Security versus Power Consumption

Stian Jahr

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2006

Abstract

In many cases there are trade-offs between security and user-friendliness, or security versus speed. Our society is moving towards a more wireless community. This thesis will cover an aspect of security versus user-friendliness in the form of how do the security implementations in battery-powered devices affect the battery life. Do we need to charge the batteries more often when we implement security? Other authors has found the power consumptions of several security implementations. Previous work presents their results in joule, however, the user-friendliness cannot be measured in joule. A better unit for measuring the decreased charge interval. This thesis presents a theoretical model to find the decrement of the charge interval when security is implemented.
The model is used in a case study to find how much the charge interval is decreased when we implement a PGP inspired encryption of SMSes. The results shows that the charge interval is reduced by 42% (from 202 hour to 117 hours) with an RSA key of 2048 bits and we send 20 encrypted SMSes each day. The main reason for the decreased charge interval is the use of the resource demanding RSA algorithm, however it is shown that the sending of extra bits due to the expansion of the message when PGP is applied played an important role in the decrement.

Sammendrag

I mange sammenhenger må man ta avveininger mellom sikkerhet og brukervennlighet, eller sikkerhet mot hastighet. Vårt samfunn er på vei til å bli mer og mer trådløst. Denne masteroppgaven dekker et aspekt ved sikkerhet aveiet mot brukervennlighet i form av hvordan sikkerhets implementeringer i batteridrevende enheter påvirker levetiden av batteriet. Tidligere arbeid har funnet energiforbruket av diverse sikkerhets implementeringer. Resultatene av disse er ofte angitt i joule, men brukervennlighet kan ikke måles i joule. Et bedre mål for brukervennlighet er antall timer ladningsintervallet synker når vi implementerer sikkerhet. Denne masteroppgaven beskriver en teoretisk modell for å finne ut hvor mye levetiden på batteriet senkes når man implementerer sikkerhet.
Modellen er brukt i et case study for å finne hvor mye ladningsintervallet synker når vi implementerer en PGP inspirert kryptering av SMSer. Resultatet viste at ladnings intervallet sank med 42% (fra 202 til 117 timer) med en RSA nøkkel på 2048 bits og vi sender 20 krypterte meldinger hver dag. Hovedgrunnen til dette er den resurskrevende RSA krypteringen, men det viste seg at å sende ekstra bits på grunn av meldings ekspansjon av en PGP kryptert melding spilte en viktig rolle i nedgangen av batterilevetid.

Acknowledgments

First of all I will like to thank my supervisor, professor Einar Snekkenes, for valuable guidance through this master thesis. Einar is a man of many thoughts and ideas that was helpful during the dissertation. I will also like to thank my thesis opponent, Vidar Evensrud Seberg, for fabulous feedbacks to my thesis.

My next thanks are going to Hossein Hayati Karun for introducing me to regular expressions. Without his help, the time of implementation of my application (EnergyCalc, presented later in this thesis) was reduced.

I will also like to thank my student friends, written their master thesis on the project room A-220, for motivation and discussions during the master thesis.

-Stian Jahr

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic covered by this thesis

This thesis presents an aspect of the trade-off between user-friendliness and security. The thesis' angle of user-friendliness is how often do we need to re charge the batteries of a battery powered device. The main question is «Do we need to charge the batteries more often when we implement security?»

Keywords: Power consumption, usability, availability, information security.

1.2 Problem description

There are often requirements to both security and the battery's charge interval. When security is implemented, it is at the sacrifice of battery lifetime. Also when there is a need to increase the battery lifetime it may be at the sacrifice of the security. The trade-off between these requirements may be hard to consider. When designing hardware in wireless environments, the power consumption of the functions is an important issue. E.g. when making a device to log acceleration to see if a package has been handled with care during transportation, and want the log to be stored encrypted in the memory. To select the encryption algorithm and key size there is a need to know how much power the different encryption algorithms consume of power. In other cases, encryption or not should be considered. E.g. when using an unsecured wireless network with a battery powered laptop and need to upload a big file to an FTP server. To consider using encryption or not, one need to know how much the use of for example an SSH tunnel increases the power consumption and reduces the charge interval.

Battery lifetime is a part of the user-friendliness. E.g. when implementing secure messaging on a mobile phone we may need to know how much the security decreases the user-friendliness in the shape of how long the battery last. Can we continue the same way as we did before the security is implemented, or do we need to charge the battery more often than before?

