^{l_n + l_m + l_r}$, with l_r a security parameter, and compute the signature $\sigma = (A, e, v)$ on $(m_0, ..., m_{L-1})$ such that $A^e \equiv \frac{Z}{R_0^{m_0} \ldots R_L^{m_{L-1}} S^v} \mod n$. The master secret will always be m_0.

V : $(Bool) \leftarrow \mathsf{verify}(\sigma, m_0, \ldots, m_{L-1}, pk_{Sig})$
Parse σ as a tuple (A, e, v) and return true if $Z \equiv A^e R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v \mod n$, $2^{l_e - 1} < e < 2^{l_e}$ and $m_i \in \pm\{0,1\}^{l_m}$ for $0 < i \leq L$ holds, else return false.

Commitments. A requirement for ZKPKs is a commitment scheme [DF02, FO97, Ped92], with which a user can commit to a value. A commitment can be seen as a digital analogue of a lock box having two important properties: i.e., *hiding* and *binding*. When a user commits to a value, the value is stored in the lock box, which is locked afterwards and given to a receiver. The value is hidden to the receiver (hiding) and the owner can not alter the value (binding). To state this more formally, let C be the commitment of a value v: $C = \mathrm{Comm}(v)$. Then hiding means that the recipient is not able to find v from C, while binding means that the owner cannot convince the recipient that $C = \mathrm{Comm}(v')$ when $v \neq v'$.

Both Pedersen commitments [Ped92] and the schemes proposed by Damgård et al. [DF02] and Fujisaki and Okamota [FO97] have the same construction. They only differ in the mathematical properties of the group parameters x, y, the modulus n, and the restrictions on r:

$$C = x^v y^r \mod n \ . \tag{8.1}$$

An essential feature is that varying r results in different values of C, hence contributing to the anonymity and unlinkability of the committer.

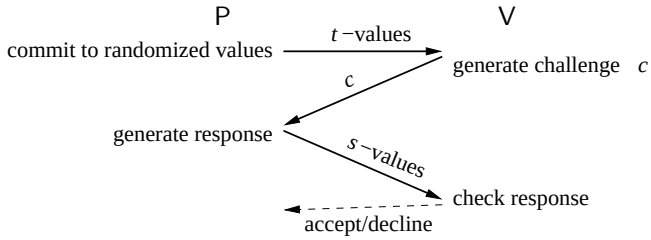


Figure 8.1: Representation of an interactive Σ-protocol.

Proofs-of-Knowledge. In short, we can state that a proof-of-knowledge is a process in which a prover interacts with a verifier, where the prover convinces the verifier that he knows a certain value (i.e., the statement). As the verifier learns nothing more than the fact that the statement is true, the term zero-knowledge proof-of-knowledge is used. Proofs-of-knowledge should carry certain properties:

- **Soundness:** A verifier will always accept the statement of an honest prover.
- **Completeness:** A cheating prover cannot convince a verifier that a statement is true.
- **Zero-knowledgeness** (extra property of ZKPKs): A (cheating) verifier learns nothing more than the truth of the statement.

Typically, the interactions between prover and verifier follow the same pattern, often referred to as Σ-protocols (Fig. 8.1). The protocol starts with the prover committing to a "randomized version" of the values he wants to prove knowledge of. These so-called t-values are sent to the verifier, who returns a challenge. This challenge is used by the prover to generate a response (the s-values). The verifier can use the response to complete the check of the proof and act accordingly.

As an example we recall the proof-of-knowledge of a discrete logarithm. Let $y = g^x \mod n$, x being the value we want to prove knowledge of and g, n public values. This proof-of-knowledge assumes that the discrete logarithm problem is hard (see p. 114).

1. The prover chooses a random value r_x and commits to this value: $t = g^{r_x} \mod n$. Then t is sent to the verifier.
2. The verifier chooses a random challenge c and sends that to the prover.
3. The prover computes the s-value $s_x = r_x - xc$ and sends this as a response the to verifier.
4. The verifier can check if $t \equiv y^c g^{s_x} \mod n$.

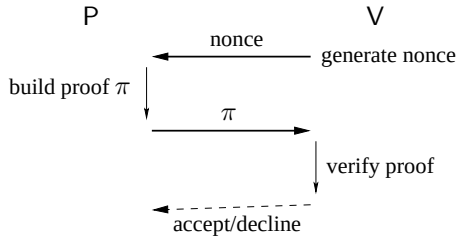


Figure 8.2: Representation of an non-interactive proof-of-knowledge.

We can see that if r_x has a sufficient length, s_x will in turn be close-to-random, hence hiding x to the verifier (and potential eavesdroppers). We will not delve deeper into the requirements, properties and lengths of all parameters used in the protocol. A table of lengths and constraints can be found in the `Identity Mixer` specification [17]. We use these values in all practical cases.

For more detailed information about different kinds of proofs, we refer to the literature. The proof shown above is described for groups of unknown order in [FO97, BCM05] and for known order in [Sch91]. How to combine both types of proofs is detailed in [CKY09]. Different variations also exist in the construction of the challenge. The challenge can be constructed by hashing the t-values, together with some common values and public information [FS87]. In that case the proof is called non-interactive i.e., it requires one transaction less (see Fig. 8.2). When the hash includes a message this is typically called a signature proof-of-knowledge.

More complex variations also exist: proving equality of two public keys with respect to different bases [CP93, CS97]; proving that a value lies in a certain interval [BCDvdG88, CFT98, Bou00]; proving of logical relations [CG08]; ...

8.4 Verification of CL-Based Credentials Proofs

Attribute-based credential authentication based on CL signatures (e.g., as in the `Identity Mixer` library), mainly consists of a signature proof-of-knowledge of a CL signature $\sigma = (A, e, v)$ on a nonce n_1. We will recall the protocol for proving knowledge of a valid credential and selective disclosure as implemented in the `Identity Mixer` specification [17]. However, more advanced protocols such as interval proofs and enumeration can also be found in the specification.

The specification also defines the lengths of the parameters used in the protocol. Here ℓ_e, ℓ_v, ℓ_m and ℓ_H are the bit length of e, v, the attributes, and the challenge respectively. The security parameter that governs the statistical zero-knowledgeness is ℓ_ϕ and $\ell_{e'}$ is the size of the interval where the e values are taken from. For the

exact lengths we refer the reader to the specification [17]. The sets A_r and A_h are the sets of revealed, respectively hidden attributes and m_0 is called the master secret. As mentioned before, m_0 is never revealed.

The signature proof-of-knowledge can be divided into two protocols i.e., the construction of the proof (or building of the proof) and the verification. Every practical implementation we present in this text is implemented according to the following description.

Construction of the proof. After the prover receives the nonce n_1 from the verifier, he builds the proof as follows:

1. Randomize CL Signature $\sigma = (A, e, v)$:
$$r_A \in_R \{0,1\}^{\ell_n + \ell_\phi}$$
$$A' = AS^{r_A} \mod n$$
$$v' = v - er_A$$
$$e' = e - 2^{\ell_e - 1} \ .$$
2. Compute 1st round:
$$\tilde{e} \in_R \pm\{0,1\}^{\ell_{e'} + \ell_\phi + \ell_H}$$
$$\tilde{v}' \in_R \pm\{0,1\}^{\ell_v + \ell_\phi + \ell_H}$$
$$\tilde{m}_i \in_R \{0,1\}^{\ell_m + \ell_\phi + \ell_H} \ \forall i \in A_h$$
$$\tilde{Z} = (A')^{\tilde{e}} \left(\prod_{i \in A_h} R_i^{\tilde{m}_i} \right) (S^{\tilde{v}'}) \mod n \ .$$
3. Compute challenge:
$$c = H(context \parallel A' \parallel \tilde{Z} \parallel n_1) \ .$$
4. Compute 2nd round:
$$\hat{e} = \tilde{e} + ce'$$
$$\hat{v}' = \tilde{v}' + cv'$$
$$\hat{m}_i = \tilde{m}_i + cm_i \ \forall i \in A_h \ .$$

Then the proof is $\pi = (c, A', \hat{e}, \hat{v}', \hat{m}_i, m_j \ \forall i \in A_h \text{ and } \forall j \in A_r)$. This is sent to the verifier.

Verification of the proof. When the verifier receives π, he can verify the proof as follows:

1. Compute:

$$\hat{Z} = \left(\frac{Z}{\prod_{j \in A_r} R_j^{m_j} (A')^{2^{l_e - 1}}} \right)^{-c} (A')^{\hat{e}} \left(\prod_{i \in A_h} R_i^{\hat{m}_i} \right) (S^{\hat{v}'}) \mod n \ . \tag{8.2}$$

2. Check whether:

$$\hat{m}_i \stackrel{?}{\in} \{0,1\}^{\ell_m + \ell_\phi + \ell_H + 1} \ \forall i \in A_h$$

$$\hat{e} \stackrel{?}{\in} \pm\{0,1\}^{\ell_{e'} + \ell_\phi + \ell_H + 1}$$

$$c \stackrel{?}{=} H(context \parallel A' \parallel \hat{Z} \parallel n_1) \ .$$

The proof is rejected if any of these checks fail.

We can see that Eqn. (8.2) requires at least four modular exponentiations (for a credential with only a master secret). It is clear that this requires a great deal of computational effort, especially on an embedded device.

8.5 Conclusions

In today's (mobile) embedded environments, attribute-based credentials systems can offer a more privacy-friendly solution in comparison to classic certificate technology when it comes to accessing electronic services. The downside is the increased computational effort, mainly the number of modular exponentiations required for the zero-knowledge proofs used with attribute-based credentials.

Several solutions and approaches have been proposed to decrease the run times of a credential verification. Unfortunately, all these solutions aim to optimize the performance on the side of the prover. However, there is a specific field of applications where verification is performed on embedded (computationally less powerful) devices. We feel that embedded support for faster – and possibly simultaneous – calculation of modular exponentiations may reduce the response times at the verifier, and consequently, increase the overall performance of an attribute based authentication procedure.

A great deal of this text will hence focus on the design of a hardware accelerator for modular exponentiations that is able to meet the requirements for attribute-based

credential applications. In a case study (Chapter 9) we will use this hardware accelerator with our embedded test platform (Chapter 2) and evaluate the effect on the run times of credential verification in an embedded context.

Chapter 9

Attribute-Based Credential Authentication with Embedded Devices

9.1 Introduction

With the aid of our platform (Chapter 2) we can now analyze attribute-based credential verification on an embedded terminal. More particularly, we will investigate an ABC application over NFC between a smartphone (acting as a prover) and an embedded terminal (the verifier). We are able to measure the run times of all aspects of the protocol such as communication and computation, for different modulus lengths and attributes, and in different scenarios i.e., computations in software, single-base exponentiation hardware offload, and dual-base simultaneous exponentiation offload. For this purpose the IP core we developed and the accompanying software API, will be used (Chapter 4). By looking at the relative shares of all parts of the protocol, we can show what can be gained by using some form of hardware offload for verification on an embedded platform. Because this platform implements a standard embedded design setup, conclusions drawn here can be generalized to non-FPGA-based systems.

We will start by explaining the details of the implementation we have used for the tests (Section 9.2). Then we will look at what tests we have performed and what were the results (Section 9.3). We will evaluate these results (Section 9.4) and state some conclusions (Section 9.5)

Publications. The results of this case study have been described in a paper, presented at the *14th Joint IFIP TC6 and TC11 Conference on Communications and Multimedia Security* in Magdeburg, Germany [OLN+13].

9.2 Test Setup and Implementation Detail

9.2.1 Platform Details

We recall some features of the test platform (Chapter 2) as well as the specifications of the smartphone, that are important for these tests.

- *Embedded processor.* The MicroBlaze 32-bit RISC processor runs embedded Linux. For the computation of modular inverses or (large) integer multiplications, we use the GMP arithmetic library [38]. All peripherals are connected to the processor via the PLB bus. The processor and bus operate at 100 MHz.
- *Cryptographic accelerators.* For the hashing we use the SHA-256 IP core present in the platform. For the modular arithmetic i.e., multiplication, single-base and dual-base simultaneous exponentiation, we use the IP core design described in Chapter 4. The core supports modulus lengths of 512, 1024 and 1536 bits and has 16-bit stages. Run times are given in Chapter 5 for multiplication and Chapter 6 for exponentiation. Both hardware accelerators run at 100 MHz.
- *NFC Communication.* An NXP PN532 development kit implements the RF front end for the NFC communication and is able to emulate different types of NFC devices such as, Mifare, ISO14443-A/B, DEP, Felica, etc. We configure the PN532 as an NFC initiator scanning for ISO14443-A tags. For this purpose we use `libnfc` (described in Sect. 7.5). Note that we do not use NFC P2P communication in this case study.
- *Smartphone.* The smartphone that computes the proof and sends it over NFC to the embedded terminal, is a Google Nexus S running the custom-built CyanogenMod 9.1. The phone operates in card emulation mode with a raw NFC data rate of 106 kbit/s.

9.2.2 Implementation Details

We recall Eqn. (8.2), which is to verify a CL-based credential proof.

$$\hat{Z} = \left(\frac{Z}{\prod_{j \in A_r} R_j^{m_j} (A')^{2^{l_e - 1}}} \right)^{-c} (A')^{\hat{e}} \left(\prod_{i \in A_h} R_i^{\hat{m}_i} \right) (S^{\hat{v}'}) \mod n \ .$$

To apply the hardware accelerator (in simultaneous mode), this equation is rearranged and split into a modular product of dual-base exponentiations as presented in Eqn. (9.1). The A' exponent $\left(c \cdot 2^{l_e - 1} + \hat{e}\right)$ as well as the modular inverse Z^{-1} are computed in software. All multiplications are computed in hardware. Note that in case of a credential with only the master secret m_0, only (a) and (b) need to be computed.

$$\hat{Z} = \underbrace{\left(Z^{-1}\right)^{c} \cdot R_0^{\hat{m}_0}}_{\text{(a)}} \cdot \underbrace{(A')^{\left(c \cdot 2^{l_e - 1} + \hat{e}\right)} \cdot S^{\hat{v}'}}_{\text{(b)}} \cdot \underbrace{\prod_{j \in A_r} R_j^{c m_j} \cdot \prod_{i \in A_h} R_i^{\hat{m}_i}}_{\text{attribute-dependent}} \mod n \ . \tag{9.1}$$

In Chapter 6 we have shown that the grouping of the exponents is important when performing simultaneous exponentiations. For a minimal run time it is important that the length of the exponents differs as little as possible. That is why the verification step is rearranged and computed as shown in Eqn. (9.1). For the attribute-based part, the revealed and hidden attributes are grouped in separate dual-base exponentiations where possible.

