Sami Kerim Galal

Physical Layer Attacks on Sensor Nodes

Masters Thesis
May 2009 to November 2009

Supervisor: Nils Ole Tippenhauer
Supervisor: Professor Srdjan Čapkun
Abstract

The goal of this thesis was to evaluate different academic implementations of cryptographic algorithms for the Mica2 and TelosB sensor nodes and test their resilience against simple and differential power analysis. Weaknesses in physical implementation are exploited in several software packages including TinyPK, EccM and TinyECC. These include a Diffie Helmann key exchange with RSA, an elliptic curve implementation of the same key exchange, and an optimized Elliptic Curve Digital Signature algorithm, respectively. In addition, various modifications of TinyPK and EccM were enacted and evaluated. The results show that providing an implementation secure against this particular class of side channel attack is easy to ensure while maintaining similar performance statistics to the original implementations.
# Contents

1 Introduction .............................. 1
  1.1 Motivation .......................... 2
  1.2 Problem Statement ................. 2
  1.3 Related Work ....................... 3
  1.4 Overview .......................... 3

2 Background .............................. 5
  2.1 Cryptography ....................... 5
    2.1.1 Symmetric Cryptography ....... 5
    2.1.2 Asymmetric Cryptography ...... 6
  2.2 Cryptanalysis ...................... 6
    2.2.1 Side Channel Attacks ........... 6
    2.2.2 Simple Power Analysis ......... 7
    2.2.3 Differential Power Analysis ... 8
    2.2.4 Template Attacks ............... 8
  2.3 Hardware Platforms ................ 8
    2.3.1 Mica2 .......................... 9
    2.3.2 TelosB/TMote Sky ............... 9
  2.4 Cryptographic Algorithms ......... 10
    2.4.1 RSA ........................... 10
    2.4.2 ECC .......................... 11
    2.4.3 ECDSA ......................... 11
2.5 Cryptographic Implementations ........................................ 12
  2.5.1 TinySec ......................................................... 12
  2.5.2 MiniSec ......................................................... 13
  2.5.3 TinyPK & EccM .................................................. 13
  2.5.4 TinyECC ......................................................... 14
2.6 Attacker Model .......................................................... 14
3 Practical Attacks .......................................................... 15
  3.1 Code Analysis ....................................................... 16
    3.1.1 TinyECC ....................................................... 16
    3.1.2 TinyPK ......................................................... 17
    3.1.3 EccM .......................................................... 19
    3.1.4 TinySec ....................................................... 20
  3.2 Exploits ............................................................... 21
    3.2.1 TinyECC ....................................................... 21
    3.2.2 TinyPK ......................................................... 23
    3.2.3 EccM .......................................................... 25
    3.2.4 TinySec ....................................................... 27
  3.3 Discussion ............................................................. 29
4 Proposed Countermeasures ............................................... 31
  4.1 TinyPK .............................................................. 32
    4.1.1 Square-and-Always-Multiply ................................ 32
    4.1.2 Montgomery's Ladder Technique ............................. 34
    4.1.3 M-Ary Multiplication ........................................ 37
    4.1.4 Private Exponent Blinding .................................. 40
  4.2 EccM ................................................................. 41
    4.2.1 Double-and-Always-Add ...................................... 41
    4.2.2 Montgomery Ladder for ECC .................................. 43
    4.2.3 Joye-Tymen Base Point Randomization ....................... 45
  4.3 Discussion ............................................................ 48
Chapter 1

Introduction

Wireless sensor nodes (WSN) are mobile platforms with limited computational power that are used for a variety of experimental purposes. This includes collection of sensory information (such as temperature, humidity, etc.) and a minor amount of data processing. They generally consist of a radio, sensors and external memory connected to a micro-controller. Given that the most powerful devices on the market have a 16-bit processor that runs at 25MHz, with 16 KB of RAM, it is not difficult to imagine that having an efficient implementation is the name of the game.

In practice, WSNs are used as part of a large failure-tolerant distributed network and cannot easily be monitored on an individual basis. Without arousing suspicion, evil-doers may take advantage of this to spirit them away to a lab for in-depth analysis. One popular method of gaining access to privileged information is to employ side channel attacks; a super-class of approaches by which ‘secret’ information can be gained through weaknesses in the actual implementation of a cryptosystem. This includes weaknesses in the hardware it is implemented on, where information may leak from channels not modelled by cryptography. The most interesting of these attacks are those that analyze the power consumption of a device during cryptographic computation.

Secure communication on power and computation-restricted nodes is therefore of the highest priority. The unusual circumstances under which these programs must run make the problem all the more difficult. As a result, only a handful of cryptographic protocols have been implemented in practice on WSNs. In this paper, these implementations and the environment in which they operate will be evaluated in the context of side channel attacks (specifically, power analysis), and solutions will be offered that aim to increase the immunity against said attacks.
1.1 Motivation

Since the inception of Public Key Cryptography in the mid-70's by Diffie and Hellman, the field has experienced rapid progress and development. Major discoveries have happened in every decade hence, with the 70's additionally producing the RSA algorithm by Rivest, Shamir and Adleman. As the field of cryptography evolved, as did cryptanalysis; the art of breaking cryptographic protocols. PKC is based on the idea that there are functions that are easy to compute in one direction, but hard to undo. In the case of plain PKC, this may be the discrete logarithm problem, where it is easy to compute $a^x = b$ given $a$ and $x$, but significantly tougher to find $x$ given $a$ and $b$.

With the discovery of Elliptic Curve Cryptography by Kocher in 1985, entirely new avenues of research have opened up. The fundamental advantage is that the discrete logarithm problem is significantly more difficult for elliptic curves than for regular PKI, meaning that a smaller group order can be used to obtain the same amount of security. The use of smaller groups and simpler operations means that it becomes feasible to implement cryptographic protocols on constrained devices, neither compromising speed nor security.

1.2 Problem Statement

One month after this Master’s project began, a paper was published in China at the Second International Symposium on Electronic Commerce and Security which discussed the use of Simple Power Analysis to break ECC cryptosystems. This feasibility study was done to provide conclusive experiments where actual elliptic curve cryptosystems were successfully attacked. The double-and-add-always method and Montgomery method for point multiplication were studied and were shown to be secure against simple power analysis [49]. In our work, we aspire to go one step further, by studying a variety of real-world implementations of both DL and ECC cryptosystems. Additionally, we provided practical solutions to protect against SPA and template attacks.

The goal of this Master’s thesis is to evaluate different academic implementations of cryptographic algorithms for sensor nodes and test their resistance to simple power analysis and differential power analysis. Three software packages which will be studied in-depth are TinyPK, EccM and TinyECC, representing an RSA PKI, an ECC implementation, and an optimized Elliptic Curve Digital Signature algorithm, respectively.
1.3 Related Work

The cornerstone of the field of power analysis was laid by Paul Kocher in 1998 when he published his seminal work in collaboration with Joshua Jaffe and Benjamin Jun. Therein, the concepts of simple and differential power analysis (SPA and DPA respectively) were introduced. As far as implementations are concerned, the analyses presented in this thesis would not have been possible were it not for the research done by the team at Berkeley who developed TinyOS [13] – a framework for programming WSNs – and the subgroup who worked on TinySec [20] – the first link-layer security architecture for wireless sensor networks. Minisec [25], a more advanced link-layer crypto library, was developed as an extension of TinySec in 2004 at Carnegie Mellon University.

On the application layer, TinyECC [24] is a rather comprehensive elliptic curve cryptographic library developed with WSNs in mind. Along with TinyPK and EccM, it formed the basis of my in-depth review. The second part of the thesis is based on the works of Hedabou and Pinel [12], Chevallier-Mames [5] and Liardet [23], using their suggestions on how to prevent SPA attacks. Additionally, the source code of TinyECC proved useful during the implementation phase.

1.4 Overview

The purpose of this thesis is to delve into the field of Side Channel Analysis of cryptographic algorithms, to experiment on current implementations and to mitigate weaknesses when they are discovered. The structure is as follows:

- In Chapter 2 cryptanalysis will be introduced, and the hardware platforms will be discussed. The measurement infrastructure will also be described. Furthermore, we will study the techniques used to attack the devices, including attacker model used and assumptions made.

- Following that, in chapter 3, we will analyze the chosen cryptographic implementations on the look-out for weaknesses which may expose privileged information.

- Chapter 4 will offer solutions to these problems in EccM and TinyPK, while discussing the difficulty in breaking the TinyECC.

- In the final chapter 5, we will summarize the lessons learned.
Chapter 2

Background

Before discussing the details of specific implementations, it is important to cover the fundamen-
tals. The purpose of this thesis is to analyze cryptographic libraries and to provide modifications
to said libraries in order to protect them from side channel attacks attacks. We begin by in-
troducing the basic techniques, including cryptography and cryptanalysis. Then the hardware
platforms are presented, after which we discuss the libraries to be analyzed, and the algorithms
they implement.

2.1 Cryptography

Cryptography is the mathematical study of techniques related to aspects of information secu-
rity [32]. Amongst its objectives are confidentiality, integrity, authentication and non-repudiation.
To put it simply, cryptography tries to ensure the secure and trusted exchange of priviledged
information in a hostile environment. Many tools have been developed to solve this complex
problem. The ones relevant to our discussion are based on symmetric and asymmetric-key
primitives.