1.3 Justification, motivation and benefits

In our wireless society it is necessary to design the hardware devices with low power consumption. When knowing how much implementation of security decrease the charge interval, it may be easier to decide the trade-off between security implementations and charge interval.

The stakeholders of this thesis are those making battery-powered devices and needs to implement security. This thesis will help us to find out how much more power consuming the hardware will be when they implement security in the hardware device.

1.4 Research questions

The research questions answered in this thesis are:

1. How to find power consumption of functionality with and without security?
    - Theoretically

- Experimental

2. What is the power consumption of different security implementations?
3. How do security implementations affect the battery's charge interval?

2 Related work

This chapter searches the literature for previous work related to the thesis. The chapter is divided in sections based on the research questions stated in the previous chapter.

2.1 How to find power consumption of security implementations?

2.1.1 Theoretically

In [1], Osmulski et al describes a probabilistic power prediction tool for the Xilinx 4000-series FPGA[1]. The tool is a java based application, which based on two input files can predict the power consumption with typically less than about 5% error. The input files are:

1. A configuration file associated with an FPGA design
2. A pin file that characterizes the signal activities of the input data pins to the FPGA

> Based on the two input files, the tool propagates the probabilistic information associated with the pins through a model of the FPGA configuration and calculates the activity of every internal signal associated with the configuration. The activity of an internal signal s, denoted a_s, is a value between zero and one and represents the signal's relative frequency with respect to the frequency of the system clock, f.Thus, the average frequency of signal s is given by $a_s f$. [...]
>
> The average power dissipation due to a signal s is modeled by
>
> $$\frac{1}{2}C_{d(s)}V^2 a_s f$$
>
> ,where $d(s)$ is the Manhattan distance the signal s spans across the array of CLBs, $C_{d(s)}$ is the equivalent capacitance seen by the signal s, and V is the voltage level of the FPGA device. The overall power consumption of the configured device is the sum of the power dissipated by all signals. [...] Letting S denote the set of all internal signals for a given configuration, the overall power consumption of the FPGA is given by:
>
> $$\begin{aligned} P_{avg} &= \sum_{s \epsilon S} \frac{1}{2} C_{d(s)} V^2 a_s f \\ &= \frac{1}{2} V^2 f \sum_{s \epsilon S} C_{d(s)} a_s \end{aligned} \tag{2.1}$$

[1].

This approach can only test one specific setting with a specific set of input signals. In the literature this is referred to as *input pattern-dependent*. The advantage of this method

[1]Field Programmable Gate Array (FPGA)is an electronic device that can be programmed with software in a language called VHSIC(Very-High-Speed Integrated Circuit) Hardware Description Language(VHDL)

is the platform independence. It can be used to estimate the power of any circuit, regardless of technology, design style, functionality, architecture, etc[2].

In Najm's survey[2], he found that most power estimation techniques have simplified the problem in three ways:

- The chip's power supply delivers a stable and non-changing voltage. The power can now be calculated with the draw current. ($P = U \cdot I$).
- The circuit is build of logic gates and latches and has a design style of synchronous sequential circuit.
- The latches are edge-triggered and the circuit draws no steady-state supply current.

With these assumptions the power dissipation is broken down into

1. the power consumed by the latches
2. the power consumed by the combinational logic blocks

The estimation methods Najm presents in [2], are approaches to overcome the *strong pattern-dependence problem*. However the techniques presented are *weakly pattern-dependent* since the user has to supply typical behavior of the system's input in form of probabilities (average fraction of time that the signal is high) and density (average number of fractions per second).

In [3], Jagau presents Simcurrent, a tool to predict the power consumption of complex CMOS circuits. The accuracy of Simcurrent is about 5%. The tool is based on the fact that the power consumption of a CMOS circuit is present when the logical values are switched. It combines the necessary logic simulation with results from an analog simulation. Previous work has shown that the current flow is dependent on the following parameters[4]:

- Gate size (i.e. resistance of the pull-up and pull-down path)
- The load capacitance (line and gate input capacitance)
- The slope of the input signal

The tool contains an algorithm taking care of these dependencies. Simcurrent uses a set of permanent data and a set of generated data produced during the circuit analysis. The permanent dataset is permanent for a given technology or process family and contains the following data:

- A list of gate names with their respective input capacitances, driver capability (fan-out), and gate type (inverting or non-inverting)
- Calculated switching capacitances of each gate
- A set of analog signal shapes for two reference gates under representative load conditions (0% ... 100% load)

From the given dataset Simcurrent finds the current consumption by the following algorithm:

1. Search the netlist for all gate inputs with the same signal names

2. Search the netlist for corresponding cell names
3. Determine the real load (C_{load}) of each output
4. Analyze of the event table (signal, names and connected gates)
5. Calulate the actual current shapes from the database depending on the gate type and transition
6. Calculate the resulting current

The tool is validated by comparing the result of the tool to an Extended SPICE[2] simulation.