In Table 9.1(b) we have listed the lengths of the parameters used on the embedded platform. These are based on the lengths proposed in the `Identity Mixer` specification [17] (Table 9.1(a)). Due to the restriction that the exponent lengths need to be a multiple of 32 (see IP core design Chapter 4), some lengths have been increased to meet that requirement. To give an idea about the amount of data that is sent over NFC we have also added the lengths of some proofs. These do not have to be a multiple of 32. Also note that the length of the attributes has been fixed to 256 bits for a worst-case scenario.

Table 9.1: Lengths of system parameters used on the embedded platform [bits] as defined by the `Identity Mixer` specification. The lengths that have been increased to a multiple of 32 are indicated with a (*).

(a) System parameter specifications.

name	$\ell_n = 1024$	$\ell_n = 1536$
ℓ_ϕ*	96	96
ℓ_m	256	256
ℓ_v	2048	2220
$\ell_{e'}$	120	120
ℓ_H	256	256
ℓ_e	567	597

(b) Parameters used on the embedded platform.

parameter	defined by	$\ell_n = 1024$	$\ell_n = 1536$
c	ℓ_H	256	256
$\hat{e}$	$\ell_{e'} + \ell_\phi + \ell_H$	472	472
$\hat{v}'$	$\ell_v + \ell_\phi + \ell_H$	2400	2592*
$\hat{m}_0$, $\hat{m}_i$	$\ell_m + \ell_\phi + \ell_H$	608	608
m_j	ℓ_m	256	256
A'	ℓ_n	1024	1536
$\left(c \cdot 2^{l_e - 1} + \hat{e}\right)$		864	864
π (only m_0)	$\ell_n + \ell_{e'} + 3\ell_\phi + 4\ell_H + \ell_v + \ell_m$	4760	5464*
π (all hidden)	$... + 6(\ell_m + \ell_\phi + \ell_H)$	8408	9112
π (all revealed)	$... + 6\ell_m$	6296	7000

9.3 Tests and Results

9.3.1 Results in the Literature

As pointed out in Chapter 8, only little timing results on the verification of attribute-based credentials are available in the literature and they are often hard to compare. Nevertheless, Table 9.2 presents a comparison of our embedded terminal with comparable implementations available in the literature. Because of the different architectures, processor speeds, and implementations, it is hard to make a precise comparison. Still, the figures clearly show the value of our hardware accelerated solution with respect to general purpose devices. Note that the measurements by

Table 9.2: Timing results (in ms) for the verification of attribute-based credentials, with only a master secret, compared to prior work.

n	Bichsel [BCD+12]	Dietrich [Die10](DAA)			This setup
	Intel Core 2-Duo P9600 2.53 GHz	Intel Core 2-Duo T7500 2.2 GHz	P910 ARM9 156 MHz	Nokia 6131 ARM9 229 MHz	Virtex 6 MicroBlaze 100 MHz
1024	78	40	8240	16960	124
1536	187	-	-	-	215
2048	375	110	22100	64270	-

Dietrich [Die10] are performed based on the DAA verification protocol, which is closely related to the CL credential verification used in this paper.

9.3.2 High-Level Description of the Tests

As pointed out before, we are interested in the run time of the credential verification protocol on an embedded terminal. To that end, different cases have been evaluated on our test platform. With these test cases, we want to address several questions:

- What is the effect of hardware acceleration on the run time? We will compare an embedded software implementation –using GMP on the MicroBlaze CPU– with an implementation using hardware offload, both using single-base exponentiations and dual-base simultaneous exponentiations.
- What is the effect of the number of attributes in the credentials on the verification run time?
- What is the effect of the number of hidden/revealed attributes? (The master secret (m_0) is always hidden.)
- To compare the verification run time with the total protocol run time, we will also take into account the communication overhead.

9.3.3 Verification Performance

As a first test, the verification of a credential with a single hidden attribute (i.e., the master secret m_0) is evaluated. Table 9.3 presents the run time of an embedded software implementation (SW), compared with an implementation using

Table 9.3: Comparison of the average timing results for the credential proof verification on an embedded platform, where all computations are performed in software (*SW*), and where a hardware accelerator is used for single-base *1-exp* and dual-base *2-exp* exponentiations. The credential contains a single hidden attribute (m_0, the master secret).

		NFC Com.		**Verification**		**Hash and check**		**Total**
n	case	[ms]	[%]	[ms]	[%]	[ms]	[%]	[ms]
1024	*SW*	748	11.6	5696	88 .3	7	0.1	6451
	1-exp	733	81.5	159	17.7	7	0.8	899
	2-exp	721	84.6	124	14.6	7	0.8	852
1536	*SW*	765	5.2	13846	94.7	9	0.1	14620
	1-exp	782	75.8	240	23.3	9	0.9	1031
	2-exp	768	79.3	191	19.8	9	0.9	968

the hardware accelerator; both with single-base (*1-exp*) and dual-base simultaneous exponentiations (*2-exp*). The table also shows the NFC communication time (i.e., the time required to transmit the proof π) as part of the overall protocol run time.

As expected, the verification with embedded software takes significantly longer than the hardware-accelerated implementations. For a modulus of 1536 bits, it takes about 14 seconds or 95% of the total run time. The hardware accelerator clearly improves the performance of the verification and hence the overall run time. Moreover, the main share of the run time shifts towards the communication over NFC. The table also shows that the communication speed is not constant for the three implementations (for the same modulus length) and this even though the results are averages. We believe that this is mainly due to overhead introduced in the operating system; keep in mind that Linux is not a real-time operating system.

The tests also illustrate that a different size of the modulus has a much greater effect (relatively) on the verification time than on the communication time. This is the case for all three implementations. As could be expected, the time for hashing and making the necessary checks is negligible with respect to the time required for running the verification protocol.

9.3.4 Influence of the Number of Attributes

As a second case, we have examined the effect of the number of attributes in the credential, where all attributes remain hidden. Fig. 9.1 shows the run time for the communication (a) and the verification (b) with respect to the number of

attributes in the credential. Fig. 9.1(c) also presents the speedup of the run time when using dual-base exponentiations instead of single-base exponentiations, as defined in Eqn. (6.8).

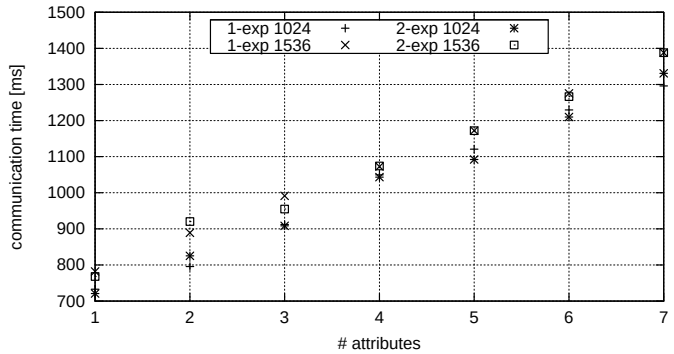
Increasing the number of attributes increases both the time required for communication and verification. This is obvious, as the proof π will also be larger. If a single-base exponentiation accelerator is used, the verification time increases linearly with the number of attributes. However, if dual-base exponentiations are used, the verification time increases stepwise. This is because the time for computing a dual-base exponentiation is comparable to a single-base exponentiation. Hence, based on Eqn. 9.1, when verifying a credential with an odd number of attributes (master secret included), our dual-base exponentiations can be used to their full extent. In contrast, in the case of an even number of attributes, one extra single-base exponentiation is required. In other words, increasing the number of attributes from an even number to an odd number results in replacing a single-base exponentiation by a dual-base exponentiation, which requires only a small overhead. Increasing the number of attributes from an odd number to an even number results in having an additional single-base exponentiation.

As can be seen, using dual-base exponentiations instead of single-base exponentiations results in a verification speedup when verifying more attributes (e.g., a speedup of about 1.4 for a credential with 7 attributes). As a previous example (Chapter 6, outline *A realistic example*) we calculated the verification speedup to be 1.28 (in the case where we have only a master secret). As we can see in Fig. 9.1(c) this is indeed the case (not exactly, but close enough). The speedup is roughly the same for both 1024-bit and 1536-bit moduli. This is to be expected because we have seen that the time to compute a multiplication (defined by the modulus length) has no influence on the speedup. However, the fact that in case of a 1536-bit modulus (1) the length of $\hat{v}'$ is longer and (2) computing a modular inverse takes longer, can explain the difference. The stepped behavior of the dual-base exponentiation time combined with the linear increase for single-base exponentiations, results in a saw-tooth behavior for the speedup.

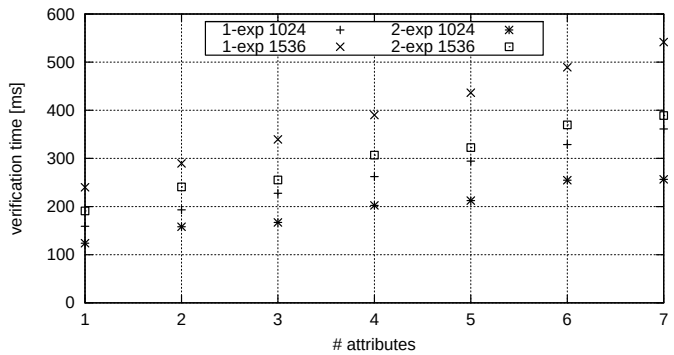
9.3.5 Influence of the Number of Revealed Attributes

As a last test, we have looked at the influence of the number of revealed attributes on the run time. The number of revealed attributes varies from 0 to 6 (i.e., the master secret is never revealed). The communication time and verification run time is set out as well as the verification speedup when using dual-base exponentiations (Fig. 9.2).

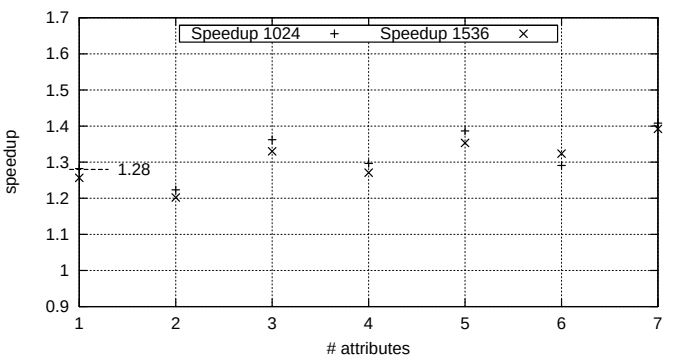
Both the communication time and verification time decrease with an increasing number of revealed attributes. This is because the exponent m_j of a revealed attribute is shorter than the exponent $\hat{m}_i$ of a hidden attribute. Again, there is a



(a) Communication



(b) Verification



(c) Verification speedup

Figure 9.1: Influence of the number of attributes on the run time. All attributes remain hidden.

linear relationship when single exponentiations are used and a stepwise behavior for dual-base exponentiations. Note that for our tests the size of the attribute values was 256 bits, while in reality this will often be much smaller. In fact, an attribute representing, for instance, the user's gender, could require only a single bit. Hence, the decrease in time when releasing more attributes would become even more significant.

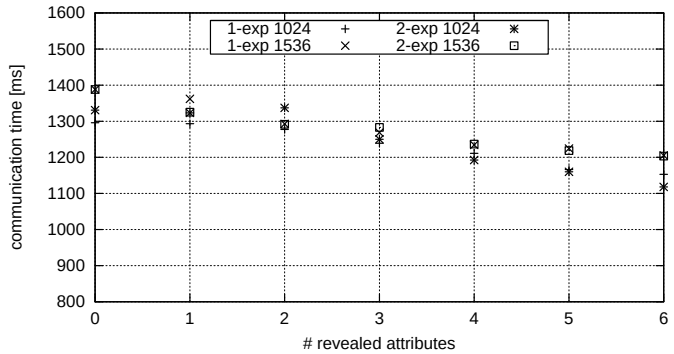
9.4 Evaluation

Run time. It is clear that for embedded applications the use of a hardware accelerator for credential verification greatly increases the feasibility in terms of run time. Compared to a powerful back end as is used in [BCD+12, Die10], the run times in this embedded context are of the same order of magnitude. Thus, applications such as physical access control (e.g., opening a locker) or stand-alone vending machines that support privacy, anonymity and/or unlinkability become feasible.

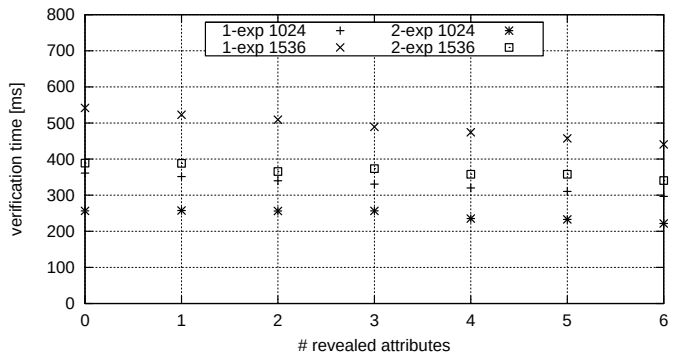
The efficiency of the simultaneous hardware accelerator is maximal in the case of exponents of the same length. However, in this case, the difference in length of the exponents in dual-base exponentiation (b) of Eqn. (9.1), is significant (cf. Table 9.1). A solution is to split exponentiation $S^{\hat{v}'}$ into a dual-base exponentiation $S_1^{\hat{v_1}'}.S_2^{\hat{v_2}'}$ with smaller exponents (see [BCC04, BCGS09] for more details). This, however, may require a slight modification of the prover protocol.