2.1.1 Symmetric Cryptography

Symmetric cryptography is based on the use of a secret key which is shared between com-
municating parties. Mathematically, such a one-key cryptographic system is a tuple \((E_e, D_d, m)\)
where \(E\) and \(D\) are the encryption and decryption functions, \(e\) and \(d\) are the encryption and
decryption key, and \(m\) is the message. Conventionally \(e = d\), which begs the complex issue
of secure key distribution. Several of the cryptographic implementations studied in this paper,
make assumptions that some appropriate method has been used to predistribute the keys, but
this issue is not one which will be dealt with in depth.
2.1.2 Asymmetric Cryptography

Asymmetric cryptography uses a private and public key to encrypt and decrypt a message. These systems have four fundamental properties:

1. Given an encryption function $E()$, decryption function $D()$ and message $M$, $D(E(M)) = M$.
2. Conversely, encrypting the decryption of $M$, gives $M$: $E(D(M)) = M$.
3. The encryption and decryption functions are easy to compute.
4. Revealing $E()$ does not reveal anything about the decryption function.

Since symmetric cryptography has been utilized in TinySec, many developers have taken it upon themselves to implement asymmetric cryptography solutions to secure communication.

2.2 Cryptanalysis

Cryptanalysis describes techniques by which encrypted messages from a cryptographic device can be decoded without knowledge of secret information. More obvious methods such as social engineering are laid aside, and the focus is put on mathematical analysis and side channel attacks.

2.2.1 Side Channel Attacks

![Diagram: Data lost through side channels can be used to compromise a cryptographic system](image)

**Figure 2.1**: Data lost through side channels can be used to compromise a cryptographic system

Side channel attacks are a class of techniques used to compromise a hardware device by means of analyzing side channel information such as time measurements and power consumption. In the context of cryptographic systems, it was believed for the longest time that they were a veritable black box, providing no other information that the plain-text input and the encrypted output. The two possible attack methods, using this information are called plain-text attacks and
ciphertext-only attacks, respectively. Timing information and power consumption, on the other hand, are unintentional sources of data, which are by definition tightly correlated with hidden internal state of the system.

For the purposes of this investigation, we will be studying the types of attacks which use the power emissions of a system running cryptographic computations to derive secret information.

### 2.2.2 Simple Power Analysis

Simple Power Analysis (SPA) is a visual interpretation of the power trace of a system [21]. Using again the example of DES encryption, the sixteen rounds of execution – corresponding to 16 executions of the Feistel function – can clearly be seen. By visual inspection, this may allow the secret key to be deduced. Furthermore, a naive RSA implementation based on the square-and-multiply method, can be broken with SPA by observing patterns in the number of peaks in power consumption, caused by branching in the code 2.1.

If RSA exponentiation is implemented exactly as described in the pseudocode, it would imply that two multiplications would happen if $e(j) = 1$ and one multiplication would happen otherwise. This result can be easily validated against the power plot.

Even more complex cryptographic protocols such as AES may be defeated by SPA. In the figure 2.2, the annotated power trace of an execution of AES clearly indicates how the private key influences the execution of the algorithm.

![Figure 2.2: Another example of simple power analysis using the power trace of the execution of AES [29]](image-url)
2.2.3 Differential Power Analysis

Differential Power Analysis (DPA) is a significantly more powerful attack than SPA, while also is more difficult to prevent. This method uses statistical analysis over multiple experimental runs to extract sensitive information. The process is split into two phases: data collection, and data analysis. Reusing the example of DES from the above section, if we study the cumulative executions of this function on varying input we may be able to derive parts of the key. This is done using a function of the form $D(k, c)$ where $k$ is some key information and $c$ is the ciphertext [21].

The power consumption of the device can be modeled by a function, which is the sum of a 2-dimensional random variable with an unknown distribution. Yet, using the Central Limit Theorem, if all possible random variables are chosen independently, at random, this implies that our function has a normal distribution. Consider each random variable as a power trace, then these random variables are grouped into sets (based on hamming weight), which will be used to verify the prediction of the secret key.

Using the example of DES again, the outcome of $D()$ depends on the output of subroutines of the algorithm. By running the experiment thousands of times while permuting the key information, an average trace can be constructed using the output of $D()$. The resulting plot will show what effect the changing input had in the average power consumption of the routine, which – given detailed knowledge of the construction of the cryptographic algorithm in use – may divulge secret key information which can then be used to compromise a wireless sensor network 1.

2.2.4 Template Attacks

Resting between SPA and differential power analysis, are templating attacks. This attack implies the use of fingerprints (or templates) which describe an execution based on a particular private exponent. As opposed to DPA, template attacks have a corser resolution. The signal as a whole is considered and true feature extraction does not occur. Instead, several traces are averaged, and then compared to another set of traces using some distance measure (in our case, regular geometric distance $d = \sqrt{(x_2 - x_1)^2}$). If we prepare our samples properly, this technique will provide some insight into the possibility of success of DPA.

2.3 Hardware Platforms

Sensor nodes are small wireless devices which are designed for use in distributed systems. Two of the most popular devices are the Mica and Telos series, on which the majority of platforms have been developed.

1 Methods for preventing side channel attacks of this nature have been patented by Cryptography Research Inc.
2.3 Hardware Platforms

2.3.1 Mica2

The Mica2 Wireless Measurement System is a wireless platform for smart sensors. It is optimized for very low power operation, with a life-time of up to one year when powered by two AA batteries. It has a current draw of about 16 $\mu$A in sleep mode and maximally 35 mA when both the microcontroller and the CC1000 radio are active [7]. To achieve these extraordinary figures yet maintain functionality, the processor must be able to spend most of its time in sleep mode with a minimal wake-up time. The 8-bit ATmega128L microcontroller achieves this, and more, operating at 8MHz [46].

![Figure 2.3: Block diagram of the Mica2 wireless sensor device](image)

The ATmega128L has 128 kilobytes of flash memory [1]. Additionally, it has 4096 bytes of EEPROM, 4096 bytes of SRAM and has an input voltage of 3V. The amount of memory available severely limits the how much data may be stored for precomputation tasks (which would be used to cloak our side-channel output). These memory constraints must remain foremost in consideration when writing code for the device.

2.3.2 TelosB/TMote Sky

Similar memory restrictions exist on TelosB nodes. It runs an MSP430F1611 microprocessor from Texas Instruments at 8 MHz, with 10 kilobytes of RAM and 48 kilobytes of flash memory $^2$. It has up to 1024 kilobytes of storage space (as opposed to the 128 kilobytes of the Mica2). The most useful capabilities of this board might prove to be the additional output pins (both a 6 and 10-pin connector), which can be individually set to High or Low. This may be used as an external trigger for the oscilloscope when gathering measurements $^3$.

$^2$http://www.capsil.org/capsilwiki/index.php/TELOS/TMote_Sky
2.4 Cryptographic Algorithms

2.4.1 RSA

RSA is a public key cryptography algorithm which was named after its creators: Rivest, Shamir and Adleman at MIT in 1987 [38]. It is based on the work of Diffie and Hellman in 1976, expanding the idea of public key cryptography to an encryption scheme that also allows digital signatures. RSA is based on the difficulty of factoring a number which is the product of two primes, while the inverse operation – finding primes, and multiplication – is easy. Layered on top of that are a fact and a conjecture which are used to construct the encryption and decryption operations. Firstly, modular exponentiation is easy to do, and secondly, modular root extraction is hard.

The algorithm begins with the generation of two large, random prime numbers and computes the product of these numbers. The result gives the group order in which further calculations are done. Subsequently, a public exponent must be chosen which is between 1 and the totient of \( n = p \times q \). This value can be freely exposed. The private exponent is calculated such that \( de \equiv 1 \mod \varphi(pq) \). The encryption function is a result of raising the message to the public exponent. The recipient then raises the encrypted message to the private exponent \(^4\).

Exponentiation is done using the square and multiply method. We may be able to take advantage of weakness presented in the power analysis example in the previous section.

\(^4\)http://www.nku.edu/~christensen/themathematicsoftheRSAcryptosystem.pdf
2.4 Cryptographic Algorithms

2.4.2 ECC

Elliptic Curve Cryptography (ECC) is a new cryptographic method independently discovered by Koblitz and Miller in 1985. It is touted as a major breakthrough in the field by many researchers. Since 2005, the U.S. National Security Agency (NSA) has gone so far as to include it in the Suite B encryption group, good for use in protecting both classified and non-classified information. ECC is a type of Public Key Encryption (PKE), and like the first generation of encryption schemes, is based on a variant of the Diffie-Hellman (DH) assumption. While standard PKC relies on the fact that computing a discrete log is hard, ECC depends on the idea that it is computationally difficult to find an integer \( k \), given points \( Q \) and \( P \) on an elliptic curve such that \( Q = kP \). This is the operation you need to execute to derive a plain-text from the cyphertext using only public information (the public key).

ECC is based on elliptic curves which have the form \( y^2 = x^3 + ax + b \). This curve must be defined in a large prime field, and the integer coordinates the curve passes through form a group. There exist recommended curves to use in prime fields as well as in \( 2^m \). The fundamental operation in ECC is point multiplication, where one needs to calculate \( Q \) (public key) in \( Q = kP \) where \( k \) (private key) and \( P \) are given. While brute force is the only way to do the reverse operation, there exist algorithms which allow you to run point multiplication efficiently.