2.1.2 Experimental

In [5], Bhargava et al present a method to measure the energy consumption in a PDA based mobile devices as a part of their work. They disconnected the battery from the PDA and replaced it with a DC power supply. The energy consumption was determined by measuring the input voltage and current across a test resistance of 1Ω using an oscilloscope. The reported result was a average of five independent tests.

In [6] and [7] Tiwari et al found the power consumption of several instructions in the 486DX2 CPU with a mobile PC evaluation board. The board was designed for current measurements and thus the power supply connection to the CPU was isolated from the rest of the system. The current can then be measured by an ammeter[3]. They also used a similar technique to measure the power consumption of a Fujitsu DSP, but the DPB board needed some modification since it was not designed with current measurement in mind. Tiwari et al found the power consumption of a Fujitsu SPARC lite 934 the processor chip and an IC tester machine. In their work they focused on techniques to optimize the power consumption. In [6] they proposed several techniques to optimize the power consumption: reducing memory access, energy cost driven code generation, instruction reordering for low power and processor specific optimizations.

2.2 What is the power consumption of different security implementations?

In [8], Prasithsangaree and Krishnamurthy compare the power consumption of AES and RC4 encryption algorithms. These algorithms are used in WLANs[8]. RC4 is used in Wired Equivalent Privacy(WEP)[9] and Wi-Fi Protected Access(WPA)[10], and AES is proposed used in WPA2[11]. They used a laptop with Pentium III processor with OpenSSL version 0.9.7a[12]. They tested the following metrics denoted in the respective results:

Encryption speed. The experiment showed that with small data packets (less than approximately 90 bytes) the AES algorithm was the most effective in MB/sec. However, the effectiveness of the RC4 algorithms increased with the increase of the packet size, while AES hardly changed with increasing packet size.

CPU work load. This metric shows the time of the encryption. The result was similar to the previous result. With small packets the AES was the most effective algorithm

[2]Simulation Program with Integrated Circuits Emphasis
[3]Device to measure current

and packets larger than approximately 70 bytes were encrypted faster with RC4.

Energy cost. Same result as above. The current drawn while encrypting was assumed as stable. Since the energy consumed is proportional with the current drawn, this will lead to similar result.

Key size variation. The energy consumption increased slightly with increased key size for the AES algorithms. The RC4 algorithm was not affected by the increase of key size.

In Prasithsangaree's and Krishnamurthy's conclusions they suggested a combination of RC4 and AES to save energy.

In [13], Prasithsangaree and Krishnamurthy study the energy consumption of the different protocols used in WLAN (WPA, WEP and WPA2) with different keysizes. They used the energy consumption and compared this with the encryption strength. To find the energy consumption they applied a simulated environment with recorded traffic from both a campus and a home WLAN. The results of their simulation showed that the WEP-104 and WPA-128 did not consume noticeably more energy than WEP-40. The reason for this phenomenon is that they all use RC4 encryption, and as described in [8], the RC4 power consumption does not depend on key size. However, the WPAv2-128 used approximately three times more energy in a home network and almost doubled energy consumption in a campus network. This is because of the CBC-MAC.

In [14], Karri and Mishra studied the energy consumption of secure wireless sessions. Their goal was to find the power consumption as it is today and then try to optimize the power consumption without decreasing the security. The device they found the power consumption of was a Symbol PPT2800 ^{TM} Pocket PC running Windows CE ^{TM} 3.0 operating system. They equipped the pocket PC with an 11 Mbps Spectrum24 ^{TM} WLAN adapter card. To find the power consumption they applied a mobile test bed where they connected a Tektronix TDS 3054 Oscilloscope with a Tektronix TCP 202 current probe between the battery package and the pocket PC. The security protocol they used as an example in their paper was the Wireless Transport Layer Security (WTLS) [15]. In their work they divided the power consumption in two parts:

- Cryptographic computations
- Message exchange during secure session establishment and data transfers during secure data transactions.