Communication. The scenario presented here, uses NFC for communication. It is clear that the communication results in a great deal of overhead, at least 70% for all of the examined cases (see Fig. 9.3). This implies that the overall speedup of the protocol run time by using dual-base simultaneous exponentiations is less than the verification speedup. The figures in this case study where obtained using NFC @ 106 kbit/s. Clearly, faster communication (e.g., NFC @ 424 kbit/s or Bluetooth v2.1 @ 3 Mbit/s) makes the use of a simultaneous exponentiation accelerator more interesting. We have to note that the communication has been kept to a bare minimum i.e., only the proof is sent. We assume the issuer public key is known by the platform.

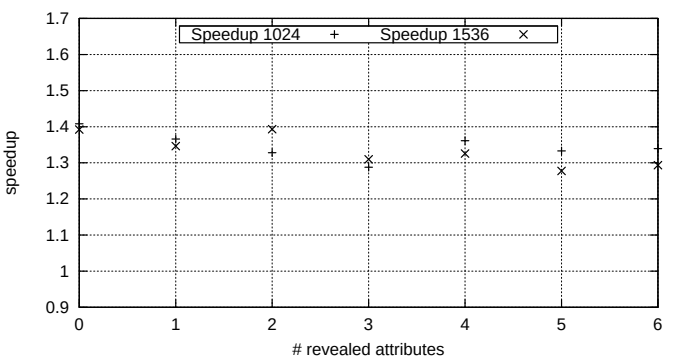
Keep in mind that, although NFC P2P operates at 424 kbit/s, it also introduces two extra layers of communication overhead. Also when using communication handover to another technology like e.g., Bluetooth or WiFi, the connection handover time needs to be taken into account. In Chapter 10 we examine communication handover in the case of WiFi.



(a) Communication



(b) Verification



(c) Verification speedup

Figure 9.2: Influence of the number of revealed attributes on the run time. The credential contains 7 attributes (master secret included).

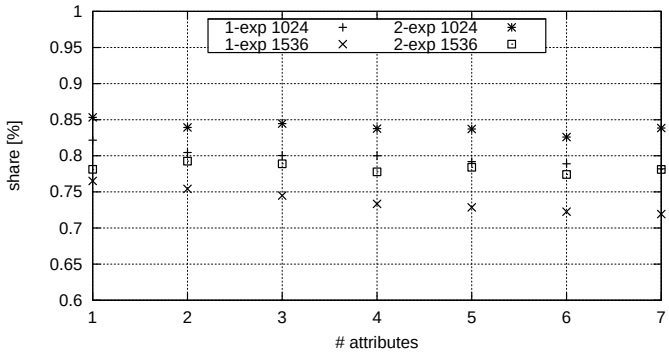


Figure 9.3: The relative share of the NFC communication in our test setup for different numbers of attributes.

Applicability. In this scenario, the complete embedded terminal has been implemented on an FPGA. For several applications where form factor or energy consumption are key requirements, this is undesirable. However, the accelerator could be implemented on an ASIC, which is better suited for commercial applications. The advantages demonstrated with our prototype implementation remain valid in these settings as well.

The current hardware accelerator is not tamper-proof and resistant against side-channel attacks. For applications that only require credential verification, this is not problematic. If, however, disclosed information must not be learned by other parties than the verifier, extra measures should be taken. For instance, to prevent timing analysis (which may reveal for instance if more or less attributes are disclosed), exponentiations should be performed with a constant timing; more specifically with the worst case timing: $(2 \cdot w_0 \cdot t_{\text{mult}})$. Obviously this entails a decrease in performance. On the other hand, this timing is the same for both single-base and l-base exponentiations. Hence, multi-base exponentiations become even more attractive in this setup.

In this case study, we only examined the speedup of the verification of attribute-based credential proofs on embedded platforms. Note that for a complete application, certificate revocation should also be supported.

9.5 Conclusions

In this chapter, we have presented a case study in which we use our embedded test platform for an ABC application i.e., a smartphone acting as a prover sends the proof over NFC to an embedded verifier (our test platform). In particular we have

used a hardware accelerator for modular exponentiations, in order to reduce the run time of applications that require attribute-based credential verification in an embedded context.

A first remarkable fact is that the communication is responsible for the largest share of the overall run time. This raises questions about the applicability of NFC as a medium for this type of applications. However, this requires more testing.

A second conclusion is that using hardware offload greatly increases the feasibility in terms of run time of ABC applications in an embedded context e.g., access control, vending machines, etc. In addition, the use of dual-base (simultaneous) exponentiation hardware may further increase the overall performance. This is especially the case when the overhead caused by communication can be decreased and/or a large number of attributes are included in the credential.

A logical target for future work, is the communication speed of NFC. The efficiency of communication handover (e.g., from NFC to Bluetooth) could be studied, as well as the performance of NFC P2P (operating at 424 kbit/s). These communication issues are partly tackled in Chapter 10 where we present our case study on communication handover to WiFi.

Another path is the support for more complex credential proofs such as range proofs and credential revocation. Future work could be directed to find out the impact of these additional features on the performance of the verification and the requirements of the embedded platform (e.g., connectivity with a revocation server). Also the effect of the length of the attributes (now fixed at 256 bits) on the verification should be further investigated.

To conclude we state the interesting fact that, by using the libraries installed on the platform (like `GMP` and `libnfc`) and by using the developed IP core and accompanying API, the development of this test case application only took one day.

Chapter 10

NFC P2P versus NFC handover to WiFi

10.1 Introduction

As we have seen in the previous case study (Chapter 9), sending the proof required for CL-based credential verification over NFC takes up a considerable amount of time when the phone is operating in card emulation mode.[1] For example, sending the proof for a credential containing only a single attribute takes between 700 and 800 milliseconds. For a credential with 7 attributes of which one is revealed, it takes about 1300 milliseconds.

For NFC P2P communication, a raw data rate of 424 kbit/s is used. This is 4 times as fast as in card emulation mode. On the other hand two extra layers (LLCP and SNEP) each introduce a certain amount of overhead, which in turn reduces the effective data rate. Unfortunately, only the raw data rates are specified (in the specification as well as in the literature). However, when considering several communication options, it is imperative that designers know the effective data rates. We will therefore investigate which data rates can be achieved by using NFC P2P communication and relate the newly obtained figures to an ABC scenario.

We will compare the effective data rate for NFC P2P communication with a case where the NFC connection is used to initiate a communication handover to a faster wireless protocol. All tests have been performed with our embedded test platform using the libraries for NFC P2P communication.

[1] In that case the raw data rate of NFC is 106 kbit/s.

10.2 NFC P2P

10.2.1 Test Setup

To be able to measure the data rate of an NFC P2P connection between an embedded platform and a mobile device we use our embedded test platform with the NFC P2P stack on one end. An Android smartphone with a custom NFC service acts as the other end of the connection. The custom service is implemented using JNI (see Section 7.4.3). As we explained in Chapter 7 this is necessary because current Android builds do not implement the full NFC P2P stack.

We also made some simplifications to the communication itself. The first one is that we exchange data over a connection that has already been established. This way we eliminate the connection setup time in our data rate measurements.[2] The second simplification is that only short NDEF records are being used. The main reason is that the underlying hardware has a maximum payload length and thus normal NDEF records would have to be parsed over several LLCP messages anyway.

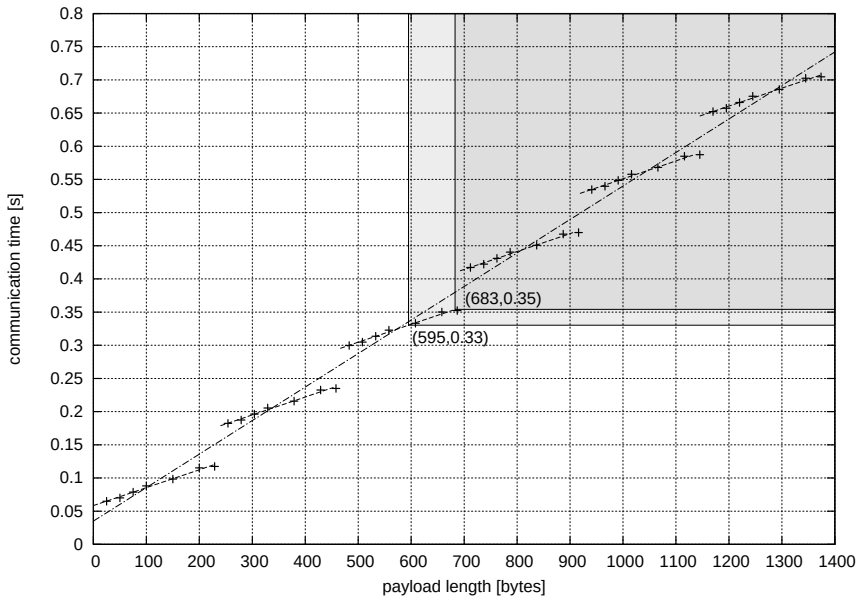
As SNEP is a request/response protocol: a new message will only be sent when a *SUCCESS* response on a previous message has been received. We will use this mechanism to measure the time needed to send a certain number of bytes from the platform to the smartphone. At the moment when our embedded platform sends a SNEP *PUT* request, a time stamp is taken. The SNEP *PUT* message contains an NDEF record which in turn contains the actual payload. When the smartphone receives this message successfully, it will reply with a SNEP *SUCCESS* response. Upon receipt of this message at the embedded platform, a second time stamp is taken to measure the time needed to send a message with a certain payload size. Because the NFC stack on the embedded platform is influenced by the load on the OS, communication times will vary (several ms). To obtain an averaged result, we performed 10 measurements for every payload size.[3]

10.2.2 Results

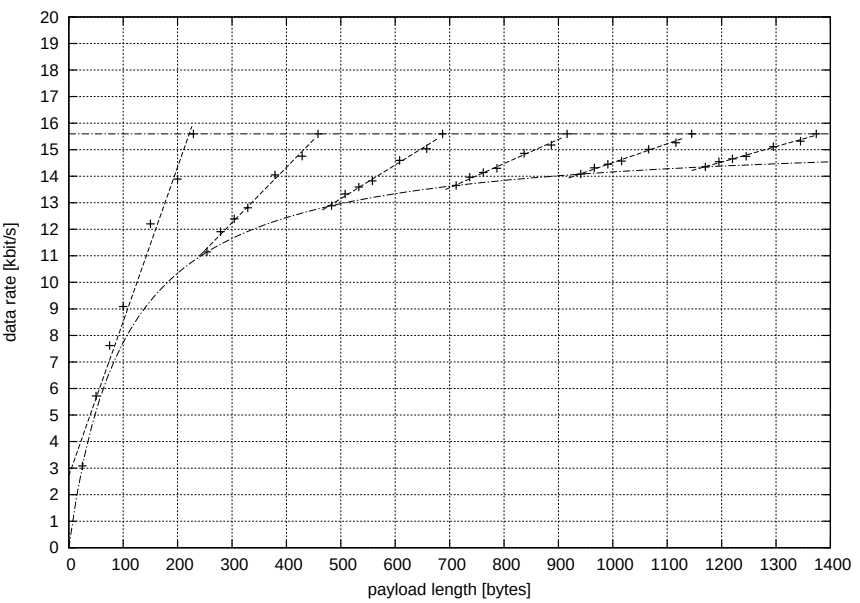
We have set out the average time required to send an NFC P2P message (i.e., an NDEF record contained in a SNEP message) for different payload sizes in Fig. 10.1(a). By dividing the payload size by the time to send the data, we have obtained the effective data rate. This is set out in Fig. 10.1(b).

[2] Typically connection setup times for NFC are under 100 ms.

[3] We limited ourselves to 10 measurements per payload, because the measurement itself is a tedious procedure that involves resetting the Android application and the embedded NFC stack, as well as "touching" the platform with the smartphone.



(a) Time required to send a number of bytes over an NFC P2P link (in one direction).



(b) Effective data rate for NFC P2P in function of the payload length.

Figure 10.1: Speed measurements for NFC P2P communication.

A first remarkable observation is that the time increases linearly in intervals of 229 bytes. Whenever the payload size exceeds a multiple of 229, an extra overhead is introduced. The reason for this is that 229 bytes is the maximum payload size that can be sent with one packet by the underlying layers. This means that when exceeding a payload size of 229 bytes (or a multiple) an extra message needs to be sent, which introduces overhead in the form of headers and the processing of intermediate SNEP *CONTINUE* responses.

This obviously has its influence on the data rate as well. Only for payloads above 1024 bytes, the data rate more or less stabilizes at about 15 kbit/s, with a maximum of 15.6 kbit/s. This is only 3.7% of the physical data rate of 424 kbit/s.

Applicability in ABCs. If we look at a CL-based credential proof for a modulus size of 1024 bits and a single attribute, the size of the proof π will be 4760 bits or 595 bytes (Table 9.1(b)). For card emulation (operating at 106 kbit/s), sending the proof takes between 720 and 750 ms (see Table 9.3). When NFC P2P communication would be used, it would take about 330 ms. This has been indicated on the graph (Fig. 10.1(a)) together with the payload size and communication time for a 1536-bit modulus. In the case of CL-based credential verification over NFC, these are the absolute minimum communication times. When either the modulus or number of attributes becomes larger, the communication time will increase.

In general it is also interesting to note that, although the data rate for card emulation is four times lower that the one used for NFC P2P, the latter is only about twice as fast.

10.3 Connection Handover to WiFi

10.3.1 Test Setup

An alternative to NFC P2P communication is using NFC to set up a connection over a different wireless standard. In this case we will investigate connection handover to WiFi (TCP/IP). For that, we use the setup as shown in Fig. 10.2. When the smartphone is brought in the vicinity of the embedded platform an NFC connection will be set up. Over this connection, a message containing handover information is sent to the smartphone after which the connection is closed. The handover data contains information such as the platform's IP address and the TCP port to connect to.