ECC is preferential to other cryptographic building blocks because while the forward operations are light enough to be computed on low-power devices, the reverse operations are significantly harder than those of standard PKE. For example, while it is recommended to have an RSA key of size 2048 bits, to get equivalent security using ECC you only need a key of size 224 bits. As computer systems grow more powerful, the inherent advantage of ECC will become more apparent.

2.4.3 ECDSA

ECDSA is a form of the Digital Signature Algorithm based on properties of elliptic curves, rather than the discrete logarithm problem. A digital signature is a value computed from a secret value and a message, and was invented as a modern replacement to the handwritten signature. ECDSA is an asymmetric signature scheme with the assurance of unforgeability with chosen-

---

\[ \text{http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml} \]

\[ \text{Of the form } \log_b: G \rightarrow Z_n \]

\[ \text{That is to say, you can apply group operations to any element of the group and the result is another element in the group. Associativity, identity and invertibility properties also hold.} \]
message attack. This means that the signature, and the message from which it is derived are independent from one another [15, 4-26].

Listing 2.3: Elliptic Curve Digital Signature Algorithm

```plaintext
begin
    e = hash(m) /* Calculate the Hash of the message */
    1: Rand(k) mod n /* Choose a random k */
    r = x1 mod n
    if r == 0 then
        GOTO 1
    s = k-1*(z + r*d) mod n mod n /* Exponentiation as in the previous examples */
    if s == 0 then
        GOTO 1
    return (r, s)
end
```

ECDSA is derived from DSA by replacing the subgroup of order $q$ in $\mathbb{Z}_p^*$, with the subgroup of points on an elliptic curve created with generator $G$. In the case of TinyECC, domain parameter generation is done in advance on a powerful (desktop) computer, while the signature generation happens in situ on the low-powered device.

### 2.5 Cryptographic Implementations

#### 2.5.1 TinySec

TinySec [20] is a link layer security architecture for wireless sensor networks [20] which has been integrated into TinyOS since v1.0. TinySec offers data authentication and (optionally) encryption. Data is authenticated with CBC-MAC, and encrypted using the Skipjack block cipher. Although Skipjack has been compromised, the impact on embedded devices is minimal due to the slow data link, and low processing power of both Mica2 and TelosB motes.

The Skipjack algorithm uses symmetric key cryptography. The problem again becomes efficient and secure key distribution, and secure storage of keys on the device. Naturally, symmetric key cryptography implies pair-wise keys for every node you communicate with, so this also quickly becomes a storage issue. Wireless motes have severely limited local memory, which is further reduced by the size of the running application and the operating system, so another more efficient method of encryption needs to be found. TinySec solves this problem by limiting the communication to one group of nodes.

As a link-layer architecture, this means it is relatively straight-forward to integrate TinySec into any TinyOS application. Adding the argument `TINYSEC=true` when calling the makefile, activates Authentication-Only TinySec. For TinySec-AE a single key is hardcoded into the application when it is compiled [42], which means this sensitive data will be sitting in device memory. When sending data, the key is read from memory and is used when running the SkipJack or

---

8It has been estimated that it would take ~20 months to break [20], using the provided 200 kbps radio link. For one example of the weakness of Skipjack, see: [http://www.cs.technion.ac.il/~biham/Reports/SkipJack/note1.html](http://www.cs.technion.ac.il/~biham/Reports/SkipJack/note1.html)
2.5.2 MiniSec

MiniSec [25] improves upon TinySec by offering a high level of security, while simultaneously keeping energy consumption low. More specifically, MiniSec goes one step further by protecting against replay attacks, and by using per-device keys (instead of group keys) [25]. As a result, it provides secrecy, authentication and replay protection. Assumptions are similar to those of TinySec: for example, since symmetric keys are used, the problem of key distribution is ignored.

As for the attacker model, Dolev-Yao was used [4], wherein an attacker can intercept, alter and inject messages into the communication channel. This is considered to be one of the most powerful attacker model.

At each corner, MiniSec goes one step further than TinySec. Where TinySec uses CBC-encryption with a repeating counter, MiniSec uses OCB-encryption with a non-repeating counter. Also using Skipjack as the encryption algorithm, the block size has been increased to 80 bits, so as to remain flexible.

2.5.3 TinyPK & EccM

This project has completed an implementation of RSA on Mica2 motes. TinyPK [48] is supposed to be used in combination with Tinysec on the use of TinySec and enables mutual authentication without a preshared key. Resource-heavy private RSA operations are outsourced to a powerful computing device, while public operations are done on the mote. Although the implementation on the mote was generic, the exponent has been set to three for experimental purposes. The public-key protocols allow authentication and key agreement between a sensor network and a third party as well as between two sensor networks. Authentication is done based on Diffie-Hellman key agreement protocol.

The EccM [27] application is remarkably similar to TinyPK, except for the use of a different cryptographic foundation. TinyPK is built on the discrete logarithm problem, while EccM is based on the intractability of elliptic curve operations. This conceptual change, means that instead of dealing with single values, we are now computing with points in a 2-dimensional field. In fact, Malan et al. borrow byte level multiplication, subtraction and modulus functions from TinyPK.

---

9Groups are used to introduce namespaces into TinyOS communication, allowing data transfer between more than two devices.

10OCB is a blockcipher method which provides both privacy and authenticity. It has been shown to be more efficient than encryption with conventional methods.
2.5.4 TinyECC

TinyECC [24] is a comprehensive ECC toolkit for mobile sensor devices. It provides a digital signature scheme (ECDSA), a key exchange protocol (ECDH), and a public key encryption scheme (ECIES). Additionally, a number of switches are provided to toggle specific optimizations at compile-time based on developer’s needs. Some of these optimization techniques are optimized modular reduction using a pseudo-Mersenne prime, the sliding window method for modular exponentiation, the use of Jacobian coordinates for base operations, inline assembly and hybrid multiplication to achieve computational efficiency. In effect, this application, which contains dozens of implementation tricks, will be very useful when it comes time to offer countermeasures protecting against various power analysis attacks.

2.6 Attacker Model

We assume the existence of an attacker who possesses a physical duplicate of the device on which the cryptographic software is installed. We also assume we have the source code of the device, and have the ability to modify the the private key on our own device for the purpose of divining the hidden information. This type of exploit is known as a multiple-exponent single-data (MESD) attack. For the same input text as the target node, the power consumed by the calculations for both known and unknown exponents are compared. If resulting outputs are the same, then the exponents are equivalent too.
Chapter 3

Practical Attacks

Figure 3.1: Diagram of experimental setup showing mote, amplifier, DC power supply and resistance

This chapter is divided into an analysis and an implementation section. We begin with platform and code preparation, as described in appendices B and C, followed by code analysis. Gaining an intuitive feeling about the execution of a program will allow us to identify relevant sections of code to single out for further study.

Table 3.1 shows the libraries that will be discussed in this section. The ‘protection’ column lists potential strengths in design of the algorithm which might complicate our efforts to break the system.

The chapter is continued by constructing an experimental setup as described in figure 3.1. As for the technical details, the voltage drop over the 1 Ohm resistor is stabilized, amplified by 60 dB and then passed through a 20 MHz lowpass filter, before reaching the oscilloscope. A DSA90804A Infiniium oscilloscope was used, along with a Femto DHPVA Voltage Amplifier.
Table 3.1: Properties of cryptographic libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Platform</th>
<th>Primitive</th>
<th>Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>EccM</td>
<td>Mica2</td>
<td>Elliptic Curve DH(^1)</td>
<td>None (Binary Algorithm)</td>
</tr>
<tr>
<td>TinyPK</td>
<td>Mica2</td>
<td>Discrete Logarithm DH(^1)</td>
<td>None (Binary Algorithm)</td>
</tr>
<tr>
<td>TinySec</td>
<td>Mica2</td>
<td>Symmetric Key Block Cipher</td>
<td>IV as part of key</td>
</tr>
<tr>
<td>TinyEcc</td>
<td>TelosB</td>
<td>Elliptic Curve DSA(^2)</td>
<td>Windowing Technique</td>
</tr>
</tbody>
</table>

\(^1\) Diffie Hellman Key Exchange  
\(^2\) Digital Signature Algorithm

Standard BNC cables were used to feed the voltage signal to our measurement device\(^1\). The findings are presented in section 3.2.

### 3.1 Code Analysis

Prior to starting the labwork, time was spent understanding the code processes. This gives us insight on the precise data-flow of the library under investigation.

#### 3.1.1 TinyECC

Each experiment began with a perusal of the source code to achieve an understanding of the program flow. The part of the TinyECC application which was studied consisted of the files `AliceM.nc`, `NNM.nc` and `ECDSAM.nc`. The Elliptic Curve Digital Signature 2.3 algorithm is described, which used an exponentiation technique similar to that in other applications. `AliceM.nc` provides one half of a transcript with `BobM.nc`, and describes a scenario where the public key has been pre-shared. Alice broadcasts a signed packet, which Bob verifies\(^2\). Since it has been shown that the Digital Signature verification algorithm is generally not susceptible to side channel attacks, Alice was given in-depth analysis. The program flow is described in figure 3.2.