In the experiment they measured the energy consumption of the secure session establishment to be approximately 1062mJ where 7% was cryptographic calculations and 93% was message exchanges. The secure transmission energy depends on the size of the data transmitted. When large blocks of data is transmitted the keys need to be refreshed during the transmission. The energy consumption of transmitting 2560KB and 8KB is found in Table 1

The approaches to reduce the power consumption consisted of three different techniques:

- Compression

	2560 KB data	8 KB data
SHA-256 MAC	1130	3.53
AES-128-encryption	1372	4.29
Transmission	13480	42.13
Key-refresh	245	-
Idle system	16604	51.87
Total	32831	101.82

Table 1: Energy consumed by secure wireless data transmission

- Optimizing the handshake protocol
- Use hardware to perform the encryption (FPGA)

With these techniques Karri and Mishra reduced the session establishment energy more than 6.5 times, the transaction energy more than 1.5 times and the reception energy more than 2.5 times.

In [16] Karri and Mishra further studied the energy consumed by secure wireless sessions and focused on the secure session *negotiation protocols* in IPSec[17] and WTSL[15]. To find the power consumption of IPSec they used a testbed consisting of an IBM 600E series ThinkPad^{TM} equipped with 11Mbps Spectrum24^{TM} WLAN card from Symbol Tech Inc. The processor was a 366MHz Mobile Pentium II^{TM} with 64MB SDRAM, 64KB L1 internal cache and 256KB L2 external cache. The operating system was Windows 98. The current was measured using a Tektronix TCP202 current probe and a Tektronix TDS 3054 oscilloscope.

In their IPSec experiment they tested the IPSec session negotiation in various modes. The results are summarized in Table 2

Mode	Authentication method	Client(mJ)	Server (mJ)
Main	Pre-shared key	2936	2927
	Public key signature	3717	3708
	Public key encryption	3160	3151
	Revisited public key encryption	3687	3339
Aggressive	Pre-shared key	2935	2927
	Public key signature	3666	3657
	Public key encryption	3160	3151
	Revisited public key encryption	3687	3339

Table 2: Total energy consumed by IPSec session negotiation protocol in various modes

The WTSL protocol consumed 2100mJ.

As a part of their report they proposed methods to reduce the power consumption. The energy savings are summarized in Table 3.

In [18], Nachiketh et al studied the power consumption of different cryptographic algorithms on a Compaq iPAQ H3670 PDA. To measure the power consumption they used a configuration as in Figure 1. To get an accurate measurement of the power consumption they applied synchronization between the iPAQ and the computer with LabVIEW using the serial port.

Method	IPSec	WTSL
Compression	Primary SA: 1.58x IPSec SA: 1.86x	2x
Choice of cryptographic algorithms	Primary SA: 1.44x IPSec SA: 2.1x	1.58x
Optimized client authentication	1.17x	1.34
Security association refresh and energy-efficient secret key generation	Variant 1: 1.65x Variant 2: 22x Variant 3: 16x	1.3x 12.5x 9.5x

Table 3: Energy reduction factor of secure session negotiation with several techniques

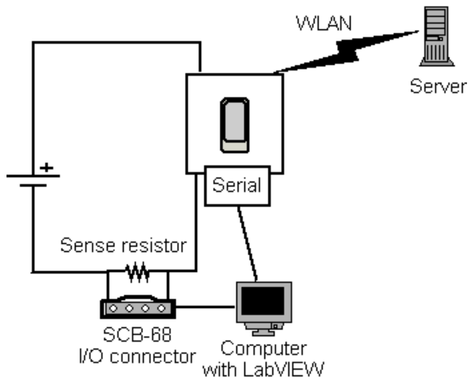


Figure 1: Energy measurement test bed with Compaq iPAQ H3670 PDA

In their experiment they tested symmetric encryption algorithms (DES, 3DES, IDEA, CAST, AES, RC2, RC4, RC5 and BLOWFISH). In each algorithm they measured the power consumed by key setup and encryption/decryption. Further, they tested hash algorithms (MD2, MD4, MD5, SHA, SHA-1 and HMAC), asymmetric encryption algorithms (RSA-1024, DSA-1024 and ECDSA-163) in key generation- signing- and verifying modes. They also tested different key exchange algorithms (DH-1024, ECDH-163 and DH-512). Different operating modes of the symmetric block ciphers was also tested (DES in CBC-, PCBC-, CFB64-, OFB64- and DESX-CBC mode and AES in ECB-, CBC-, CFB- and OFB mode). In their experiment they also tested the power consumption of the SSL protocol with different configurations.