The timing measurements have been performed with the smartphone already connected to the WLAN. This eliminates variations due to the wireless router's response times. The time to handover the connection has been measured as time

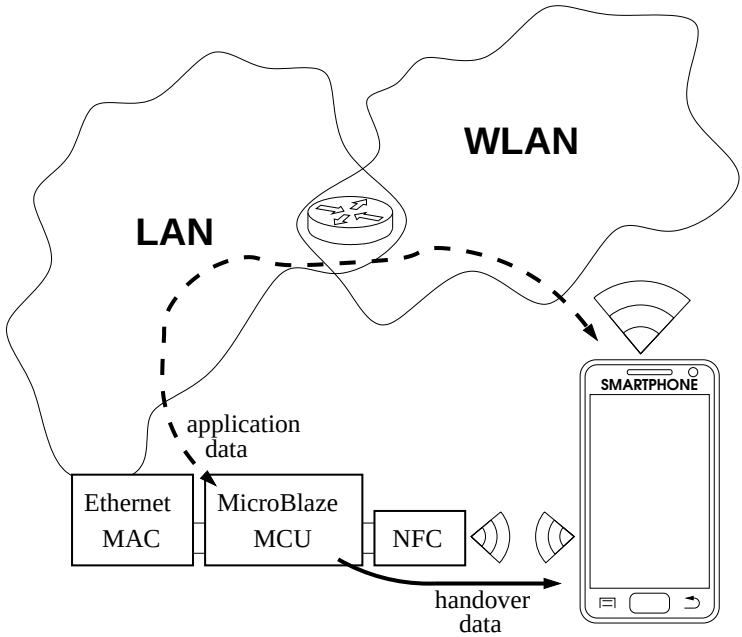


Figure 10.2: Setup for testing communication handover from NFC to WiFi.

difference between sending the handover data and accepting the TCP connection. As a practical case we have transmitted a credential proof over TCP/IP to the embedded platform (1536-bit modulus, only master secret). We have measured this transmission time as well. The connection handover with TCP/IP communication has been repeated 40 times.

10.3.2 Results

We have generated a histogram for the handover timing, the time required for TCP/IP data transmission, and the total communication timing (handover and data transmission). This has been set out in Fig. 10.3. It is clear that the communication handover is responsible for a large portion of the total communication time. Moreover, it is impossible to do the handover under 0.4 seconds. This is more than the 350 ms for the total communication when NFC P2P is used. The TCP/IP data transmission on the other hand clearly outperforms the NFC P2P communication. This shows that communication handover only pays off when the amount of data to be sent is large enough. More detailed measurements, however, are required to determine the exact tipping point. Specifically for attribute-based credentials

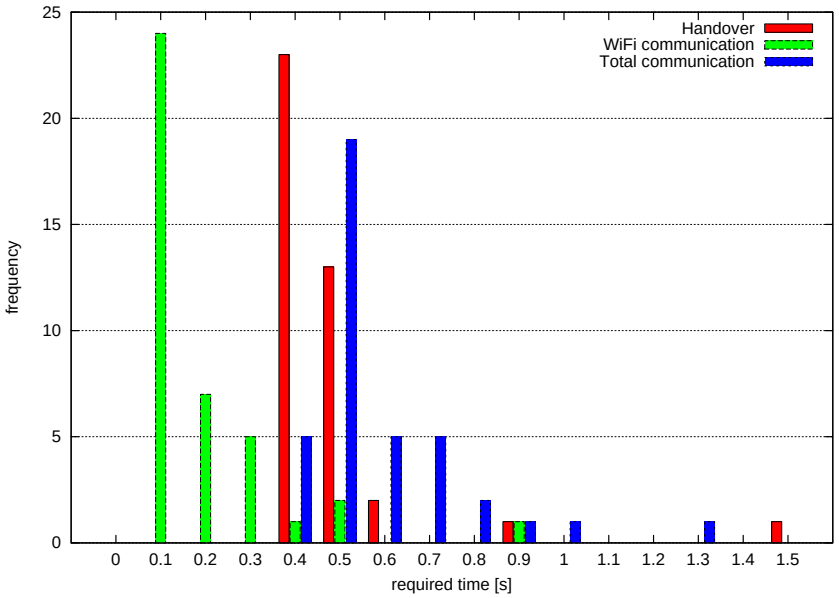


Figure 10.3: Communication handover timing.

it should be investigated if it pays off to use handover, when more attributes are being used.

10.4 Conclusions

As a first practical result, we have been able to determine the effective data rate for NFC P2P communication. These are likely the first figures published regarding this subject. However, more measurements are required to determine the effect of the connection set up and the effect of transmitting application data in the other direction as well. Furthermore, when an new Android release implements NFC P2P (according to the specification), it would be interesting to repeat the measurements using this Android version and compare the results with our measurements.

A second conclusion is that NFC P2P is a valid candidate as communication standard for ABC applications. Newer versions of the standard (at 848 kbit/s and higher) will result in even shorter communication times. For protocols that require more data transmission, communication handover could be used. However, more detailed measurements are required to determine the exact tipping point.

We must note that we have only measured the time of setting up a TCP connection

and not the time required to connect to a WiFi network. This should deserve some further attention. Other future work could be targeted to comparing connection handover to other technologies such as Bluetooth or GPRS.

Chapter 11

General Conclusions

11.1 Summary

The motivation of this work has been based on some emerging trends. That is, smartphones will play an increasingly prominent role in people's lives. They will be used to gain access to all kinds of services, both online and at designated terminals. These terminals (e.g., vending machines, electronic storage lockers or access control terminals) can be considered resource-limited in comparison to the mobile device. They typically have less memory, a limited user interface, less communication interfaces and run at lower frequencies.

When using the smartphone to obtain a certain service, the time to execute an authentication, payment, or even a simple data exchange, is a key factor. Think of a ticketing application in the metro, where it is not desirable that users have to wait a few seconds before their ticket is verified. That would be a downturn of the current situation. For obtaining these services in a privacy-friendly manner, attribute-based credential systems can offer a solution.

Designing these dedicated embedded terminals requires specific knowledge and tools. Furthermore, when evaluating the performance of (new) security protocols like attribute-based credentials on such a terminal, both the run time of the computations and the time required for communication need to be considered.

With that in mind we have directed our work along three tracks. First of all we have designed a configurable and extendable embedded platform to aid engineers in prototyping and evaluating security protocols on embedded terminals. Secondly, we have designed a customizable hardware accelerator for supporting attribute-based credentials on this (and other) embedded platform(s). A third track was the study

of NFC, which is an emerging standard for short-range wireless communication, for bidirectional communication between a mobile device and an embedded terminal. Combining these three tracks into two practical cases has been the final part of this work.

We restate our conclusions for each of the main parts in the following paragraphs.

11.2 Embedded Test Platform

Realization. Starting from an existing Xilinx FPGA development board, we have designed an embedded hardware platform for prototyping and evaluating security protocols that operate in and embedded context. This platform resembles a typical embedded design (whether it is e.g., a SoC or FPGA-based) in that it consists of a central processor with an operating system and peripherals connected to the processor via a bus. With this platform, design engineers can work on the different levels of embedded system design.

It is possible to develop and test custom hardware blocks. We have shown this with the design of our own IP core (Chapters 3 to 6 of this thesis). These and licensed cores can be easily added to the system by connecting them as memory-mapped devices to the system's bus. Interfacing with these hardware blocks is facilitated and generalized by using the UIO kernel module, present in the OS.

Aside from custom peripherals the platform already offers a wide range of functionality required in typical embedded systems such as timers, GPIO, memory controllers, and communication interfaces.

For the OS we have opted to use embedded Linux. A main advantage is that standard libraries can be used for large integer arithmetic, file I/O, threading, manipulation of communication sockets. Also, a program (or a part of it) can be tested and developed on a regular workstation.

To further facilitate application development we added several extra features to both the standard platform and the standard Linux build. This includes a boot sequence tailored to speed up development, support for Compact Flash, NFS and NTP, startup scripts to automatically enable this additional functionality, as well as hardware and libraries for NFC P2P communication.

We have shown the value of this platform with the development and analysis of a custom IP core as well as with two case studies regarding attribute-based credentials and NFC communication. By using the developed libraries (APIs) a demonstrator application for attribute-based credentials has been developed in mere days.

Proposals for future work. Currently, only one wireless interface has been implemented (i.e., NFC). Future improvements to this platform should definitely be targeted to adding other much-used embedded wireless standards e.g., Bluetooth, Bluetooth 4.0, GPRS. For off-line platforms, it would be advised to add a real-time clock IC e.g., connected with I^2C or SPI, for keeping track of the time. Adding hardware support for other cryptographic protocols would further increase the applicability of the platform.

Publications. The design and applications of this platform have been presented in several articles at international conferences. A first version, with the motivation of the design choices, has been presented at the *12th Joint IFIP TC6 and TC11 Conference on Communications and Multimedia Security* [OMS+11]. It has received the award for *Best short paper*. A more advanced version of the platform with extended NFC support, together with a first demonstrator application, has been presented at the *XXI International Scientific Conference on Electronics* [ODW+12].

11.3 Modular Exponentiation IP Core Design

Realization. As part of our embedded test platform, we have designed a hardware accelerator targeted to speed up the multiple modular exponentiations required for attribute-based credential verification. Because the IP core is intended for prototyping and evaluation, it had to be customizable to the developer's needs. To keep the core as versatile as possible we have made a design that does not require complex control logic (e.g., for recoding operands) and that is not based on certain assumptions (e.g., the structure of the modulus). A FIFO buffer to store the exponents ensures that different exponent lengths can be processed at run time. This is a necessity for e.g., attribute-based credentials.

A systolic pipelined multiplier with adjustable stage length as kernel of the design, ensures that the IP core can be synthesized for a given multiplication run time while requiring an acceptable amount of resources (i.e., same order of magnitude as multipliers tailored to certain specifications) regardless of the length of the operands. Furthermore, we have provided expressions to determine the number of resources required for a given stage length and operand length.

The multiplier is used to perform both the squaring and multiplication step of the modular exponentiation algorithm. Because we target embedded systems, we have explicitly chosen not to perform the squaring and multiplying in parallel as this would lead to a doubling of the required resources. The execution of the algorithm's main loop is controlled by the IP core's control logic. When the necessary operands are precomputed and stored in the correct memory locations, no time-consuming bus traffic is required during the automatic operation. This greatly reduces the

run time, which is now only determined by the length of the exponents and the configuration of the pipeline.

With this IP core we are also able to execute simultaneous dual-base and single-base exponentiations both under the same conditions. We have provided expressions for the multiplication and exponentiation run times as well as for the speedup by using simultaneous exponentiations. With the IP core as part of our embedded test platform we have verified these expressions based on measurements.

We found that in practice, run times are slightly higher because there is an overhead introduced by the latency in the OS and the communication with the IP core. This overhead is constant regardless of the length of the exponents and modulus, but it is dependent on the system setup e.g., clock frequency, CPU tics,... However, the overhead is very small with respect to the actual run time, when the exponents are sufficiently large, which is always the case in practice.

Regarding the speedup when using simultaneous exponentiations, we have shown that the fastest run times for dual-base exponentiations are achieved when the exponent length difference is minimal. When the exponents differ in length, the speedup decreases. However, a developer can learn this speedup beforehand (and make the trade-off) by using the expressions we have provided.

In order to make this design available for developers, we have published the VHDL code as well as the libraries with the driver source and API. It is device-proven on both Altera and Xilinx FPGAs.

Proposals for future work. Future updates could include other bus interfaces (e.g., Whishbone), side-channel resistance measures or support for different kinds of adders (e.g., carry-select to find a better trade-off between resources and speed). Also a JTAG interface [19] or some kind of self-test ability could be a useful addition.

In addition, we might investigate how this IP core can be used to speedup other protocols as well, e.g., DSA verification, DAA or secret sharing protocols.

Finally, we have also proposed a redesign of the IP core's operand memory so l-base simultaneous exponentiations can be implemented without the core's internal operand memory becoming too large. It requires the presence of shared memory and a DMA controller or a master/slave bus with a bus arbiter.

Publications. A first design of the pipelined Montgomery multiplier has been presented at the *8th Workshop on RFID Security and Privacy 2012* [Ott12]. The complete IP core design has been presented at the *9th International Symposium on Applied Reconfigurable Computing* [OPGS13]. The performance of the IP

core as part of our embedded test platform has been presented in an article in *EDN* [OPS+13].

11.4 NFC P2P Communication

Realization. We have investigated the value of NFC for exchanging messages as part of a cryptographic protocol between an Android device and an embedded platform. NFC has its specific place among other short-range wireless technologies i.e., short connection setup time (in a "touch"), but at low data rates. Like most wireless technologies, it is susceptible to denial-of-service attacks and eavesdropping. A relay attack is also easy to stage. If developers take precautions, these attacks can be countered or detected.

Another main problem, however, is related to the current state of the technology. The NFC P2P specification, defined by the NFC forum, is the standard for bidirectional communication between two NFC devices. Unfortunately, the current version of Android implements this standard only partially. We have compared and implemented several alternatives. Although functional, these solutions are either complex and/or require root access to the phone. Only communication handover to another wireless standard, seems to be a valid alternative at the moment.

We have added support for NFC P2P communication to our embedded test platform in the form of two open source libraries: `libnfc` and `libnfc-llcp`. We have also created our own basic design of SNEP and provided an example application for sending and receiving data over NFC P2P on an embedded platform.

Proposals for future work. The main drawback of our current (basic) SNEP implementation is that only short NDEF records are supported and that the user is responsible for parsing long payloads over multiple records. Obviously, this could be extended. For use across different applications, it would also be preferable that the SNEP layer be compiled into a library (just like the NFCIP-1 and LLCP layers).

11.5 Case Studies

Attribute-based credential verification. We have examined the scenario where a smartphone acts as a prover, sending the proof over NFC to our embedded test platform, which then performs the verification using the hardware accelerator for modular exponentiations. Hardware acceleration is a necessity on this kind of systems because a software implementation would lead to unacceptable run times.

However, when offloading the exponentiations to hardware, the communication now accounts for the bulk of the total protocol run time; at least 70% in this specific case. Multi-base (simultaneous) exponentiation hardware may further increase the overall performance. We have verified this for dual-base exponentiations.

This case study shows that attribute-based credential applications in an embedded context are feasible when using hardware acceleration. However, while much research has been (justly) targeted to speeding up the arithmetic, it turns out that communication (especially with standards with a low data rate like NFC) takes up a considerable amount of the overall run time as well.

Publications. The results of this case study have been described in a paper, presented at the *14th Joint IFIP TC6 and TC11 Conference on Communications and Multimedia Security* [OLN+13].

NFC peer-to-peer and connection handover. With our embedded test platform we were able to measure practical data rates for NFC P2P communication. The maximum data rate of 15 kbit/s is only a few percent of the advertised physical data rate (424 kbit/s). Connection setup times as well as bidirectional data traffic will probably decrease the data rate even further. On the other hand, as a communication medium in embedded attribute-based credential applications, NFC P2P seems to be a valid candidate.