The initialization function, initializes the random number generator as well as the LEDs. When the application starts `init_data()`, clears all messages, and sets the private exponent and the public point. The ECDSA module is also initialized before calling a 10-second timer, which will allow any 'posted', that is to say asynchronous, tasks to complete. When the timer fires, the `sign()` function is called, where a random message is generated, the message is signed, the packet is prepared and eventually sent.

There are some inline assembly code in `NNM.nc` to speed up natural number operations. These inline assembly code are written in AVR instruction set and are therefore optimized for the

\(^1\)The disadvantage compared to using calibrated probes was not observable.
Mica2 node. This feature was disabled by toggling the `#define INLINE_ASM` switch in NN.h to allow compilation for the TinyOS simulator. The exponentiation function is hidden away within the Natural Numbers module, several layers down from the application module.

3.1.2 TinyPK

The TinyPK application, consists of a file called `DHm.nc` wherein a Diffie-Hellman protocol is described for two nodes to establish a shared secret. The program flow is described in figure 3.3.

The modulus and basepoint have already been hardcoded in `DH.h`, and have therefore been initialized on the stack. During the initialization of the device, the private exponent is
generated by successive calls to the Random module. After the private exponent has been set, the Exponentiate() function is called to find \texttt{basenum768} to the basesize modulus \texttt{modulus768}. When the exponentiation function returns, the communication controller is initialized, and the \texttt{StdControl.start()} function is called. Here the data packet (potentially multiple packets) is prepared and sent.

The exponentiation function accepts the input of various values as pointers to the most significant byte of a sequence of bytes. On the ATmega128L processor, \texttt{uint8\_t} creates an 8 bit unsigned integer, which adds a layer of complication, as it would have been easier to work with a sequence of Boolean values. Instead, we move through the sequence of bytes from MSB to LSB using an inner and an outer for loop. The outer loop decrements the pointer address (by \texttt{sizeof(uint8\_t)}) and the counter variable (from \texttt{datasize-1} to 0). The inner loop uses an 8 bit mask initially set to \texttt{0b1000000}, which is right-shifted until it reaches 0. In this way, each bit of the exponent is dealt with in turn.

The meat function is a right-to-left square-and-multiply algorithm as described in [ref], where squaring is always done, but you only multiply if the current exponent bit is 1.

Asynchronously, the receive function may get called when a message is passed up from the network layer. In this case, the value being received represents the private key of the communication partner. If a mote has both received a value, and sent its value then it calculates \texttt{remote^{local} \mod \texttt{modulus768}}.

In summary, we expect to see the computation taking place in the Exponentiate() function quite clearly, due to the straightforward implementation.
3.1 Code Analysis

3.1.3 EccM

The EccM application, consists of a file called EccM.nc wherein a Diffie-Hellman protocol is described for two nodes to establish a shared secret using elliptic curves. The program flow is described in figure 3.4.

Divergent from TinyPK, the modulus and basepoint are set in StdControl.start(). A three second timer is set, upon which the private key is randomly generated, and then used to generate the public key by exponentiation ($m^k$). After the exponentiation function returns, a communication timer is initialized, upon which the data packet is prepared and sent. Although it has been removed in the minimal implementation, after the message has been sent, and the message has been received from the communication partner, generate_secret() generates the the shared secret.
TinySec is a link layer tool for providing privacy and authenticity for communication. It has been integrated into the radio stack. In the earlier version of TinyOS which was in use, implementations for the CC2420 radio found in TelosB had not yet been completed, so we ran the test application on the Mica2 node. TinySec has a complicated code path through several different modules, many of which have aliases, complicating the analysis.

Most important for our analysis are the rounds of calls to `BlockCipherMode.encrypt()`. A potential weakness in the CBC mode block cipher encryption is that it was designed to be used with random IVs, yet TinySec uses the IV as a counter. The problem is solved by encrypting the IV in advance and then using the encrypted IV to initialize the block cipher. The IV consists of 2 bytes which increment starting from 0, 2 bytes which contain the source address of the sender, and 4 bytes of randomness.
3.2 Exploits

In this section we practically analyze TinyECC, TinyPK, EccM and TinySec, and present our findings regarding simple power analysis and template attacks. Additionally, we provide a direction for further analysis with DPA. In the recordings made, we set the trigger to 900 mV corresponding to the peak of the LED initialization phase. This suited our purposes quite well as it, in effect, solved our alignment problem.

3.2.1 TinyECC

We began by connecting the mote to our oscilloscope to get an idea of how the power trace looked. After our initial code analysis, we knew what to expect. As with the other implementations, in preparation for this stage the radio was turned off and the code was minimized so as to remove unnecessary debugging and messaging operations as described in the appendix C.3.

![Labeled oscilloscope hardcopy of an execution of TinyECC 0.3 ECDSA, using horizontal division = 5 seconds, vertical division = 1.802 mV, 1-value averaging](image)

Figure 3.6: Labeled oscilloscope hardcopy of an execution of TinyECC 0.3 ECDSA, using horizontal division = 5 seconds, vertical division = 1.802 mV, 1-value averaging

When the significant portion of the screen was magnified, it was immediately possible to see the individual operations taking place. Nonetheless, even though the operations could be seen, this does not necessarily imply that the information is useful, because a windowing method is used to mask the private exponent. That being said, it has been shown that it is possible to break the sliding window method using the values gleaned from differential power analysis to backtrack and reveal the original hidden values [31]. Also interesting to note is that the fundamental addition operation is done in 3-dimensional affine coordinates (with the Z coordinate set to 0). This
allows the addition and multiplication functions to be implemented more efficiently. The fact that this framework already exists might simplify later efforts to implement basepoint randomization as a way of defeating differential power analysis.

![Figure 3.7: Labeled detail of oscilloscope hardcopy (from figure 3.6) of an execution of TinyECC 0.3 ECDSA showing square-and-multiply operations, using 1-value averaging](image)
3.2 Exploits

3.2.2 TinyPK

TinyPK is a much more straight-forward application, as was described in flowchart 3.3. In the full-length operation there is an initialization phase, and 12 rounds of exponentiation. Clearly, we can see 8 bits being processed, followed by a peak caused by the outer for-loop. This information has been plotted on figure 3.8. When reconstructing the key, it is important to remember that the bytes are processed from most to least significant bit.

![Oscilloscope hardcopy of an execution of TinyPK showing 12 rounds of exponentation](image)

**Figure 3.8:** Oscilloscope hardcopy of an execution of TinyPK showing 12 rounds of exponentation

In order to get a better picture, we averaged several execution runs. In figures 3.9(a) and 3.9(b), the individual operations are made visible.
(a) Matlab plot (from figure 3.8) of the first six rounds of exponentation of TinyPK (average of 10 runs)

(b) Detail of Matlab plot of exponentation (from figure (a)) of TinyPK showing squaring-only and square-and-multiply (average of 10 runs)
3.2 Exploits

3.2.3 EccM

From the appearance presented in figure 3.9, the EccM program execution looks nebulous, but when the signal is magnified, it is still possible to see two distinct shapes, representing the doubling and the adding operations. These are displayed in figure 3.10, where the type of operation, and the resulting private key has been labeled. Figures 3.11(b) and 3.11(a) show the two types of operations that exist, with an overlay of the rough sketch of the filtered signal. It is quite apparent that this library can be broken with simple power analysis. In later stages, masking techniques may be used to protect against this.

Figure 3.9: Averaged oscilloscope hardcopy of an execution of EccM, using horizontal division = 5 seconds, vertical division = 500 mV, 2 kSa/s
Figure 3.10: Detailed oscilloscope hardcopy of an execution of EccM showing an addition followed by four doubling operations

Figure 3.11: Comparison of adding (b) and doubling (a) operation in EccM
3.2.4 TinySec

As with the previous applications, the radio itself provides a significant amount of masking, to the extent that the original signal is not at all visible. When the radio was turned off, the only distinctive features which appear are the 20ms ticking of the DC power supply. To encrypt a packet with TinySec takes less than 5ms, which means that the middle two peaks in figure 3.13 must be analyzed.

Figure 3.12: Oscilloscope hardcopy of an execution of TinySec, radio turned on, using horizontal division = 50 ms, vertical division = 100 mV

Figure 3.13: Oscilloscope hardcopy of an execution of TinySec where only the amplified ticking of the power supply is visible, radio turned off, using horizontal division = 50 ms, vertical division = 2.0 mV

This does not mean that it is impossible to break the implementation. Figure 3.15 shows one
way in which information is leaked about the secret key, although the primary indication may be misleading. The relevant piece of information which may be gleaned from this graph is that a private key with a higher hamming weight (in this case, the blue trace) results in higher energy consumption for the device, ceteris paribus. Some information about the Hamming weight of the private key is leaking into the side channel, which means there exists a potential to break the system. Unfortunately, any other way of using power analysis is thwarted due to the low power consumption of the system. Looking at the active section of execution, as in figure 3.14, we see nothing but noise.