The results showed that the asymmetric cryptographic algorithms were the most power consuming algorithms. The symmetric algorithms came second and the least power consuming algorithms were the hash algorithms. In the asymmetric encryption algorithms the key size was a significant parameter. However, in symmetric encryption the extension of the key size is not affected. There were a wide variations in the energy cost of different algorithms in the same group. As examples the 3DES algorithm used 6,04µ J/byte and the RC4 used 3,93µ J/byte. For the asymmetric algorithms there were some interesting findings. As shown in Table 4 the RSA algorithm is the least power consuming algorithm in verifying signatures (decryption with private key). However, the ECDSA is the least power consuming signing algorithm.

The results shows that the selection of cryptographic algorithms are dependent on the selection of algorithms and the specific cases. In the experiment they tested the

Algorithm	Key size bits	Key generation (mJ)	Sign (mJ)	Verify (mJ)
RSA	1024	270,13	546.5	**15,97**
DSA	1024	293,20	313,6	338,02
ECDSA	163	226,65	**134,2**	196,23

Table 4: Energy consumption of asymmetric crypto algorithms on iPAQ

power consumption of SSL with two different cipher suites: ECC-3DES-SHA and RSA-RC5-SHA1. The result showed that the ECC-3DES-SHA cipher suite was the less power consuming suite for small data transaction sizes. However, when the data transaction size exceed 20.36KB the RSA-RC5-SHA1 was the less power consuming suite. The explanations of this phenomenon are that the RSA is more power consuming that ECC, and that this algorithm is used in the initial state. However, the RC5 is less power consuming than 3DES so when the data transaction size increases, the RSA-RC5-SHA1 cipher suite is the least power consuming suite.

2.3 How do security implementations affect the battery's charge interval?

Vectro TEL[19] has developed a cellular phone with voice encryption. The key exchange protocol is Diffie Helman[20] and the encryption algorithm is AES[21]. The encryption scheme is found in figure 2.

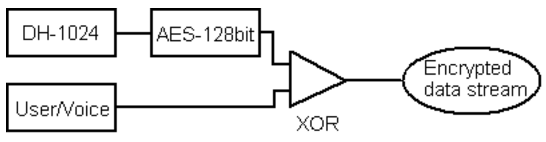


Figure 2: Encryption scheme in VectorTEL X8 and S3

The keys are recalculated each call and deleted directly afterwards.

In the specifications the talk time is different with encrypted and plain conversations. Without encrypted conversations the talk time is up to 5 hours. However with encrypted conversations the talk time decreases with an hour (up to four hours)[22]. This means that the security implemented in this cellular phone consumes $((5-4)/5) =)$ 20%.

In [23] Thull and Sannino analyzed the OMA DRM 2[24] to find out how the DRM standard affects the mobile terminal's[4] processing performance and battery life. They also compared the algorithms in software and hardware to find the time savings due to hardware implementation. Table 5 shows the results from their timing experiment.

Their conclusion say that the hardware acceleration reduces the time of the cryptographic calculations significantly. However, the hardware added to do the cryptographic calculations also consume power. In this article Thull and Sannino have not analyzed the extra power consumption due to introduction of more hardware.

In [25] the mp3.com staff have tested several MP3 players to see whether the DRM protection[24] of music impacts the charge interval. In their experiments they found a

[4]MP3 players, mobile phone with MP3 players etc

Algorithm	Software (cycles)	Hardware (cycles)
AES Encryption	360 + 830/128 bit	10/128 bit
AES Decryption	950 + 830/128 bit	10 + 10/128 bit
SHA-1	400/128 bit	20/128 bit
HMAC SHA-1	1200 + 400/128 bit	240 + 20/128 bit
RSA 1024 Public Key Op	2’160’000/1024 bit	10’000/1024 bit
RSA 1024 Private Key Op	37’740’000/1024 bit	260’000/1024 bit

Table 5: Execution times for different cryptographic algorithms in hardware and software

significant difference of charge interval between DRM protected music and unprotected MP3 files. Some of their results are presented in Table 6.

MP3 player	Unprotected (hours)	DRM protected (hours)	Difference (hours)	Difference %
Creative Zen Vision:M	16	12	4	25
Archos Gmini 402	11	9	2	18
iRiver U10	32	27	5	16
iPod	N/A	N/A	N/A	8
Creative Nomad Jukebox	N/A	4	N/A	N/A

Table 6: Decreased charge interval with the WMA DRM 10 format

In their conclusions they claim that the MP3 player specs should include charge interval when playing DRM protected music due to the power consumption of DRM.