For protocols that require the transmission of a significant amount of data, communication handover could be used. We have observed that a handover takes at least 0.4 s. However, more detailed measurements are required to determine the conditions for which handover to WiFi is beneficial with regard to the total communication time.

In embedded mobile applications, connection handover to Bluetooth or GPRS might also be useful or desirable. Future work could be targeted to investigating handover to these technologies as well.

11.6 Concluding Remarks

It is clear that the mobile revolution that is happening today, brings with it the need for a well-designed security infrastructure. This infrastructure has components at every level, each with its specific problems and difficulties. Embedded system design always requires making trade-offs. For relatively new techniques such as attribute-based credentials in an embedded context,[1] making design choices is extra hard due to the unknown performance of these techniques and the communication standards they rely on. Users (and companies alike) will only be able to benefit from these new technologies when companies, consortia and even the government work together to standardize, maintain and implement these measures. Our work can help developers in bringing these techniques into our everyday lives.

[1] These are relatively new at least to the non-academic world.

Appendix A

Systolic Pipeline Configuration Examples

In this appendix we give two examples of how the multiplier pipeline can be configured (before synthesis). Figures A.1 and A.2 show two alternative configurations. When only one operand length is required at run time one can use a single pipeline as shown in Fig. A.1. When several operand lengths will be used, one can use the pipeline configuration as shown in Fig. A.2. Instead of one large pipeline, two smaller pipelines can be used (only one at a time), which reduces the multiplication time in case of shorter operands. The two smaller pipelines, however, can be combined to form one large pipeline.

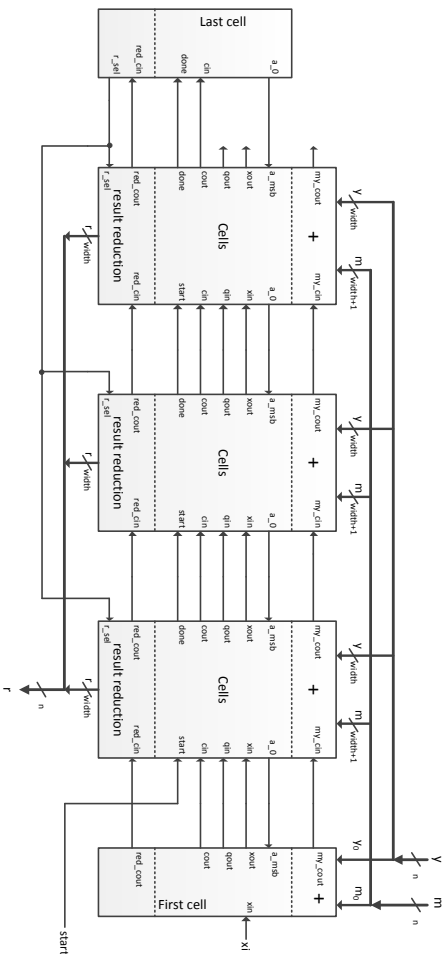


Figure A.1: Example of a 3-stage pipeline.

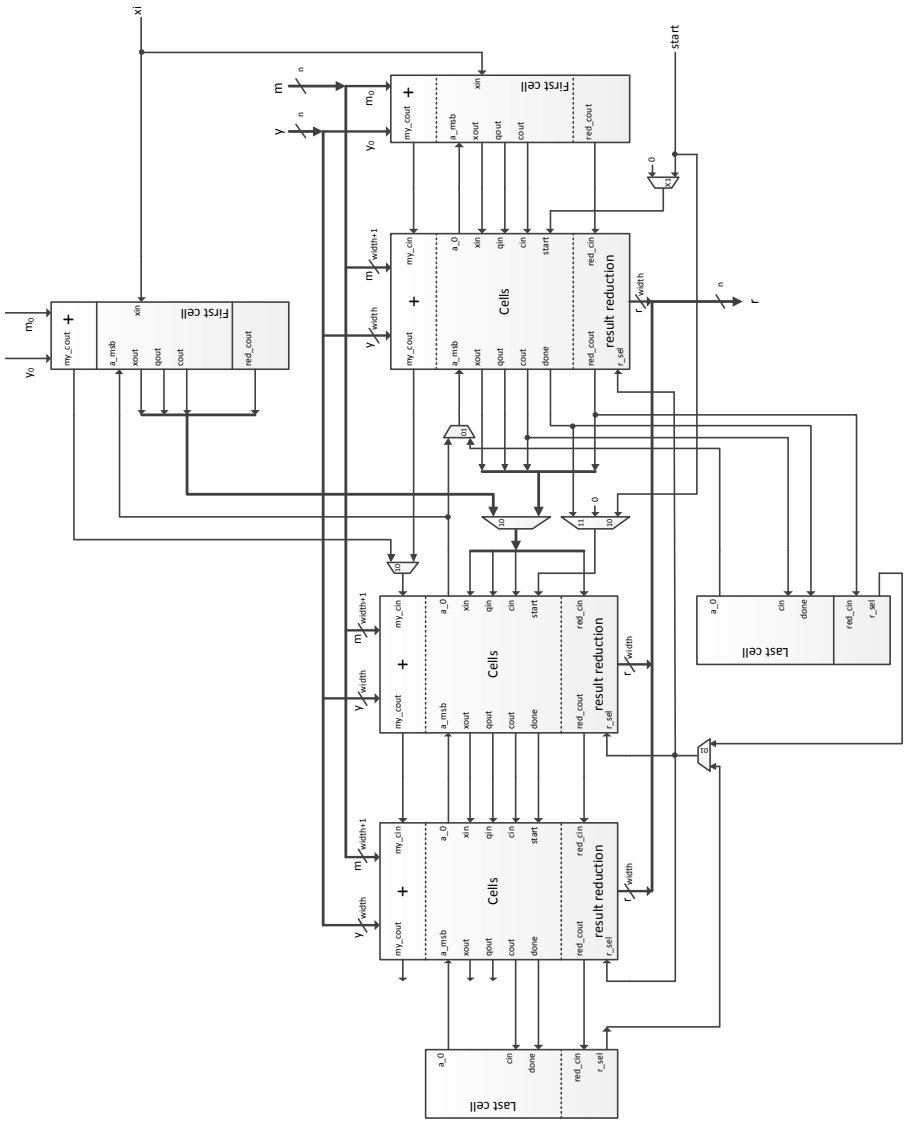


Figure A.2: Example of a split 3-stage pipeline (1 + 2 stages).

Appendix B

SNEP Server Application on an Embedded Terminal – Coding Example

This appendix gives a coding example on how to run a SNEP server application on an embedded terminal using our own SNEP implementation and the `libnfc` and `libnfc-llcp` libraries.

To understand how we structured the SNEP server, we will briefly explain how two services can communicate over the LLCP layer. The LLCP layer will keep track of which services are available and what is their respective SAP. Messages from the services will be passed to the NFCIP-1 layer, here implemented with `libnfc`, that relays the messages to the other end of the connection.

When a packet is received by the LLCP layer, several actions are possible. When there is not yet a logical link between two services, the first packet will be a `CONNECT` PDU.[1] Upon receiving this PDU, the LLCP layer will check if there is a service registered for the requested SAP. If this is the case, a connection will be set up and all following packets will be delivered to this service.

The example we provide here is split into two parts. The main application is responsible for initialization and clean-up. This involves initialization of both `libnfc` and `libnfc-llcp`, setting up the LLCP link, connecting to the PN532, creating a thread for every service (here only one – the SNEP service) and freeing memory when the application terminates. The second part is the SNEP service

[1] PDU stands for Protocol Data Unit, which is the name of the LLCP packets.

thread itself, which serves as the actual application i.e., in this case sending and receiving a data packet.

Main application. The main application will first initialize both libraries.

```
// Initialize libnfc
static nfc_context *context;
nfc_init(&context);
if(context == NULL){
        printf("Unable to init libnfc.\n");
        exit(EXIT_FAILURE);
}
// Initialize libnfc-llcp
if(llcp_init() < 0)
        errx (EXIT_FAILURE, "llcp_init()");
```

After successful initialization, `libnfc` will be used to connect to the NFC device, in this case the PN532 connected over RS232 to serial port `/dev/ttyUL1`.

```
// Create an NFC device handler
nfc_device *device;
// Open the NFC device
if(!(device = nfc_open(context, "pn532_uart:/dev/ttyUL1
    :115200"))) {
        errx (EXIT_FAILURE, "Cannot connect to NFC device");
}
```

When the device is opened, a new LLCP link will be created as well as a handler to the SNEP service thread. When the service thread is successfully created, it will be bound to the LLCP link.

```
// Create LLCP link
struct llc_link *my_llc_link = llc_link_new();

// Create handler to the service thread
struct llc_service *snep_service;
// Create SNEP service
if(!(snep_service = llc_service_new_with_uri(NULL,
    snep_service_thread, LLCP_SNEP_URI, NULL)))
        errx (EXIT_FAILURE, "Cannot create snep service");

//Bind SNEP service to LLCP link
if(llc_link_service_bind(my_llc_link, snep_service,
    LLCP_SNEP_SAP) < 0) {
        errx (EXIT_FAILURE, "llc_service_new_with_uri()");
}
```

At this point everything is configured for NFC P2P communication. The only thing that needs to be done is activate the NFCIP-1 link, in this case the embedded platform will act as NFC initiator.

```
// Create MAC link
struct mac_link * my_mac_link = mac_link_new (device,
    my_llc_link);
if(!my_mac_link)
        errx (EXIT_FAILURE, "Cannot establish MAC link");

// Act as initiator
if(mac_link_activate_as_initiator(my_mac_link) <= 0)
        errx (EXIT_FAILURE, "Cannot activate link");
```

When a target device that supports NFC P2P is brought in the platform's vicinity, `libnfc` will set up a MAC link between the two devices. Then `libnfc-llcp` will establish the LLCP connection and start delivering messages from and to the SNEP service.

When this service terminates, the main thread will resume and free any allocated resources.

```
//Wait for MAC link to finish
void *status;
mac_link_wait(my_mac_link, &status);
// optionally check status

// free MAC and LLCP link
mac_link_free(my_mac_link);
llc_link_free(my_llc_link);

// close the connection to the PN532
nfc_close(device);

// terminate libraries
llcp_fini();
nfc_exit(context);

exit(EXIT_SUCCESS);
```

SNEP service thread. In the SNEP service thread we have written our own basic implementation of SNEP[2].

```
// Function prototype
void *snep_service_thread (void *arg);
```

[2]The application is in the service itself. The protocol functionality is in the files `ndef.c/h` and `snep.c/h` – see online/appendix !?!?!?

When the thread is created, it can get hold of the LLCP connection by means of the function argument.

```
struct llc_connection *connection = (struct llc_connection
    *) arg;
```

We create an NDEF record containing the data we want to send together with some data buffers.

```
struct ndef_record *myRec;
if((myRec = malloc(sizeof *myRec))) {
        myRec->MB = 1;
        myRec->ME = 1;
        myRec->CF = 0;
        myRec->IL = 0;
        myRec->TNF = 2;
        myRec->SR = 1;
        myRec->type = (uint8_t *)"text/plain";
        myRec->type_length = strlen((char *)myRec->type);
        myRec->payload = (uint8_t *)"This is some data!";
        myRec->payload_length = strlen((char *)myRec->
            payload);
}

uint8_t recv_buffer[1024];
uint8_t send_buffer[1024];
```

To send a packet, the NDEF record is packed in a SNEP message and placed in a buffer and sent over the LLCP connection. The receiving party should answer with a SNEP *SUCCESS* response. When we do receive a message, it is unpacked and by examining the type we can determine if it is actually a success.

```
// Pack NDEF message
snep_pack(myRec, send_buffer);
// Send SNEP message
llc_connection_send (connection, send_buffer, len);

//Receiving the SNEP success response!
if((len = llc_connection_recv (connection, recv_buffer,
    sizeof (res_buffer), NULL)) < 0)
        return NULL;

// Answer received, check if it's a success.
struct snep_message *msg = snep_unpack(recv_buffer, len);
if(msg->type_field == RESPONSE_SUCCESS) {
        llcp_log_log("[nfc-p2p-example.c]",
            LLC_PRIORITY_DEBUG, "SNEP Response: Success" );
}else{
```

```
        llcp_log_log("[nfc-p2p-example.c]",
            LLC_PRIORITY_FATAL, "SNEP Response: Not success
            !!" );
}
```

Receiving data packets works exactly the same way as receiving a SNEP *SUCCESS* response. Aside from a method for unpacking the SNEP message, we also provided a method for unpacking the NDEF record (i.e., the payload of the SNEP message). When the sending party, i.e., the client, sends a SNEP message, it will be a *PUT* message. This implies that the server needs to answer with a response success.

```
// Receiving a SNEP packet
if((len = llc_connection_recv (connection, recv_buffer,
    sizeof (recv_buffer), NULL)) < 0)
        return NULL;
// Unpack
msg = snep_unpack(recv_buffer, sizeof(recv_buffer));
struct ndef_record *record = ndef_unpack(msg->ndef_message,
    msg->data_length);

// Send a response
if(msg->type_field == REQUEST_PUT ) {
        int length = -1;
        uint8_t *success_response =
            snep_create_success_response(&length);

        llc_connection_send(connection, success_response, 6)
            ;
}
```

Specifications, Datasheets and Libraries

[1] ISO/IEC 14443-1:2008 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 1: Physical characteristics.

[2] ISO/IEC 14443-2:2010 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 2: Radio frequency power and signal interface.

[3] ISO/IEC 14443-3:2011 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 3: Initialization and anticollision.

[4] ISO/IEC 14443-4:2008 Identification cards – Contactless integrated circuit cards – Proximity cards – Part 4: Transmission protocol.

[5] ISO/IEC 18092:2013 Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1).

[6] ISO/IEC 8859-1:1998 Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1.

[7] Altera®. Nios II Processor Reference Handbook. `http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf`, 2011.

[8] Altera®. Avalon Interface Specifications. `www.altera.com/literature/manual/mnl_avalon_spec.pdf`, 2013.

[9] American National Standards Institute. ANSI X9.62 The Elliptic Curve Digital Signature Algorithm (ECDSA). `http://www.ansi.org/`.