**Figure 3.14:** Oscilloscope hardcopy of an execution of TinySec where one encryption operation is visible, radio turned off, using horizontal division = 1 ms, vertical division = 50 mV

**Figure 3.15:** Matlab plot of two executions of TinySec (average of 10 runs). Blue: private key alternates ones and zeros, Red: private key all zeros
3.3 Discussion

Careful analysis helped simplify the process of finding exploits using power analysis in the case of TinyPK, TinyECC, TinySec and EccM. Weaknesses were found in EccM and TinyPK which allow us to reconstruct the secret key. In the TinyECC ECDSA implementation, we were also able to see fundamental operations, but were not able to derive the key information from it with the techniques we had available. With differential power analysis, it would be possible to decide which precomputed value was used in each step of exponentiation. Therefore, using this method, when \( w = 2 \), we would be able to derive 2 bits of the key per execution loop. Lastly, the link-layer TinySec application was unable to be broken, due to extraordinarily low power consumption. Nonetheless, side-channel leakages have been found which may be exploited with more advanced techniques.

The results gathered show that it is possible to efficiently break existing cryptographic implementations using power analysis attacks. Based on the work of this chapter, in chapter 4 we will implement and test various protective measures for the aforementioned crypto-libraries.
Chapter 4

Proposed Countermeasures

(a) Raw power trace of TinyPK from oscilloscope, 200 kSa/sec
(b) Filtered power trace of TinyPK, using filter length of 1025 samples and a cutoff frequency of 40 Hz

Figure 4.1: Before (a) and after (b) shot showing the improvement brought on by use of low-pass filtering

In this chapter modifications to TinyPK and EccM are proposed and implemented in order to protect against simple power analysis. These new libraries are then tested in the lab and the results are discussed. If they are indeed shown to be secure, then template attacks are applied as an example of more complicated exploit. In this case, filtering becomes very important. While in the last chapter, it was often possible to derive the key without modifying the voltage trace, now we may need to clean the signal in order to create coherent templates. As shown in figures 4.1(a) and 4.1(b), it is possible to greatly improve the quality of the plot. In turn, TinyPK and then EccM will be presented.
4.1 TinyPK

4.1.1 Square-and-Always-Multiply

The Square-and-Always-Multiply method was derived from the background of Johnson and Menezes’s paper [15], and aims to prevent simple power analysis on modular exponentiation. The downside is that on average it takes 1.5 times as long as regular exponentiation. In fact, the worst case time complexity of regular exponentiation (namely, a private exponent consisting solely of 1s) becomes the average case. From the listing 4.1, we can see that in every loop both a square and a multiply operation are performed. In the case where the private exponent bit is 0, multiplication is performed with the same values, but stored in a dummy variable [16].

```
1 begin d, B
2 Q[0] = B
3 Q[1] = 1
4 for i = k-2 down to 0
5   Q[0] = Q[0] * Q[0] /* square */
6   Q[d[i]] = Q[d[i]] * B /* multiply */
7 return Q[0] /* B^d */
8 end
```

Listing 4.1: Square-and-Always-Multiply Method for Exponentiation in TinyPK

As we can see from figure 4.3, every square operation is now followed by a multiplication operation ¹. From the perspective of SPA, the entire private key consists of ones, and there are no distinguishing markers in the signal which would help distinguish one block from the next. In figure 4.2, we see that when comparing the execution of two different keys, the graphs sync

¹As far as macro plots in this section are concerned, unless otherwise mentioned, they have been recorded at 200 kSa/sec (meaning $5 \times 10^{-6}$ seconds per sample).
up quite well. In fact, while in the diagram it appears to be the case that the green trace has considerably less energy than the blue trace, this only has to do with the resolution of the captured image. From figure 4.4, we see that the expected divergence at byte 5 has not occurred. On the other hand, it appears that the signals vary greatly from one another in some way which is not immediately obvious from the previous illustrations. Here it would be useful to use DPA techniques to check the resiliency of the system. We expect the Always-Multiply method to fail at that stage.

**Figure 4.3:** 6-averaged, filtered power traces of TinyPK showing multiplication and squaring operations

**Figure 4.4:** Plot of difference of two 6-averaged, filtered power traces of TinyPK showing a divergence in the 5th byte
4.1.2 Montgomery’s Ladder Technique

In order to implement the Montgomery’s Ladder technique, it was necessary to transform the Exponentiate() function from left-to-right to right-to-left order. The technique was described in the paper by Joye and Yen [18], and in the seminal work by Montgomery [34].

The Ladder is based on several mathematical observations. In solving the problem of computing \( y = g^k \) in group \( G \), we take into considering the binary expansion of the private exponent \( k \). This is represented as \( \sum_{i=0}^{t-1} k_i 2^i \). We can define an \( L \) such that \( L_j \) is the value of the \( t-j \) lowest bits.

\[
L_j = \sum_{i=j}^{t-1} k_i 2^{i-j}
\]

If we also define \( H_j \) as \( L_j + 1 \), then a statement can be made expressing \( L_j \) in terms of \( L \) and \( H \). This leads to two expressions for the case that \( k_j = 1 \) and \( k_j = 0 \). These expressions can be used to exponentiate a value in DL or even ECC cryptographic systems. While protecting against SPA, the Montgomery Ladder also leads to a 33% speed up factor compared to the normal multiplication method.

Listing 4.2: Montgomery’s Ladder Technique for Exponentiation in TinyPK

```
begin d, B
Q[1] = B
Q[0] = 1
for i = 1 up to k-1
  if (d[i]==0)
    Q[0] = Q[0] * Q[0] / * square */
  else
    Q[0] = Q[0] * Q[1] / * multiply */
end if
return Q[0] / * B^d */
end
```

In figure 4.6, we see a detailed of figure 4.5. By visual inspection, it is clear that the figures align quite well in frequency, amplitude and phase, excepting the small highlighted areas. This discrepancies may be due to the experimental setup. As evidenced by figure 4.7, the two voltage trace-templates are highly similar, although two different private exponents were used. This implies that the Montgomery method for multiplication is quite stable and secure against simple power analysis (including templating attacks). It has been shown though, that this method is vulnerable to true differential power analysis. An attack has been proposed by Koichi and Tetsuya in [22] which extends the work by Messerges et al. on data-bit DPA.

Additional modifications were made, which – as can be seen in figure 4.7 – contributed to the regularity of the voltage trace. For example, care was taken to use the same function call for multiplication as for squaring, as the squaring function was optimized for the given scenario and therefore executed faster.

\(^2\)The Koichi Tetsuya method can also be used to break the ECC double-and-add algorithm.
**Figure 4.5:** Superposition Plot of Exponentiation with Montgomery Multiplication, 6-averaged, filtered, Green: 77776177777777, Blue: 77777177777777

**Figure 4.6:** Plot of Exponentiation with Montgomery Multiplication, 6-averaged, filtered, highlighting suspicious differences
Figure 4.7: Plot of difference of Exponentiation with Montgomery Multiplication, 6-averaged, filtered, showing high similarity
4.1.3 M-Ary Multiplication

We consider the description of M-Ary Multiplication in [30] by Masahiro et al.. Again, we try to solve the problem of computing $Y^y = g^k$ in group $G$, we take into considering the binary expansion of the private exponent $k$. In the $j$-Ary method, the exponent is divided into $j$ blocks (assuming an even $j$) and processed. Parallels can be made with the windowing method [47, 16]. If the intention is to increase the processing speed, $j$ should be increased. On the other hand, in memory-limited environments $j$ is fixed to 2.

Listing 4.3: M-Ary Multiplication in Exponentiation in TinyPK

```
begin d, B
Q[0] = B
Q[1] = 1
r[0] = B
w = 2
/* Window size */
for i = 1 up to w
r[i] = r[i-1] * B
/* Precomputation */
for i = 1 up to k-1 step 2
Q[0] = Q[0] * Q[0] /* square */
if (d[i]==0)
else
Q[0] = Q[0] * r[d[i]] /* multiply */
end if
return Q[0] /* B^d */
end
```

In figure 4.8, we see four loops of the inner-for loop of TinyPK. Certain similarities can be seen between operation a of bits 2 and 4, and operation b of bits 1 and 3. These represent different paths through the if-statement (as seen in the listing 4.3). That being said, because of a precomputation round it will still be difficult to deduce the private exponent. We process the entire private key radix 2, so if no exploit is found, then no information is revealed about the
key. On the other hand, if the specific precomputed value used during an execution loop can be deduced, then by extension the exact sequence of every pair of 2 bits of the secret key is exposed.

In figure 4.9, a divergence in the 5th bit is revealed, which is exaggerated in figure 4.10. Again, due to the way the key is processed, even this information is not enough to divulge the key. Using DPA and timing attacks it will be possible to analyze which $r[]$ was selected during the multiplication stage. This will allow us to backtrack and expose the key.

![Figure 4.9: Superposition Plot of Exponentiation with M-Ary Multiplication, 6-averaged, filtered, Green: 77776177777777, Blue: 77777177777777](image-url)
Figure 4.10: Plot of difference of two 6-averaged, filtered executions of M-Ary Multiplication showing a divergence in the 3rd repetition of the outer for-loop.
4.1.4 Private Exponent Blinding

Exponent blinding is an excellent way to mask computations and thwart DPA attacks. The implementation simply requires the construction of a private exponent $r$. A random number is generated, and multiplied by the modulus. The resultant value is added to the private exponent to get $r'$. This value is used in calculations, as per standard procedure. Because a multiple of the modulus is added to the original exponent, when calculating values modulo that modulus, the result is the same as using the original exponent. The length of the exponent is positively affected by this procedure, having a direct impact on the length of computation. Restricting the random number to 1 byte, means the exponent becomes at least the length of the modulus. Originally the application had a 14 byte exponent and a 96 byte modulus, and took 24 seconds. Having a key length of approximately 100 bytes would increase this processing time to about 180 seconds. Whether this is still feasible is another question.