3 Summary of claimed contributions

Literature study has been conducted to find material related to this thesis. Several analysis of power consumption are done. The literature can tell us the energy consumed by RC4 and AES in a laptop with Pentium III processor, energy consumed in a secure wireless session with a Symbol PPT2800^{TM} , the power consumption of several cryptographic algorithms on a Compaq iPAQ H3670 PDA, etc. But what are the consequences of these power consumptions?

We want to know how these security implementations affect the user-friendliness. This cannot be measured in Joule. This thesis describes a method to determine the decrease of charge interval when security is implemented. This is a theoretical model based on input from the battery powered device and the user profile. This method will be exemplified in a cellular phone environment using J2ME[26].

This thesis gives new knowledge by defining a model to find the decreased charge interval when security is implemented based on a profile of use.

4 Method

To answer the research questions, research methods must be considered. This chapter presents the method used in this thesis to attack the problem description.

In almost every cases there exist previous work, work that other authors have done before. The fist one usually do before the researching, is to review the literature to identify what is done in this research area, and what is not done.

> As a researcher, you should ultimately know the literature about your topic *very, very well*[27].

Sources frequently used in this thesis are ACM Portal[28], SpringerLink[29], Science Direct[30], IEEE Explore[31], CiteSeer[32] and the Gjøvik University College Library.

The most interesting and relevant literature is summarized in chapter 2.

To find the decreased charge interval when security is implemented, a model must be used. Since other authors have focused on the energy consumed for each secure session negotiation, or energy consumed for each bit encrypted, such model is not found in the literature study. We need to develop our own model. This model is described in chapter 6. The measurement tool of the model can be various with various accuracy. The requirements to the measurement tools are an accurate measurement tool to find an average current. In many cases, an ammeter is appropriate, however, to get more accurate measurements, an oscilloscope with logging can be used.

When the model is described, an experiment will be carried out. This experimental work takes use of the developed model, and find the decreased charge interval when we want to implement a PGP inspired encrypted messages in a cellular phone environment. This experiment is, as much as a case study, an example of use of the model. Many details are included to preset the way of thinking when finding the decreased charge interval.

At last, an analysis of the experiment is carried out. In this phase, the results of the case study will be discussed. The results of the experiment will then be compared to related work and abstracted to a higher level to make general conclusions.

5 Theory

This chapter introduces a theoretical background for the topic covered by this thesis.

5.1 Where does the energy go?

The energy E, measured in joules, consumed by a circuit is described in the physics as the equation 5.1, where P is the average power consumed by the circuit, measured in Watt, and t is the time the power is consumed. The power is calculated by equation 5.2 where U is the voltage supplied to the circuit, measured in Voltage and I is the current drawn measured in ampere. By applying Ohm's law (Equation 5.3) we can rewrite the energy consumption to equation 5.4 and 5.5 (Resistance in denoted with R and measured in Ω (Ohms)) . Figure 3 shows an overview of a simple circuit.

$$\begin{aligned} E &= P \cdot t && (5.1)\\ P &= U \cdot I && (5.2)\\ U &= R \cdot I && (5.3)\\ E &= \frac{U^2}{R} \cdot t && (5.4)\\ E &= I^2 \cdot R \cdot t && (5.5) \end{aligned}$$

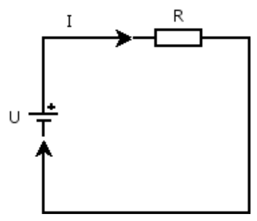


Figure 3: Simple circuit

The previous equations are only valid in a simplified world. In modern microprocessors and other *CMOS* circuits the equations are more complex.

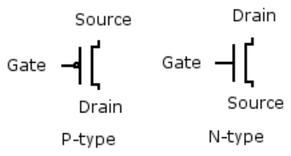


Figure 4: CMOS transistor

A CMOS transistor (figure 4) is like a latch. When the transistor gets a signal on the «gate» it blocks or open the signal between the source and drain. An N-type transistor will block signal on a logical '0' on the gate and a P-type transistor blocks the signal with a logical '1' on the gate. In CMOS transistors there are almost no static power dissipation

With these two types of transistor we can easily make an inverter as in figure 5. When the signal A is logical '1' the upper P-type transistor is closed, the lower N-type transistor is opened and the output-signal Y is logically connected to ground (GND), logical '0'. When the input signal is '0' the upper transistor is opened and the lower is closed. The output signal Y is now logically connected to VCC, logical '1'. In these states the transistors dissipate no power as mention above. However when the signal A is changing there will be a moment when the input signal is VCC/2, and both transistors are half opened. In this moment we have a directly connection from VCC to GND and the CMOS dissipate power. This phenomenon is illustrated in Figure 6.