[10] ARM Ltd. AMBA AXI and ACE Protocol Specification - AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite, 2011.

[11] Atmel®. AT97SC3204, Trusted Platform Module LPC Interface, SUMMARY DATASHEET, 2012.

[12] AuthenTec Inc. SafeXcel IP Public Key Accelerators. http://www.authentec.com/Portals/5/Documents/3103_SafeXcelIP_Public_Key_Accelerators_rv02_forPrint.pdf.

[13] Google. Android ndef push protocol specification. http://source.android.com/compatibility/ndef-push-protocol.pdf., 2011.

[14] Trusted Computing Group. TPM Main Specification Level 2 Version 1.2, Revision 116. http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2011.

[15] Hans-Jürgen Koch. The Userspace I/O HOWTO. https://www.kernel.org/doc/htmldocs/uio-howto.html, 2009.

[16] R. Herveille. Whisbone – WISHBONE System-on-Chip (SoC)Interconnection Architecture for Portable IP Cores. http://cdn.opencores.org/downloads/wbspec_b4.pdf, 2010.

[17] IBM Research – Zurich. Specification of the Identity Mixer Cryptographic Library – Version 2.3.2, 2010.

[18] IBM®. IBM 4765 PCIe Cryptographic Coprocessor, 2011.

[19] IEEE. IEEE Standard for Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pages 1–444, 2013.

[20] libnfc. Public platform independent Near Field Communication (NFC) library. Project website: http://code.google.com/p/libnfc/.

[21] N. Borenstein and N. Freed. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, 1996.

[22] National Institute of Standards and Technology (NIST). FIPS PUB 140-2: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf, 2001.

[23] National Institute of Standards and Technology (NIST). FIPS PUB 197: Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[24] National Institute of Standards and Technology (NIST). FIPS 186-3: Digital Signature Standard. http://csrc.nist.gov/, 2009.

[25] National Institute of Standards and Technology (NIST). FIPS PUB 180-4: Secure Hash Standard (SHS). http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, 2012.

[26] NFC Forum. NFC Data Exchange Format (NDEF) Technical Specification., 2006.

[27] NFC Forum. NFC Record Type Definition (RTD) Technical Specification., 2006.

[28] NFC Forum. URI Record Type Definition Technical Specification., 2006.

[29] NFC Forum. NFC Digital Protocol Technical Specification, 2010.

[30] NFC Forum. Logical Link Control Protocol Technical Specification, 2011.

[31] NFC Forum. Simple NDEF Exchange Protocol Technical Specification, 2011.

[32] NXP. AN10609_3 PN532 C106 application note Rev. 1.2, 2010.

[33] Department of Defense. Trusted Computer System Evaluation Criteria. http://csrc.nist.gov/publications/history/dod85.pdf, 1985.

[34] Standards for Efficient Cryptography. SEC2: Recommended Elliptic Curve Domain Parameters. http://www.secg.org/.

[35] Stefan Brands, Christian Paquin. U-Prove cryptographic specification v1.0, 2010. Microsoft Corporation.

[36] STMicroelectronics®. ST19NP18-TPM, Trusted Platform Module, Data Brief, 2011.

[37] STMicroelectronics®. ST33TPM12SPI, SPI based on 32-bit ARM® SecurCore® SC300TM CPU, Data Brief, 2011.

[38] Torbjörn Granlund and the GMP development team. The GNU Multiple Precision Arithmetic Library - Edition 5.1.1. http://gmplib.org/gmp-man-5.1.1.pdf, 2013.

[39] Xilinx®. System ACE CompactFlash Solution DS080 (v2.0) – Product Specificatio. http://www.xilinx.com/support/documentation/data_sheets/ds080.pdf, 2008.

[40] Xilinx®. DS449 - LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c), Product Specification. http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf, 2010.

[41] Xilinx®. DS531 - LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a), Product Specification. http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf, 2010.

[42] Xilinx®. AXI Reference Guide UG761 (v13.1). http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf, 2011.

[43] Xilinx®. MicroBlaze Processor Reference Guide – Embedded Development Kit EDK 14.1 UG081 (v14.1). http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf, 2012.

[44] Xilinx®. ML605 Hardware User Guide UG534 (v1.8). http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf, 2012.

[45] Xilinx®. Virtex-6 Family Overview DS150 (v2.4) – Product Specification. http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf, 2012.

[46] Xilinx®. Zynq-7000 All Programmable SoC Overview DS190 (v1.3) – Preliminary Product Specification. http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, 2013.

[47] Xilinx®, by: Khang Dao and Dylan Buli. AXI Multi-Ported Memory Controller XAPP739 (v1.0). http://www.xilinx.com/support/documentation/application_notes/xapp739_axi_mpmc.pdf, 2011.

Bibliography

[Acq04] Alessandro Acquisti. Privacy in Electronic Commerce and the Economics of Immediate Gratification. In *Proceedings of the Fifth ACM Conference on Electronic Commerce*, EC '04, pages 21–29. ACM, 2004.

[All03] Anita L. Allen. *Why Privacy Isn't Everything: Feminist Reflections on Personal Accountability.* Rowman & Littlefield, 2003.

[APH08] Philip Amberg, Nathaniel Pinckney, and David M. Harris. Parallel High-Radix Montgomery Multipliers. In *Proceedings of the Forty-Second Asilomar Conference on Signals, Systems and Computers*, pages 772–776, 2008.

[Bal08] Josep M. Balasch. Smart Card Implementation of Anonymous Credentials. Master's thesis, KU Leuven - Faculty of Engineering - Departement of Electrical Engineering, 2008. Bart Preneel (promotor).

[Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 311–323. Springer Berlin Heidelberg, 1987.

[BBRS06] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G. Sheridan. The Smart Phone: A Ubiquitous Input Device. *Pervasive Computing, IEEE*, 5(1):70–77, 2006.

[BCC04] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In *Proceedings of the Eleventh ACM Conference on Computer and Communications Security*, CCS '04, pages 132–145. ACM, 2004.

[BCD+12] Patrik Bichsel, Jan Camenisch, Bart De Decker, Jorn Lapon, Vincent Naessens, and Dieter Sommer. Data-Minimizing Authentication Goes Mobile. In Bart De Decker and David W. Chadwick, editors, *Communications and Multimedia Security*, volume 7394 of *Lecture Notes in Computer Science*, pages 55–71. Springer Berlin Heidelberg, 2012.

[BCDvdG88] Ernest F. Brickell, David Chaum, Ivan B. Damgård, and Jeroen van de Graaf. Gradual and Verifiable Release of a Secret (Extended Abstract). In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 156–166. Springer Berlin Heidelberg, 1988.

[BCGS09] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. Anonymous Credentials on a Standard Java Card. In *Proceedings of the Sixteenth ACM Conference on Computer and Communications Security*, CCS '09, pages 600–610. ACM, 2009.

[BCM05] Endre Bangerter, Jan Camenisch, and Ueli Maurer. Efficient Proofs of Knowledge of Discrete Logarithms and Representations in Groups with Hidden Order. In Serge Vaudenay, editor, *Public Key Cryptography – PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 154–171. Springer Berlin Heidelberg, 2005.

[BDK97] Jean-Claude Bajard, Laurent-Stéphane Didier, and Peter Kornerup. An RNS Montgomery modular multiplication algorithm. In *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, pages 234–239, 1997.

[Ber13] Sam Van Den Berge. Onderzoek en realisatie van P2P communicatie tussen een Android smartphone en een embedded NFC terminal. Master's thesis, KAHO Sint-Lieven – Dept. I.I., Electronics-ICT, 2013. Jean-Pierre Goemaere (promotor).

[BFK01] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag New York, Inc., 2001.

[BGV94] Antoon Bosselaers, René Govaerts, and Joos Vandewalle. Comparison of three modular reduction functions. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 175–186. Springer Berlin Heidelberg, 1994.

[Blu99] Thomas Blum. Modular Exponentiation on Reconfigurable Hardware. Master's thesis, Worcester Polytechnic Institute, 1999. Christof Paar (promotor).

[Boo51] Andrew D. Booth. A Signed Binary Multiplication Technique. *Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[Bou00] Fabrice Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Berlin Heidelberg, 2000.

[BP99] Thomas Blum and Christof Paar. Montgomery Modular Exponentiation on Reconfigurable Hardware. In *Proceedings of the 14^{th} IEEE Symposium on Computer Arithmetic*, pages 70–77, 1999.

[BP01] Thomas Blum and Christof Paar. High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware. *IEEE Transactions on Computers*, 50(7):759–764, 2001.

[Bra00] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy.* MIT Press, 2000.

[CFT98] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy Come – Easy Go Divisible Cash. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer Berlin Heidelberg, 1998.

[CG08] Jan Camenisch and Thomas Groß. Efficient Attributes for Anonymous Credentials. In *Proceedings of the Fifteenth ACM Conference on Computer and Communications Security*, CCS '08, pages 345–356. ACM, 2008.

[Cha85] David Chaum. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[cKKAJ96] Çetin K. Koç, Tolga Acar, and Burton S. Kaliski Jr. Analyzing and Comparing Montgomery Multiplication Algorithms. *Micro, IEEE*, 16(3), 1996.

[CKY09] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the Portability of Generalized Schnorr Proofs. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 425–442. Springer Berlin Heidelberg, 2009.

[CL01] Jan Camenisch and Anna Lysyanskaya. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Berlin Heidelberg, 2001.

[CL03] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Berlin Heidelberg, 2003.

[CL05] Chin-Chen Chang and Yeu-Pong Lai. A Parallel Modular Exponentiation Scheme for Transformed Exponents. In Jiannong Cao, Wolfgang Nejdl, and Ming Xu, editors, *Advanced Parallel Processing Technologies*, volume 3756 of *Lecture Notes in Computer Science*, pages 443–452. Springer Berlin Heidelberg, 2005.

[Cla12] Sarah Clark. Paris commuters to get travel passes that work with NFC phones. 2012. Online: `http://www.nfcworld.com/2012/01/30/312832/paris-commuters-to-get-travel-passes-that-work-with-nfc-phones/`.

[CP93] David Chaum and Torben P. Pedersen. Wallet Databases with Observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, Lecture Notes in Computer Science, pages 89–105. Springer Berlin Heidelberg, 1993.

[CS97] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer Berlin Heidelberg, 1997.

[DF02] Ivan B. Damgård and Eiichiro Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT '02*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer Berlin Heidelberg, 2002.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Dhe98] Jean-François Dhem. *Design of an Efficient Public-Key Cryptographic Library for RISC-based Smart Cards*. PhD thesis, Université Catholique de Louvain, 1998. Jean-Jacques Quisquater (promotor).

[Die10] Kurt Dietrich. Anonymous Credentials for Java Enabled Platforms: A Performance Evaluation. In Liqun Chen and Moti Yung, editors, *Trusted Systems*, volume 6163 of *Lecture Notes in Computer Science*, pages 88–103. Springer Berlin Heidelberg, 2010.

[DJM97] V.S. Dimitrov, G.A. Jullien, and W.C. Miller. Algorithms for Multi-Exponentiation Based on Complex Arithmetic. In *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, pages 208–215, 1997.

[dlPTC11] Antonio de la Piedra, Abdellah Touhafi, and Gianluca Cornetta. Cryptographic Accelerator for 802.15.4 Transceivers with Key Agreement Engine Based on Montgomery Arithmetic. In *Proceedings of the 18th IEEE Symposium on Communications and Vehicular Technology in the Benelux*, pages 1–5, 2011.

[dMMN05] Luiza de Macedo Mourelle and Nadia Nedjah. Hardware for Modular Exponentiation Suitable for Smart Cards. In Zhaohui Wu, Chun Chen, Minyi Guo, and Jiajun Bu, editors, *Embedded Software and Systems*, volume 3605 of *Lecture Notes in Computer Science*, pages 196–202. Springer Berlin Heidelberg, 2005.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.

[DOH+10] Bert Dufraimont, Geoffrey Ottoy, Tom Hamelinckx, Lieven De Strycker, and Jean-Pierre Goemaere. The difference between a Software and a Hardware implementation of AES in a ZigBee Network. In Lieven De Strycker and Anneleen Van Nieuwenhuyse, editors, *Proceedings of the Fourth European Conference on the Use of Modern Information and Communication Technologies*, volume 1, pages 225–232. Nevelland v.z.w., 2010.

[DPWD12] David Derler, Klaus Potzmader, Johannes Winter, and Kurt Dietrich. Anonymous Ticketing for NFC-Enabled Mobile Phones. In Liqun Chen, Moti Yung, and Liehuang Zhu, editors, *Trusted Systems*, volume 7222 of *Lecture Notes in Computer Science*, pages 66–83. Springer Berlin Heidelberg, 2012.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., 2002.

[DW09] Gauthier Van Damme and Karel Wouters. Practical Experiences with NFC Security on Mobile Phones. In *Workshop on RFID Security 2009*, Lecture Notes in Computer Science, page 13. Springer-Verlag, 2009.

[Elg85] Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Eur12] European Network of Excellence in Cryptology II. ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012). `http://www.ecrypt.eu.org/documents/D.SPA.20.pdf`, 2012.

[FBV11] Junfeng Fan, Lejla Batina, and Ingrid Verbauwhede. Design and Design Methods for Unified Multiplier and Inverter and Its Application for HECC. *Integration, the VLSI Journal*, 44(4):280–289, 2011.

[FHMM10] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones. In Siddika Berna Örs, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 35–49. Springer Berlin Heidelberg, 2010.

[FHMM11] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones. Cryptology ePrint Archive, Report 2011/618, online: `http://eprint.iacr.org/2011/618`, 2011.

[FK07] Apostolos P. Fournaris and Odysseas G. Koufopavlou. Hardware Design Issues in Elliptic Curve Cryptographic for Wireless Security. In Nicolas Sklavos and Xinmiao Zhang, editors, *Wireless Security and Cryptography, Specifications and Implementations*, chapter 4, pages 79–151. CRC Press, 2007.

[FM05] Tim France-Massay. MULTOS - the High Security Smart Card OS. Technical report, MAOSCO Limited, 2005.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 1997.

[FS87] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in cryptology – CRYPTO '86*, Lecture Notes in Computer Science, pages 186–194. Springer Berlin Heidelberg, 1987.