On wireless devices using TinyOS, there are some issues using random numbers, due to weaknesses in random number generator [12]. The seed used to initialize the generator is directly dependent on the mote id, which is unique per mote. For most applications this suffices, but in terms of cryptanalysis, it is woefully insufficient. Therefore, from a practical perspective, this protective measure can easily be circumvented, unless a better source of randomness can be developed.
4.2 EccM

4.2.1 Double-and-Always-Add

The Double-and-Always-Add algorithm was developed by Coron in 1999 [6], and is sometimes called Coron’s Dummy Multiplication. The principle is analogous to the Square-and-Always-Multiply method for the DL problem. It has been elaborated upon in the work by Izu and Moller [14]. As in subsection 4.1.1, exponentiation takes equal amounts of doubling and adding operations, and therefore increases the processing time compared to the regular binary algorithm. As far as protection from SPA, Coron’s Multiplication is sufficient, but Brier and Joye have proposed improvements which make addition and doubling algorithms indistinguishable, further hardening the system.

Listing 4.4: Coron’s dummy addition method

```
1  begin d, P
2  Q[0] = P
3  Q[1] = 0
4
5  for i = k-2 down to 0
6    Q[0] = Q[0] + Q[0] /* double */
7    Q[1] = Q[0] + P /* add */
8    Q[d[i]] = Q[0]
9  return Q[0] /* d*P */
10 end
```

Figure 4.11: Plot of one execution of EccM using Exponentiation with dummy addition
Figure 4.12: Plot of individual double and add operations in EccM using Exponentiation with dummy addition.

Figure 4.13: Plot of difference of two 5-averaged, filtered executions of Always-Add Exponentiation showing a divergence in the 3rd bit.
4.2.2 Montgomery Ladder for ECC

The Montgomery Ladder technique was described in the paper by Joye and Yen [18], and in the seminal work by Montgomery [34]. As it could be applied to both DL and ECC cryptosystems, we made an attempt to implement it here, having seen the positive results in the DL case. The work by Zhang et al. [50] was also used in developing the code.

![Plot of execution of EccM using Montgomery Multiplication](image)

**Figure 4.14:** Plot of execution of EccM using Montgomery Multiplication

In the detailed plot 4.15, we see alternating adding and doubling operations. It is close to impossible to tell by visual inspection the value of the data being multiplied or doubled, but when the difference of two signals is taken in figure 4.16, it is clear that the second bit is different. This shows that although this method is secure against simple power analysis, it would fail in the light of more advanced methods.
Figure 4.15: Detailed plot of execution of EccM using Montgomery Multiplication

Figure 4.16: Plot of difference of two 6-averaged, filtered executions of Montgomery Multiplication showing a divergence in the 2nd bit
4.2.3 Joye-Tymen Base Point Randomization

When working with elliptic curves, the interesting concept of base point randomization arises as a countermeasure against DPA [17]. Using the Joye-Tymen method, we must construct an isomorphism between two elliptic curves. A point $P(x, y)$ is transformed into $P'(r^2 \times x, r^3 \times y)$.

The parameters $a$ and $b$ which define the initial curve, are thereby transformed to $r^4 \times a$ and $r^6 \times b$. Elliptic curves with two defining parameters are of the Weierstrass form, which conveniently is the same as the elliptic curve used in EccM. This form of randomization will allow us to fix
the Z-coordinate of our points, allowing us to effectively ignore that dimension. In that way, we can get away without modifying the scalar multiplication function. The first set of figures – 4.17, 4.18 and 4.19 – show the application of a randomized basepoint in modular double-and-add exponentiation. The second set of figures – 4.20, 4.21 and 4.22 – show how a randomized basepoint affects the execution of ECC Montgomery exponentiation.
Figure 4.21: Detail of Montgomery exponentiation with randomized basepoint

Figure 4.22: Plot of difference of two 5-Averaged, filtered executions of Montgomery Exponentiation with randomized basepoint showing some divergence
4.3 Discussion

The SPA-resistant methods implemented from the literature have proved to be successful. While in chapter 3 it was possible to view secret key information without undue effort, now these kinds of attacks have been thwarted. The Montgomery method for multiplication was the most successful the anti-SPA techniques. In both the cases of TinyPK and EccM, it was possible to protect against both SPA and templating attacks. In this case, using the MESD attacker model presented at the beginning of the thesis, attacks are still possible if the evil-doer has access to the source code, and an identical device with which to experiment.

If a final step is taken, such as Joye-Tymen Base Point Randomization for EccM (as seen in section 4.2.3), then even the MESD attacker may be rendered insufficient. We tested the base point randomization technique with both the regular binary algorithm, and the SPA-secured Montgomery multiplication. In this way we provided an implementation that is both secure against SPA and DPA attacks.

It proved to be difficult to create an infrastructure to gather enough results to do differential power analysis, but a potential work-around was suggested. By experimentation, we found that by controlling the oscilloscope through the Ethernet port, it was possible to query a result, grab the data and write it to file in 4.5 minutes. If errors are ignored, then it would be possible to increment, decrement or randomly generate the message to be encoded every 4 minutes by use of the TinyOS Timer module. The controller-PC can then activate the trigger mode on the oscilloscope on a similar cycle. The trigger on the DSA90804A Infinium oscilloscope can be programmatically configured to wait for 4 successive peaks of a height of 700 mV (four peaks indicate that the device has been started and the three LEDs have been activated). In this way, several hundred runs can be gathered in a 12 hour period and would provide a foundation to do DPA. It would be interesting to use this method to analyze the implementations presented in this paper.

The results gathered conclusively show that it is possible to simply protect a cryptographic implementation against power analysis attacks, using techniques described in the literature. As shown in table 4.1, the speed reduction is reasonable compared to the security afforded. Outliers such as the greater than expected run-time increase from regular binary exponentiation to Double-And-Always-Add in EccM is due to a private exponent with a low Hamming weight (100001000010000100001). This can also explain the slow-down for Montgomery Multiplication.

---

3 This was assuming each measurement was flawless and the mote did not spike or restart due to a loose electrical connection.
4 An increase of 30% is expected, due to every bit of the exponent requiring an Add, and a Double operation
4.3 Discussion

Table 4.1: List of speed of various implementations of TinyECC and TinyPK, using fixed private exponents

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Execution(sec)</th>
<th>$\sigma$ for 10 reps</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>EccM</td>
<td>28.71</td>
<td>0.12</td>
<td>—</td>
</tr>
<tr>
<td>EccM with Double-And-Always-Add</td>
<td>56.17</td>
<td>0.09</td>
<td>95%</td>
</tr>
<tr>
<td>EccM with Montgomery Multiplication</td>
<td>56.19</td>
<td>0.11</td>
<td>95%</td>
</tr>
<tr>
<td>EccM with Rand. Basepoint</td>
<td>28.81</td>
<td>0.12</td>
<td>0.3%</td>
</tr>
<tr>
<td>EccM with Montgomery Multiplication &amp; Rand. Basepoint</td>
<td>57.01</td>
<td>0.06</td>
<td>98%</td>
</tr>
<tr>
<td>TinyPK</td>
<td>20.60</td>
<td>0.14</td>
<td>—</td>
</tr>
<tr>
<td>TinyPK with Square-And-Always-Multiply</td>
<td>33.75</td>
<td>0.05</td>
<td>63%</td>
</tr>
<tr>
<td>TinyPK with Montgomery Multiplication</td>
<td>37.44</td>
<td>0.05</td>
<td>82%</td>
</tr>
<tr>
<td>TinyPK with M-Ary Multiplication</td>
<td>21.28</td>
<td>0.11</td>
<td>3%</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion

It is possible, with relatively minor modifications, to protect an insecure cryptographic library from power analysis attacks. Exploits were found in TinyPK, EccM and TinySec, and a variety of solutions for TinyPK and EccM were implemented and evaluated.

The analysis of TinyPK showed that it is vulnerable to simple power analysis. The raw signal taken from the oscilloscope directly revealed hidden information. Using filtering techniques, it was even easier to identify the operations that were occurring. In the second phase of the project, several solutions to solve TinyPKs data leakage were proposed. Two attacks, namely the Square-and-Always-Multiply and Montgomery Ladder method, had the effect making the number of squaring and multiplication operators independent of the encryption key, thereby helping to blind it. M-Ary Multiplication also protected against simple power analysis, by pre-computing a sequence of squaring operations, and using the result to process larger chunks of the exponent at once.

Also EccM proved to be vulnerable to power analysis. Since its default implementation was the ECC analog of TinyPK, this was to be expected. Several methods to plug the figurative side channel leakage were suggested and enacted. The ECC Double-and-Always-Add method, and the Montgomery Ladder were shown to be as successful as the respective TinyPK versions. For EccM though, we went one step further by testing an algorithm that aims to protect against DPA attacks. When combined with the Montgomery Ladder, the result is an airtight and practical cryptographic library that is only slightly slower than the unsecured exemplar. The results of our work are summarized in table 5.1.