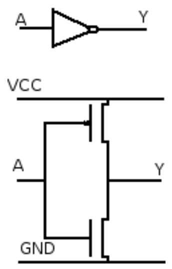


Figure 5: A CMOS inverter

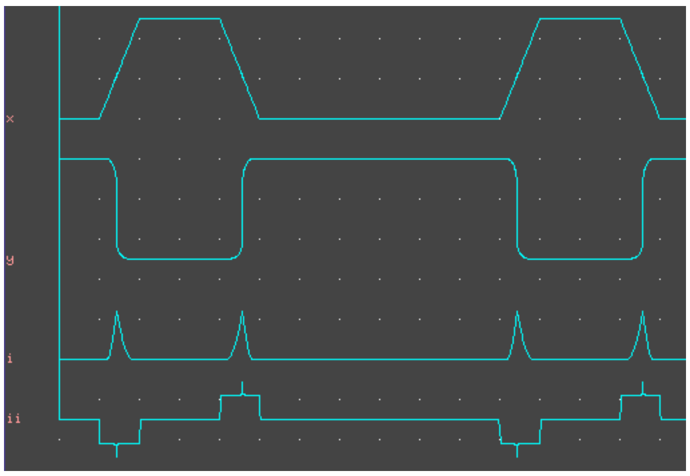


Figure 6: Voltage and current of a simulated inverter

5.2 Security implementations

This thesis defines «security implementations» as all extra activities added to implement security in a battery powered system. The intuitive activity included in this definition is encryption and hash functionality. However, there are more activities added due to security implementations. The following list describes some activities that may affect the battery charge interval:

- Encryption
- Signature generation
- Hashing

- Initializing a secure session
    - Authentication
    - Generating, calculating and exchanging encryption keys (Key Agreement Protocol)
    - Cipher agreement
- Sending extra bits (signature, hash, keys etc)
- Storing extra bits
- User types password
- Digital Rights Management (DRM)
- Certificate validation
    - Validating signature
    - Checking revocation lists
- Compression (not actually security functionality. However compression is used in many protocols to reduce amount of bytes to send or store)

This list shows some examples and is not complete. The activities differ from case to case.

6 Model

This chapter describes a method to find the decrement of charge interval when security is implemented. It describes a mathematical equation to theoretically calculate the decrement of charge interval and describes how it is used by an example.

6.1 Theoretical power consumption model

In this thesis, we are interested in a model to calculate what happens with the battery's charge interval when we include security in a device. To find a theoretical model of the decrease in battery lifetime when security is implemented we need to do some measurements. The decrement is dependent on the following variables:

Symbol	Variable	Description
t	Battery lifetime/ Charge interval	This is the variable we are interested in. It describes the time the device can run with the given condition before it needs to be recharged. The total time is given by the equation $t = t_i + t_e$.
C	Battery capacity	The decrement is dependent of the battery package specification. We need to know the electrical energy stored in the battery ($E = U \cdot I \cdot t$). If we assume the voltage to be stable, the energy can be denoted as $I \cdot t$. Batteries are often denoted with milliamperes per hour (mAh) so this notation is used. Since the voltage is assumed to be stable and we actually do not use the formula for energy, the energy is from now denoted as C (As *Capacity* shorted)
I_i	Idle current	The current drawn while the device is idle is important to find the decrement of battery lifetime. This is for example the current drawn while a cellular phone do not run any java application or illuminated the display.
t_i	Idle time	The time the device is idle. This is unknown value and needs to be calculated in combination with the other parameters.
I_e	Execution current	The current drawn while the device is not idle. For example the current drawn while fetching and storing data in an accelerator logger. This variable also includes all activities to make a user-profile of the device. We need to find all currents drawn in different stages and activities, including security activities. For example the current drawn while calculating the MD5 sum, encrypting with AES or sending extra bits to ensure confidentiality and integrity.
t_e	Execution time	The time of all the activities each time the device is executing. Each execution current(I_e) needs to be timed to find how long time the current is drawn.