[GC03] Kris Gaj and Paweł Chodowiec. Very Compact FPGA Implementation of the AES Algorithm. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2003.

[GGK+06] Jorge Guajardo, Tim Güneysu, Sandeep S. Kumar, Christof Paar, and Jan Pelzl. Efficient Hardware Implementation of Finite Fields with Applications to Cryptography. *Acta Applicandae Mathematica*, 93(1-3):75–118, 2006.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[GMS10] Jorge Guajardo, Bart Mennink, and Berry Schoenmakers. Anonymous Credential Schemes with Encrypted Attributes. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *Cryptology and Network Security*, volume 6467 of *Lecture Notes in Computer Science*, pages 314–333. Springer Berlin Heidelberg, 2010.

[GP08] Tim Güneysu and Christof Paar. Ultra High Performance ECC over NIST Primes on Commercial FPGAs. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 62–78. Springer Berlin Heidelberg, 2008.

[GRS96] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In *Information Hiding*, pages 137–150. Springer-Verlag, 1996.

[Han08] Gerhard Hancke. Eavesdropping Attacks on High-Frequency RFID Tokens. In *Proceedings of the 4^{th} Workshop on RFID Security (RFIDSec 08)*, pages 100–113, 2008.

[Hay05] Caroline Haythornthwaite. Social Networks and Internet Connectivity Effects. *Information, Community & Society*, 8(2):125–147, 2005.

[HB06] Ernst Haselsteiner and Klemens Breitfuß. Security in Near Field Communication (NFC). In *Workshop on RFID Security 2006*, 2006.

[HC09] Yajuan He and Chip-Hong Chang. A New Redundant Binary Booth Encoding for Fast 2^n-Bit Multiplier Design. *IEEE Transactions on Circuits and Systems, I: Regular Papers*, 56(6):1192–1201, 2009.

[Hea11] Heather McLean. Transport for London to accept NFC payments from 2012. 2011. Online: `http://www.nfcworld.com/2011/07/12/38537/transport-for-london-to-accept-nfc-payments-from-2012/`.

[HiHA13] Naofumi Homma, Yu ichi Hayashi, and Takafumi Aoki. Electromagnetic Information Leakage from Cryptographic Devices. In *Proceedings of the 2013 International Symposium on Electromagnetic Compatibility (EMC Europe 2013)*, pages 401–404, 2013.

[HMM09] Gerhard P Hancke, KE Mayes, and Konstantinos Markantonakis. Confidence in Smart Token Proximity: Relay Attacks Revisited. *Computers & Security*, 28(7):615–627, 2009.

[HPO13] Jens Hermans, Roel Peeters, and Cristina Onete. Efficient, Secure, Private Distance Bounding without Key Updates. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 207–218. ACM, 2013.

[HQ00] Gaël Hachez and Jean-Jacques Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 293–301. Springer Berlin Heidelberg, 2000.

[Joy07] Marc Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 135–147. Springer Berlin Heidelberg, 2007.

[Joy08] Marc Joye. RSA Moduli with a Predetermined Portion: Techniques and Applications. In Liqun Chen, Yi Mu, and Willy Susilo, editors, *Information Security Practice and Experience*, volume 4991 of *Lecture Notes in Computer Science*, pages 116–130. Springer Berlin Heidelberg, 2008.

[KBV09] Miroslav Knežević, Lejla Batina, and Ingrid Verbauwhede. Modular Reduction Without Precomputational Phase. In *IEEE International Symposium on Circuits and Systems – ISCAS 2009*, pages 1389–1392, 2009.

[KJJ08] Paul C. Kocher, Josh Jaffe, and Ben Jun. Differential power analysis, 2008. Presented at the CRYPTO '98 Rump Session.

[Kne11] Miroslav Knežević. *Efficient Hardware Implementations of Cryptographic Primitives*. PhD thesis, KU Leuven - Faculty of Engineering - Departement of Electrical Engineering, 2011. Ingrid Verbauwhede (promotor).

[KO63] Anatolii A. Karatsuba and Yuri P. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. *Translation in Physics-Doklady*, 145:597–596, 1963.

[Kob87] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin Heidelberg, 1996.

[KT05] Marcelo E. Kaihara and Naofumi Takagi. Bipartite Modular Multiplication. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 201–210. Springer Berlin Heidelberg, 2005.

[KVV10] Miroslav Knežević, Frederik Vercauteren, and Ingrid Verbauwhede. Speeding Up Bipartite Modular Multiplication. In Anwar M. Hasan and Tor Helleseth, editors, *Arithmetic of Finite Fields*, volume 6087 of *Lecture Notes in Computer Science*, pages 166–179. Springer Berlin Heidelberg, 2010.

[KW05] Ziv Kfir and Avishai Wool. Picking Virtual Pockets using Relay Attacks on Contactless Smartcard. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks – SecureComm 2005*, pages 47–58, 2005.

[KZG07] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-Based Onion Routing. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 95–112. Springer Berlin Heidelberg, 2007.

[Lap12] Jorn Lapon. *Anonymous Credential Systems: From Theory Towards Practice*. PhD thesis, KU Leuven - Faculty of Engineering - Departement of Computer Science, 2012. Bart De Decker (promotor).

[LC03] Yeu-Pong Lai and Chin-Chen Chang. An Efficient Multi-Exponentiation Scheme Based on Modified Booth's Method. *International Journal of Electronics*, 90(3):221–233, 2003.

[Len98] Arjen K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 1998.

[Len04] Arjen K. Lenstra. Key Lengths. *Handbook of Information Security*, 2:617–635, 2004.

[LV00] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 446–465. Springer Berlin Heidelberg, 2000.

[Mac13] Ewen MacAskill. NSA paid millions to cover Prism compliance costs for tech companies. *The Guardian*, August 2013. published online: `http://www.theguardian.com/world/2013/aug/23/nsa-prism-costs-tech-companies-paid`.

[McC90] Kevin S. McCurley. The Discrete Logarithm Problem. In *Proceedings of Symposia in Applied Mathematics*, volume 42, pages 49–74, 1990.

[Mil86] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Lecture Notes in Computer Sciences; 218 on Advances in Cryptology – CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, 1986.

[Mon85] Peter L. Montgomery. Modular Multiplication Without Trail Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[MSPV07] Nele Mentens, Kazuo Sakiyama, Bart Preneel, and Ingrid Verbauwhede. Efficient Pipelining for Modular Multiplication Architectures in Prime Fields. In *Proceedings of the Seventeenth ACM Great Lakes Symposium on VLSI*, GLSVLSI '07, pages 534–539. ACM, 2007.

[MV12] Wojciech Mostowski and Pim Vullers. Efficient U-Prove Implementation for Anonymous Credentials on Smart Cards. In Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *Security and Privacy in Communication Networks*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 243–260. Springer Berlin Heidelberg, 2012.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*, chapter 14. CRC Press, Inc., 1st edition, 1996.

[Nd02] Nadia Nedjah and Luiza de Macedo Mourelle. Reconfigurable Hardware Implementation of Montgomery Modular Multiplication and Parallel Binary Exponentiation. In *Digital System Design, 2002. Proceedings. Euromicro Symposium on*, pages 226–233, 2002.

[Nd06] Nadia Nedjah and Luiza de Macedo Mourelle. Three Hardware Architectures for the Binary Modular Exponentiation: Sequential, Parallel, and Systolic. *IEEE Transactions on Circuits and Systems, I: Regular Papers*, 53(3):627–633, 2006.

[NdMM03] Nadia Nedjah and Luiza de Macedo Mourelle. Fast Reconfigurable Systolic Hardware for Modular Multiplication and Exponentiation. *Journal of Systems Architecture*, 49:387–396, 2003.

[NFC11] NFC Forum. NFC in public transport – white paper. 2011. Online: `http://www.nfc-forum.org/resources/white_papers/NFC_in_Public_Transport.pdf`.

[NTR11] João C. Néto, Alexandre F. Tenca, and Wilson V. Ruggiero. A Parallel k-Partition Method to Perform Montgomery Multiplication. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 251–254, 2011.

[OBPV03] Siddika Berna Örs, Lejla Batina, Bart Preneel, and Joos Vandewalle. Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 8 pp., 2003.

[OC] Geoffrey Ottoy and Jonas De Craene. Modular Simultaneous Exponentiation IP Core Specification. published online: `https://lirias.kuleuven.be/bitstream/123456789/369058/1/mod_sim_exp.pdf`.

[ODW+12] Geoffrey Ottoy, Sven Depoorter, Nick Warlop, Jean-Pierre Goemaere, and Lieven De Strycker. Evaluation of a Secure Authentication Protocol for Access Control over Near Field Communication. *Annual Journal of Electronics*, 6(1):52–55, 2012.

[OHP+10a] Geoffrey Ottoy, Tom Hamelinckx, Bart Preneel, Lieven De Strycker, and Jean-Pierre Goemaere. AES Data Encryption in a ZigBee Network: Software or Hardware? In Andreas U. Schmidt, Giovanni Russello, Antonio Lioy, Neeli R. Prasad, and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems*, volume 47 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 163–173. Springer Berlin Heidelberg, 2010.

[OHP+10b] Geoffrey Ottoy, Tom Hamelinckx, Bart Preneel, Lieven De Strycker, and Jean-Pierre Goemaere. On the Choice of the Appropriate AES Data Encryption Method for ZigBee Nodes. *Security and Communication Networks*, 2010. published online (doi:10.1002/sec.267).

[OLN+13] Geoffrey Ottoy, Jorn Lapon, Vincent Naessens, Bart Preneel, and Lieven Strycker. Dedicated Hardware for Attribute-Based Credential Verification. In Bart De Decker, Jana Dittmann, Christian Kraetzer, and Claus Vielhauer, editors, *Communications and Multimedia Security*, volume 8099 of *Lecture Notes in Computer Science*, pages 50–65. Springer Berlin Heidelberg, 2013.

[OMS+11] Geoffrey Ottoy, Jeroen Martens, Nick Saeys, Bart Preneel, Lieven De Strycker, Jean-Pierre Goemaere, and Tom Hamelinckx. A Modular Test Platform for Evaluation of Security Protocols in NFC Applications. In Bart De Decker, Jorn Lapon, Vincent Naessens, and Andreas Uhl, editors, *Communications and Multimedia Security*, volume 7025 of *Lecture Notes in Computer Science*, pages 171–177. Springer Berlin Heidelberg, 2011.

[OPGS13] Geoffrey Ottoy, Bart Preneel, Jean-Pierre Goemaere, and Lieven Strycker. Flexible Design of a Modular Simultaneous Exponentiation Core for Embedded Platforms. In Philip Brisk, José Gabriel Figueire do Coutinho, and Pedro C. Diniz, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7806 of *Lecture Notes in Computer Science*, pages 115–121. Springer Berlin Heidelberg, 2013.

[OPS+13] Geoffrey Ottoy, Bart Preneel, Nobby Stevens, Jean-Pierre Goemaere, and Lieven De Strycker. Open-source hardware for embedded security. *EDN*, 2013. online: `http://www.edn.com/design/systems-design/4406271/Open-source-hardware-for-embedded-security`, art.nr. 4406271.

[OPV07] Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. Side-Channel Analysis Attacks on Hardware Implementations of Cryptographic Algorithms. In Nicolas Sklavos and Xinmiao Zhang, editors, *Wireless Security and Cryptography, Specifications and Implementations*, chapter 7, pages 213–247. CRC Press, 2007.

[Ott12] Geoffrey Ottoy. Flexible Design of a High-Radix Modular Simultaneous Exponentiation Core. 2012. RFIDsec2012. Nijmegen, 1-3 July.

[Pau07] Annika Paus. Near Field Communication in Cell Phones, 2007. Seminararbeit at the Ruhr-Universität Bochum – Chair for Communication Security. Christof Paar (supervisor).

[Ped92] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin Heidelberg, 1992.

[PH07] Nathaniel Pinckney and David M. Harris. Parallelized Radix-4 Scalable Montgomery Multipliers. In Antonio Petraglia, Volnei A. Pedroni, and Gert Cauwenberghs, editors, *Proceedings of the Twentieth Annual Conference on Integrated Circuits and Systems Design*, SBCCI '07, pages 306–311. ACM, 2007.

[PH08] Nathaniel Pinckney and David M. Harris. Parallelized Radix-4 Scalable Montgomery Multipliers. *Journal of Integrated Circuits and Systems*, pages 39–45, 2008.

[PH12] Roel Peeters and Jens Hermans. Wide Strong Private RFID Identification based on Zero-Knowledge. *IACR Cryptology ePrint Archive*, 2012:389, 2012.

[Phi01] Braden Phillips. Modular Multiplication in the Montgomery Residue Number System. In *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers.*, volume 2, pages 1637–1640, 2001.

[PvD02] M.J. Potgieter and B.J. van Dyk. Two Hardware Implementations of the Group Operations Necessary for Implementing an Elliptic Curve Cryptosystem over a Characteristic Two Finite Field. In 6^{th} *IEEE Africon Conference in Africa*, volume 1, pages 187–192, 2002.

[Qui92] Jean-Jacques Quisquater. Encoding System According to the So-Called RSA Method, by Means of a Microcontroller and Arrangement Implementing this System. US Patent 5,166,978, 1992.

[RK13] Kurt Rosenfeld and Ramesh Karri. Attacks and Defenses for JTAG. *Design Test, IEEE*, PP(99):1–12, 2013.

[Rol12] Michael Roland. Software Card Emulation in NFC-Enabled Mobile Phones: Great Advantage or Security Nightmare. In *Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use (IWSSI/SPMU 2012)*, page 6, 2012.

[Ros11] Jeffrey Rosen. Free Speech, Privacy, and the Web that Never Forgets. *J. on Telecomm. & High Tech. L.*, 9:345–356, 2011.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.

[RSA78] Ron 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, 1978.

[RT09] Dirk Rykx and Joris Thielen. Evaluatie van nieuwe hashfunctiekandidaten op FPGA, 2009. Master Thesis, KHLim, Nele Mentens (promotor).

[RTŠ+12] Aanjhan Ranganathan, Nils Ole Tippenhauer, Boris Škorić, Dave Singelée, and Srdjan Čapkun. Design and implementation of a terrorist fraud resilient distance bounding system. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security – ESORICS 2012*, volume 7459 of *Lecture Notes in Computer Science*, pages 415–432. Springer Berlin Heidelberg, 2012.