In the context of wireless sensor devices, every effort makes a difference because of the physical limitations of the platform. As with many security measures, doubling or tripling the amount of time it takes to break a device will discourage more people from making the attempt. The work we have done highlights the importance of protecting against side-channel attacks, and shows that many theoretical solutions from the literature can be effectively implemented in practice.
Table 5.1: Comparison of protective measures of various cryptographic libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Primitive</th>
<th>Protection</th>
<th>¹SPA</th>
<th>²DPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyPK</td>
<td>Discrete Logarithm</td>
<td>None (Binary Algorithm)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TinyPK</td>
<td>Discrete Logarithm</td>
<td>SAM²</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TinyPK</td>
<td>Discrete Logarithm</td>
<td>Montgomery</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TinyPK</td>
<td>Discrete Logarithm</td>
<td>M-Ary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TinyEcc (ECDSA)</td>
<td>Elliptic Curve</td>
<td>Sliding Window</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EccM</td>
<td>Elliptic Curve</td>
<td>None (Binary Algorithm)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>EccM</td>
<td>Elliptic Curve</td>
<td>DAA³</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EccM</td>
<td>Elliptic Curve</td>
<td>Montgomery</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EccM</td>
<td>Elliptic Curve</td>
<td>Basepoint Rand.</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>EccM</td>
<td>Elliptic Curve</td>
<td>Basepoint Rand. + Montgomery</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

¹ Secure against
² Square-and-Always-Multiply
³ Double-and-Always-Add
Bibliography


Appendix A

Physical Layer Attacks on Sensor Nodes

A.1 Introduction

Wireless sensor networks have become a popular academic research field and hardware platforms such as the TMote Sky [39] are used for a wide range of different experimental implementations, including protocols for secure ad-hoc networks. Due to the limited resources available, special protocols for secure communications such as TinySec [20], have been proposed for these platforms. TinySec attempts to provide link layer security with minimal additional power consumption, using symmetric cryptographic methods. While such protocol suites might be theoretically secure with certain attacker models, their implementation on real hardware devices enables the attacker to mount attacks based on physical phenomena, so called side channel attacks.

A.2 Objectives / Goals

Common use cases for wireless sensor nodes often assume their unsupervised operation in potentially hostile environments. In such environments, it is likely that the attacker can get physical access to the nodes themselves, and will try to obtain the keys stored on the nodes to decrypt or manipulate the network traffic. Previous research in ways to obtain this key were often limited to manipulations of the node memory over JTAG [2]. Other related attacks which were proposed are directed towards timing attacks on insecure password algorithms [8]. On the other hand, power analysis attacks should be simple to perform. Although power traces were used to evaluate the power consumption for some cryptographic algorithms such as [45], so far they have not been used in this context to extract the key. The goal of this Master’s thesis is to evaluate
different academic implementations of cryptographic algorithms for sensor nodes and test their resistance to simple power analysis (SPA) and differential power analysis (DPA).

The project consists of the following parts:

- Familiarization with the problem of Side Channel Analysis on cryptographic algorithms, and the different crypto-suites for sensor modes.
- Experimental analysis of these current implementations
- In case weaknesses were found, proposal of possible mitigations

Deliverables include the documented resulting code, a thesis report, and a final presentation.

### A.3 Prerequisites

The power analysis is done using oscilloscopes, but only minimal previous experience is required. The signals are then analysed in Matlab, which the student is expected to be familiar with. The student is also expected to have a basic understanding of pattern recognition. Summarizing, the following is required:

- Background in signal processing
- Previous experience in Matlab
- Background in Cryptography
- Interest in the related security and communications area

In addition, knowledge in the following areas might be helpful:

- Signal acquisition
Appendix B

Installing TinyOS 1.14

Are you using TinyOS 1.x on modern hardware? Have you had problems adapting to guides written in the early 00’s? I know I have, and have since compiled an updated tutorial about how to get everything running on later versions of Ubuntu. I hope this will spare you some pain.


Let us have a look at prerequisites:

- A freshly installed version of Ubuntu 8.04.3 LTS, fully updated (I have tried Redhat-based systems without success)
- A little bit of luck

To prepare the machine, please download and install the following:

- The build-essential, cpp-3.4, libc6-dev, gcc-3.4, cvs, subversion, autoconf and automake1.9 metapackages from hardy-updates

- The version of mspgcc I have provided below (in the tinysos distribution) only compiles with gcc-3.2 and gcc-3.4, but definitely not gcc-3.3 or gcc-4.0+. Using the wrong compiler causes countless errors.

- AVR Tools and Compiler [avr-binutils_2.13.2.1-2_i386.deb, avr-gcc_3.3tinyos-2_i386.deb, avr-libc_20030512cvs-2_i386.deb]

[^1]: http://www.matthewjmiller.net/howtos/installing-tinyos-for-telos-on-linux/
[^2]: http://www.5secondfuse.com/tinys/install.html
[^3]: http://www.comnets.uni-bremen.de/~mab/tinyosdebian.html
[^4]: http://www.mobilab.unina.it/TinyOSDebianH.htm
[^6]: http://none.cs.umass.edu/~dganesan/courses/fall06/slides/TinyOS_Setup.html
Please ignore all errors regarding later versions in the repository.

- Nesc 1.1.2 [nesc_1.1.2b-2_i386.deb]
- libglib 1.2 [libglib1.2.1.2.10-10_i386.deb]

Here as well, you need version 1.2, not the later version in the repository.

- libgtk 1.2 [libgtkl1.2.1.2.10-18_i386.deb]
- TinyOS Tools 1.1.0 [tinyos-tools_1.1.0-2_i386.deb]
- IBM Java SDK 1.4.2 [ibmjna2-142-ia32-sdk_1.4.2-6_i386.deb]
- IBM Javacomm 1.4.2 [ibmjna2-javacomm_1.4.2-6_i386.deb]

After installing these packages, it would be best to lock them, so they will not be washed away the next time you update the system. To do this, start aptitude, and mark the above packages using '='. This will protect your delicate setup from destruction.

Now check out TinyOS 1.14 from Sourceforge into your /opt folder (where IBM Java should have been installed) and change ownership of all files to the current user. For the sake of your later compiling efforts, set the permissions of the tinyos-1.x directory to 777:

```
Listing B.1: Download TinyOS from the CVS repository and change some premissions
1 cd /opt
2 cvs -d:pserver:anonymous@tinyos.cvs.sourceforge.net/cvsroot/tinyos login
3 cvs -z3 -d:pserver:anonymous@tinyos.cvs.sourceforge.net/cvsroot/tinyos co tinyos-1.x
4 sudo chown -R 'whoami' tinyos-1.x
5 sudo chown -R 'whoami' IBMJava2-142
6 sudo chmod -R 777 /opt/tinyos-1.x
```

To simplify the use of the TinyOS build chain, I have written two functions which can easily be added to your .bashrc. The following function configures PATH, CLASSPATH and other variables:

```
Listing B.2: Function for initializing the TinyOS development environment
1 export IS_TOS_ON="0"
2 function tosetup()
3 {
4   if [ $IS_TOS_ON = "0" ]; then
5     export O_TOSROOT=$TOSROOT
6     export O_TOSDIR=$TOSDIR
7     export O_MAKERULES=$MAKERULES
8     export O_JDKROOT=$JDKROOT
9     export O_JAVAXROOT=$JAVAXROOT
10    export O_CLASSPATH=$CLASSPATH
11    export O_PATH=$PATH
12    export O_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
13    export TOSROOT="/opt/tinyos-1.x"
14    export TOSDIR="$TOSROOT/tos"
15    export MAKERULES="$TOSROOT/tools/make/Makerules"
16    export TINYNODE_DIR=$TOSROOT/contrib/shockfish
17    export TOSMAKE_PATH=$TINYNODE_DIR/tools/make
18    export JAVAX_ROOT="/opt/IBMJava2-142"
19    export JAVA_CLASSPATH="/opt/IBMJava2-142/"
21    export PATH="/opt/msp430/bin:$JAVAX_ROOT/bin:$PATH"
22    export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/lib:$JAVAX_ROOT/jre/bin/"
```
export IS_TOS_ON="1"

if [ $IS_TOS_ON = "1" ]; then
    export TOSROOT=$O_TOSROOT
    export TOSDIR=$O_TOSDIR
    export MAKERULES=$O_MAKERULES
    export JDKROOT=$O_JDKROOT
    export JAVAXROOT=$O_JAVAXROOT
    export CLASSPATH=$O_CLASSPATH
    export PATH=$O_PATH
    export LD_LIBRARY_PATH=$O_LD_LIBRARY_PATH
    export IS_TOS_ON="0"
else
    echo "!!!TOS already off!!!"
fi

And this one puts everything back the way it was:

Listing B.3: Function for restoring the development machine to its previous state

```bash
function tosreset {}
{
    if [ $IS_TOS_ON = "1" ]; then
        export TOSROOT=$O_TOSROOT
        export TOSDIR=$O_TOSDIR
        export MAKERULES=$O_MAKERULES
        export JDKROOT=$O_JDKROOT
        export JAVAXROOT=$O_JAVAXROOT
        export CLASSPATH=$O_CLASSPATH
        export PATH=$O_PATH
        export LD_LIBRARY_PATH=$O_LD_LIBRARY_PATH
        export IS_TOS_ON="0"
    else
        echo "!!!TOS already off!!!"
    fi
}
```

Reload your .bashrc using the source command, and test it out.

Now mspgcc needs to be compiled, to allow you to compile code for telos devices:

- Initialize the environment by calling tossetup from the command line
- Go to $TOSROOT/tools/src/mspgcc
- Change the following lines in the build-mspgcc script. The default links no longer exist so we must change them:

**Listing B.4: Patch for build-mspgcc script to update repository location**

```bash
--- GCC32_URL="ftp://ftp.gnu.org/gnu/gcc/gcc-3.2.3/gcc-core-3.2.3.tar.bz2"
--- GCC33_URL="ftp://ftp.gnu.org/gnu/gcc/gcc-3.3.5/gcc-core-3.3.5.tar.bz2"
+++ GCC32_URL="ftp://sources.redhat.com/pub/gcc/releases/gcc-3.2.3/gcc-core-3.2.3.tar.bz2"
+++ GCC33_URL="ftp://sources.redhat.com/pub/gcc/releases/gcc-3.3.5/gcc-core-3.3.5.tar.bz2"
```

- Leave the editor and run the build-mspgcc script, and verify that the gcc-3.4 compiler is being used. By default, the executables will be installed in /opt/mspgcc:

```
export CC=gcc-3.4; export USE_GCC=3.4; sudo ./build-mspgcc install
```

- Symlink some important files for ease of access:

```
ln -s $TOSROOT/tools/src/mspgcc-pybsl/bsl.py /opt/msp430/bin/msp430-bsl
ln -s $TOSROOT/tools/src/motelist/motelist-linux /usr/bin/motelist
```

Now we will compile the additional java tools which come with the TinyOS distribution:


cd $TOSROOT/tools/java
make; make;

The TinyOS team has a small test script that ensures everything you need has been installed. You should only have a warning about the graphic package when you run toscheck.

cd $TOSROOT/tools/scripts
./toscheck

To enable the USB interface for transferring data to devices, please open $JDKROOT/jre/lib/javax.comm.properties, and uncomment the line:

# /dev/ttyUSB=PORT_SERIAL

Lastly, change the permissions on the ports you would like to use to 666. I do not believe this sticks when you restart, so you may have to reapply the settings upon restarting.
Appendix C

Preparing Measurements

We look at the various cryptographic implementations that exist for wireless sensor devices, and how to prepare the code for side-channel analysis. I cover Tinysec, MiniSec, TinyECC, TinyPK and EccM.

C.1 TinySec

TinySec is a crypto-system built into the network stack of the device. This means it has been implemented for the most popular radio chips, the CC1000 (found in Mica* motes) and the CC2420 (found in TelosB motes). There is no way to avoid initializing the radio, because without initialization, the encryption and signature procedures are not executed.

The way around this is to leave most of the code untouched, and simply dial down the power of the radio transmission.

In $TOSROOT/tos/lib/CC2420Radio/CC2420Const.h:

```c
--- #define CC2420_DEF_RFPOWER 0x09
+++ #define CC2420_DEF_RFPOWER 0x00
```

In $TOSROOT/tos/platform/mica2/CC1000Const.h:

```c
--- #define CC1K_PA_POW 0x09
+++ #define CC1K_PA_POW 0x00
```

This is useful in general, because by applying these fixes, no call to the radio will disturb your measurements. The second option (only for TelosB devices) is to add a flag to the Makefile of the program whose radio you want to deactivate.

In $TOSROOT/apps/TestTinySec/makefile:
As far as the sample code is concerned, we need to eliminate some of the randomness to get a clear, reproducible signal when doing SPA.

In $TOSROOT/apps/TestTinySec/TestTinySecM.nc$, in `result_t IntOutput.output()`:

```c
--- value = (uint8_t) v;
+++ // value = (uint8_t) v;
```

After the IV has been initialized, do not update it anymore. This is helpful for simple power analysis. It would be possible to extrapolate an attack with an incremental IV, because the random number generator on these devices is seeded by `TOS_LOCAL_ADDRESS`, which is fixed for a particular device.

In $TOSROOT/tos/lib/TinySec/TinySecAppM.nc$, in `void setIV(n)`:

```c
--- if (!ivlen) return;
+++ if (!ivlen) return; return;
```

In $TOSROOT/tos/lib/TinySec/TinySecM.nc$, in `result_t TinySecControl.updateEncryptionKey()`:

```c
+++ return SUCCESS; (after memcpy())
```

In $TOSROOT/tos/lib/TinySec/TinySecM.nc$, in `result_t TinySecControl.updateMACKey()`:

```c
+++ return SUCCESS; (after memcpy())
```

Adding four random bytes to the message is meant to ensure a wider distribution of the resulting ciphertext. Instead of using a random number, here we pad with 0s.

In $TOSROOT/tos/lib/TinySec/TinySecM.nc$, in `result_t addPadding()`:

```c
--- uint16_t r = call Random.rand();
+++ uint16_t r = 0;
```

### C.2 MiniSec

MiniSec is another secure network layer which was designed as a replacement for TinySec in 2007. TinySec officially supports TelosB and Mica nodes, but it only works on the latter. MiniSec, on the other hand, was implemented with the lower-memory Telos platform in mind and focuses on low energy consumption. When compromises like this have to be made, there is a good chance that several avenues of attack are left open.

As with TinySec, we must fix the IV since it is used as part of the encryption key.

In $TOSROOT/tos/system-minisec/AMstandard.nc$, in `void decIV()`:
C.3 TinyECC v0.3

TinyECC is a ECC-based encryption library from the TelosB mote. In order to remove randomness, the following patches must be applied.

In $TOSROOT/tos/system-minisec/AMstandard.nc, in void incIV():

```c
--- IV[b]--;
+++ IV[b] = IV[b];
```

In $TOSROOT/tos/system-minisec/AMstandard.nc, in void incIV():

```c
--- IVs[b]--;
+++ IVs[b] = IVs[b];
```

C.3 TinyECC v0.3

TinyECC is a ECC-based encryption library from the TelosB mote. In order to remove randomness, the following patches must be applied.

In $TOSROOT/apps/TinyEcc0.3/makefile_Alice:

```c
+++ CFLAGS+=-DTEST_VECTOR
+++ CFLAGS+=-DSIXTEEN_BIT_PROCESSOR
```

In this case, the private key is fixed, but the message is generated randomly.

In $TOSROOT/apps/TinyEcc0.3/AliceM.nc, in void sign():

```c
--- message[j] = (uint8_t) call Random.rand();
+++ message[j] = (uint8_t) 4;
```

C.4 TinyPK

TinyPK is based on the use of TinySec and enables mutual authentication without a preshared key. Resource-heavy private RSA operations are outsourced to a powerful computing device, while public operations are done on the mote. This software used the square-and-multiply algorithm, which shortened the cracking time. All that needed to be done here, was to fix the private key (which is generated by calling `Rand.Random()` on the device).

The code, as is, is non-functional. Some changes need to be made to get everything to compile and run.

In $TOSROOT/apps/DH/DHm.nc, in void clear():

```c
--- while (lth--) *from++ = 0;
+++ memset(from , (uint8_t) 0 , lth);
```

In $TOSROOT/apps/DH/DHm.nc, in result_t StdControl.init():

```c
```
--- while (i < datasize)  
---   {  
---     j = call Random.rand();  
---     testrand[i] = (uint8_t)j;  
---     i++;  
---   }  
+++ while (i < datasize)  
+++   {  
+++     if (i==5)  
+++       {  
+++         j = 0x01;  
+++       }  
+++     else  
+++       {  
+++         j = 0x07;  
+++       }  
+++     testrand[i] = (uint8_t)j;  
+++     i++;  
+++   }

In $TOSROOT/apps/DH/DHm.nc, in int8_t Exponentiate():

--- memset(work,(uint8_t) 0,sizeof(work));  
+++ clear(work, sizeof(work));

A minimal version of TinyPK has been provided in the accompanying tarball.

### C.5 EccM

Written in 2004, EccM was the first known implementation of a PKI using elliptic curve cryptography. An analogue to TinyPK it uses the double-and-add method for exponentiation. The default implementation is intended to be used in combination with TinySec, but for the purposes of experimentation, this has been disabled.

Alice and Bob’s private keys are chosen randomly. Assuming a multiple-key single-message attack, we can change the code to fix the key.

In $TOSROOT/apps/EccM-2.0/EccM.nc, in void task generate_privKeyA():

--- for (i = NUMWORDS/2; i < NUMWORDS; i++)  
---   {  
---   }
--- privKeyA.s[i] = (word_t) call Random.rand();
--- }
+++
+++
+++
+++
+++ for (i = NUMWORDS/2; i < NUMWORDS; i++)
+++ {
+++ if (i%4==1)
+++ {
+++ privKeyA.s[i] = (word_t) 0x01;
+++ }
+++ else
+++ {
+++ privKeyA.s[i] = (word_t) 0x00;
+++ }
+++ }

A second option is to use either Alice or Bob’s key as specified in
$TOSROOT/apps/EccM-2.0/EccM.h. A minimal version of EccM has been provided in
the accompanying tarball.