[SB10] Kazuo Sakiyama and Lejla Batina. Arithmetic for Public-Key Cryptography. In Ingrid Verbauwhede, editor, *Secure Integrated Circuits and Systems, Integrated Circuits and Systems*, pages 75–118. Springer, 2010.

[Sch90] Claus P. Schnorr. Efficient Identification and Signatures for Smart Cards. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689. Springer Berlin Heidelberg, 1990.

[Sch91] Claus P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[SDI11] Gustavo D. Sutter, Jean-Pierre Deschamps, and José L. Imaña. Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem Based on Digit Serial Computation. *IEEE Transactions on Industrial Electronics*, 58(7):3101–3109, 2011.

[SGPV09] Michaël Sterckx, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Efficient Implementation of Anonymous Credentials on Java Card Smart Cards. In *First IEEE International Workshop on Information Forensics and Security – WIFS 2009*, pages 106–110, 2009.

[SKF+11] Kazuo Sakiyama, Miroslav Knežević, Junfeng Fan, Bart Preneel, and Ingrid Verbauwhede. Tripartite Modular Multiplication. *Integration, the {VLSI} Journal*, 44(4):259–269, 2011.

[SKN08] Koji Shigemoto, Kensuke Kawakami, and Koji Nakano. Accelerating Montgomery Modulo Multiplication for Redundant Radix-64k Number System on the FPGA Using Dual-Port Block RAMs. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing – EUC '08*, volume 1, pages 44–51, 2008.

[Sol02] Daniel J. Solove. Conceptualizing Privacy. *California Law Review*, 90(4):1087–1155, 2002.

[Sol04] Daniel J Solove. *The Digital Person: Technology and Privacy in the Information Age*, volume 1. NYU Press, 2004.

[Sol06] Daniel J. Solove. A Taxonomy of Privacy. *University of Pennsylvania Law Review*, 154(3):477–564, 2006.

[Spa13] Shaun Spalding. Privacy Issues for Website Operators – Protect Yourself by Protecting Users. *online:* `http://privacypolicyexamples.com/bonus%20content/introduction/`, 2013.

[SSBV11] Dave Singelée, Stefaan Seys, Lejla Batina, and Ingrid Verbauwhede. The Communication and Computation Cost of Wireless Security – Extended Abstract. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, pages 1–4. ACM, 2011.

[SST04] Hisayoshi Sato, Daniel Schepers, and Tsuyoshi Takagi. Exact Analysis of Montgomery Multiplication. In *Proceedings of the 5^{th} International Conference on Cryptology in India*, INDOCRYPT '04, pages 290–304. Springer-Verlag, 2004.

[Sta06] William Stallings. Processor Structure and Function. In *Computer Organization And Architecture: Designing For Performance*, chapter 12, pages 433–434. Pearson Prentice Hall, 2006.

[TcKK99] Alexandre F. Tenca and Çetin K. Koç. A Scalable Architecture for Montgomery Multiplication. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 94–108. Springer Berlin Heidelberg, 1999.

[TcKK03] Alexandre F. Tenca and Çetin K. Koç. A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm. *IEEE Transactions on Computers*, 52(9):1215–1221, 2003.

[TcKK07] Lo'ai A. Tawalbeh and Çetin K. Koç. Efficient Elliptic Curve Cryptographic Hardware Design for Wireless Security. In Nicolas Sklavos and Xinmiao Zhang, editors, *Wireless Security and Cryptography, Specifications and Implementations*, chapter 5, pages 153–175. CRC Press, 2007.

[TG06] Andrew S. Tanenbaum and James R. Goodman. *Structured Computer Organization*. Prentice Hall PTR, 5th edition, 2006.

[The12] The Point of Sale News. Where Is NFC Going? New Reports Forecast Growth. 2012. Online: `http://pointofsale.com/20120319953/Mobile-POS-News/where-is-nfc-going-new-reports-forecast-growth.html` (date consulted October 10, 2013).

[TJ09] Hendrik Tews and Bart Jacobs. Performance Issues of Selective Disclosure and Blinded Issuing Protocols on Java Card. In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, Chris J. Mitchell, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, volume 5746 of *Lecture Notes in Computer Science*, pages 95–111. Springer Berlin Heidelberg, 2009.

[VA13] Pim Vullers and Gergely Alpár. Efficient Selective Disclosure on Smart Cards using Idemix. In Simone Fischer-Hübner and Elisabeth de Leeuw, editor, *Proceedings of the 3rd IFIP WG 11.6 Working Conference on Policies and Research in Identity Management, IDMAN 2013*, IFIP Advances in Information and Communication Technology. Springer-Verlag, 2013.

[Ver12] Ingrid Verbauwhede. Efficient and Secure Hardware. *Datenschutz und Datensicherheit*, 36(12):872–875, 2012.

[VMG+10] Jo Vliegen, Nele Mentens, Jan Genoe, An Braeken, Serge Kubera, Abdellah Touhafi, and Ingrid Verbauwhede. A Compact FPGA-based Architecture for Elliptic Curve Cryptography over Prime Fields. In *Proceedings of the 21st IEEE International Conference on Application-specific Systems Architectures and Processors*, pages 313–316. IEEE, 2010.

[Wal99] Colin D. Walter. Montgomery Exponentiation Needs No Final Subtractions. *Electronics Letters*, 35(21):1831–1832, 1999.

[Wal08] Colin D. Walter. Leakage from Montgomery Multiplication. In Çetin K. Koç, editor, *Cryptographic Engineering*, pages 431–449. Springer, 2008.

[WB90] Samuel D Warren and Louis D Brandeis. The Right to Privacy. *Harvard Law Review*, 4(5):193–220, 1890.

[WLLC07] Chia-Long Wu, Der-Chyuan Lou, Jui-Chang Lai, and Te-Jen Chang. Fast Modular Multi-Exponentiation Using Modified Complex Arithmetic. *Applied Mathematics and Computation*, 186(2):1065–1074, 2007.

[ZW10] Yachao Zhou and Xiaojun Wang. An Improved Implementation of Montgomery Algorithm Using Efficient Pipelining and Structured Parallelism Techniques. In *Signals and Systems Conference (ISSC 2010), IET Irish*, pages 7–11, 2010.

Curriculum Vitae

Geoffrey Ottoy was born in Aalst, Belgium on March 28, 1984. He studied Latin-Sciences at the Sint-Jozefscollege in Aalst. In 2006, he received the MSc. degree "Industrieel Ingenieur Electronica/ICT – optie Ontwerptechniek" from KAHO Sint-Lieven, Gent, Belgium.

In September 2006, he joined the DraMCo research group at KAHO Sint-Lieven, where he worked on an IWT TETRA project on "Locatie Afhankelijke Diensten – Wireless Technology for Positioning Applications / LADI-WITEPA". From 2009 he started work on his PhD, developing a "Reusable Embedded Hardware Architecture for Secure Wireless Communication". From 2006 onwards, he has been supervising lab exercises and theses.

List of Publications

Articles as Main Author

Articles in Internationally Reviewed Scientific Journals

[OPS+13] Geoffrey Ottoy, Bart Preneel, Nobby Stevens, Jean-Pierre Goemaere, and Lieven De Strycker. Open-source hardware for embedded security. *EDN*, 2013. online: `http://www.edn.com/design/systems-design/4406271/Open-source-hardware-for-embedded-security`, art.nr. 4406271.

[OHP+10a] Geoffrey Ottoy, Tom Hamelinckx, Bart Preneel, Lieven De Strycker, and Jean-Pierre Goemaere. AES Data Encryption in a ZigBee Network: Software or Hardware? In Andreas U. Schmidt, Giovanni Russello, Antonio Lioy, Neeli R. Prasad, and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems*, volume 47 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 163–173. Springer Berlin Heidelberg, 2010.

[OHP+10b] Geoffrey Ottoy, Tom Hamelinckx, Bart Preneel, Lieven De Strycker, and Jean-Pierre Goemaere. On the Choice of the Appropriate AES Data Encryption Method for ZigBee Nodes. *Security and Communication Networks*, 2010. published online (doi:10.1002/sec.267).

Papers at International Conferences and Symposia, Published in Full in Proceedings

[OLN+13] Geoffrey Ottoy, Jorn Lapon, Vincent Naessens, Bart Preneel, and Lieven Strycker. Dedicated Hardware for Attribute-Based Credential Verification. In Bart De Decker, Jana Dittmann, Christian Kraetzer, and Claus Vielhauer, editors, *Communications and Multimedia*

Security, volume 8099 of *Lecture Notes in Computer Science*, pages 50–65. Springer Berlin Heidelberg, 2013.

[OPGS13] Geoffrey Ottoy, Bart Preneel, Jean-Pierre Goemaere, and Lieven Strycker. Flexible Design of a Modular Simultaneous Exponentiation Core for Embedded Platforms. In Philip Brisk, José Gabriel Figueire do Coutinho, and Pedro C. Diniz, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7806 of *Lecture Notes in Computer Science*, pages 115–121. Springer Berlin Heidelberg, 2013.

[Ott12] Geoffrey Ottoy. Flexible Design of a High-Radix Modular Simultaneous Exponentiation Core. Poster Session of the 8$^{\text{th}}$ Workshop on RFID Security and Privacy 2012 – RFIDsec2012.

[ODW+12] Geoffrey Ottoy, Sven Depoorter, Nick Warlop, Jean-Pierre Goemaere, and Lieven De Strycker. Evaluation of a Secure Authentication Protocol for Access Control over Near Field Communication. *Annual Journal of Electronics*, 6(1):52–55, 2012.

[OMS+11] Geoffrey Ottoy, Jeroen Martens, Nick Saeys, Bart Preneel, Lieven De Strycker, Jean-Pierre Goemaere, and Tom Hamelinckx. A Modular Test Platform for Evaluation of Security Protocols in NFC Applications. In Bart De Decker, Jorn Lapon, Vincent Naessens, and Andreas Uhl, editors, *Communications and Multimedia Security*, volume 7025 of *Lecture Notes in Computer Science*, pages 171–177. Springer Berlin Heidelberg, 2011.

[OND+09] Geoffrey Ottoy, Anneleen Van Nieuwenhuyse, Kevin D'hoe, Lieven De Strycker, and Jean-Pierre Goemaere. Improving the Performance of a RSS-based Location Estimation System, Study and Evaluation. In *Proceedings of the European Conference on Positioning and Context-Awareness – POCA 2009*. 2009. (proceedings on CD-ROM).

[ODSG08] Geoffrey Ottoy, Kevin D'hoe, Lieven De Strycker, and Jean-Pierre Goemaere. Wireless Monitoring of Museum Display Cases: Study and Prototype Design. In Luc De Backer, editor, *Proceedings of the Third European Conference on the Use of Modern Information and Communication Technologies*, volume 1, pages 309–316. Nevelland v.z.w., 2008.

Miscellaneous Publications

[OC] Geoffrey Ottoy and Jonas De Craene. Modular Simultaneous Exponentiation IP Core Specification. published online: `https://lirias.kuleuven.be/bitstream/123456789/369058/1/mod_sim_exp.pdf`.

Articles as Co-Author

Articles in Internationally Reviewed Scientific Journals

[DNO+09] Kevin D'hoe, Anneleen Van Nieuwenhuyse, Geoffrey Ottoy, Lieven De Strycker, Luc De Backer, Jean-Pierre Goemaere, and Bart Nauwelaers. Influence of Different Types of Metal Plates on a High Frequency RFID Loop Antenna: Study and Design. *Advances in Electrical and Computer Engineering*, 9(2):3–8, 2009.

[DOGS08] Kevin D'hoe, Geoffrey Ottoy, Jean-Piere Goemaere, and Lieven De Strycker. Indoor Room Location Estimation. *Advances in Electrical and Computer Engineering*, 8(2):78–81, 2008.

Papers at International Conferences and Symposia, Published in Full in Proceedings

[WOG+12] Stijn Wielandt, Geoffrey Ottoy, Jean-Pierre Goemaere, Nobby Stevens, and Lieven De Strycker. Integration of a CAN bus in an Onboard Computer for Space Applications. In *Proceedings of the 11th International Conference on Development and Application Systems – DAS 2012*, pages 56–59, 2012.

[DOH+10] Bert Dufraimont, Geoffrey Ottoy, Tom Hamelinckx, Lieven De Strycker, and Jean-Pierre Goemaere. The Difference Between a Software and a Hardware Implementation of AES in a ZigBee Network. In Lieven De Strycker and Anneleen Van Nieuwenhuyse, editors, *Proceedings of the Fourth European Conference on the Use of Modern Information and Communication Technologies*, volume 1, pages 225–232. Nevelland v.z.w., 2010.

[DOG+10] Kevin D'hoe, Geoffrey Ottoy, Jean-Pierre Goemaere, Lieven De Strycker, and Nobby Stevens. Architecture of a ZigBee Integrated Passive RFID System. In *Proceedings of the Third International EURASIP Workshop on RFID Technology*, pages 87–92. 2010.

[DNO+09b] Kevin D'hoe, Anneleen Van Nieuwenhuyse, Geoffrey Ottoy, Jean-Pierre Goemaere, and Lieven De Strycker. A New Low-Cost HF RFID Loop Antenna Concept for Metallic Environments. In *Systems, Signals and Image Processing, 2009. IWSSIP 2009. 16th International Conference on*, pages 1–5, 2009.

[NOSG08] Anneleen Van Nieuwenhuyse, Geoffrey Ottoy, Lieven De Strycker, and Jean-Pierre Goemaere. Indoor Localisation Techniques: Comparison between the Use of RSS and TOA. In Luc De Backer, editor, *Proceedings of the Third European Conference on the Use of Modern*

Information and Communication Technologies, volume 1, pages 441–452. Nevelland v.z.w., 2008.

[BCG+07] Robrecht Bisschop, Laurent Cattoir, Jean-Pierre Goemaere, Geoffrey Ottoy, and Lieven De Strycker. Reliable Stray Detection System by Means of RFID. In *Proceedings of the 5th International Conference on Microelectronics and Computer Science – ICMCS 2007*, pages 387–390, 2007.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering Science
Department of Electrical Engineering
Computer Security and Industrial Cryptography
Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven (Belgium)