F_e	Frequency of execution	The battery lifetime is dependent on the frequency of execution. For example when implementing fingerprints of the stored value in an acceleration logger we need to know how often we log the acceleration. How many MD5 values do we need to calculate each second? Another example: approximately how many secure SMSes do we send each day.

Table 7: Variables of the model

The battery packages are often denoted in milliampere hours (mAh), which is a number of how long time the charge interval (in hours) is when the current drawn is 1 mA. To find the charge interval for a battery powered device with a fixed current and voltage, we can use the simple formula $t = C/I$. To find the decrement of the charge interval when security is implemented we can measure the average current before and after the security is implemented and use equation 6.1. When measuring these currents we may use a capacitor in parallel with the device as in Figure 7.

$$\Delta t = \frac{C}{I_{before}} - \frac{C}{I_{after}} \tag{6.1}$$

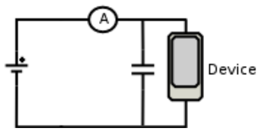


Figure 7: Capacitor in paralell with device

It gets more complicated when we have multiple sporadically activities. Imagine a case where the functionality is executed each minute and the device is in low powered idle mode between the executions. We cannot measure the average power consumption with a single ammeter due to the long time between the executions. In this case we have two activities: an «idle activity» and an «execution activity» with its respective currents (I_i and I_e) and execution times (t_i and t_e). In this case the battery package capacity must be equal to the sum of the respectively currents and times multiplied:

$$C = \overbrace{I_i \cdot t_i}^{\text{Idle charge}} + \overbrace{I_e \cdot t_e}^{\text{Execution charge}} \tag{6.2}$$

What we want to find out is an equation for the battery lifetime(t), which in this case is $t_i + t_e$. However, we do not know the values for t_i and t_e directly since they are dependent of each other.

We need to apply two equations with two unknown variables. To find the other equation we can use proportion. If we assume that each minute the functionality are executed, and the execution takes one second, we can denote that as the following equation:

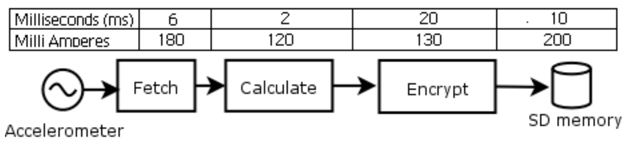
$$t_i = 60 \cdot t_e \tag{6.3}$$

With these two equations (6.2 and 6.3) we can calculate t_i and t_e and finally find $t = t_i + t_e$. This gives the charge interval of the given battery package with the given setting based on the proportions of the times. However, these proportions may get complicated with many activities. To deal with this we suggest a model based on the frequency of execution (F_e). We do not know for how long time an activity totally will execute due to the unknown charge interval (t). However, we may know the frequency of the execution. E.g. we encrypt sensor data four times each minutes or send 15 PGP encrypted SMSes a day. We can now state the following equations:

$$C = \overbrace{I_i \cdot t_i}^{\text{Idle charge}} + \overbrace{\underbrace{t \cdot F_e}_{\text{\# of executions in time t}} \cdot \underbrace{(I_e \cdot t_e)}_{\text{Charge of one execution}}}^{\text{Activitycharge}} \tag{6.4}$$

$$t_i = t - t_e \tag{6.5}$$

In many cases the functionality are divided into subactivities. as in Figure 8



Milliseconds (ms)	6	2	20	. 10
Milli Amperes	180	120	130	200

Figure 8: Multiple subactivities of an activity example

The current consumption of these subactivities may differ. It may be an idea to calculate an average current of the present sub activities. However, in this thesis it is interesting to compare different cryptographic algorithms and key sizes. Therefore, in this model we keep the currents and times separated. The equations for activities with subactivities are denoted in Equation 6.6 and 6.7, where the subactivities are labeled from $a = 0$ to k.

$$C = \overbrace{I_i \cdot t_i}^{\text{Idle charge}} + \overbrace{\underbrace{F_e \cdot t}_{\text{\#executions in time t}} \cdot \underbrace{\sum_{a=0}^{k} (I_a \cdot t_a)}_{\text{Sum of the sub-activities charges}}}^{\text{Activity charge}} \tag{6.6}$$

$$t_i = t - \overbrace{\left(F_e \cdot t \cdot \sum_{a=0}^{k} t_a\right)}^{\text{Total execution time}} \tag{6.7}$$

To find an expression of the total battery lifetime we can combine equation 6.6 and 6.7. The two equations combined, solved to allow for t, gives us equation 6.8. (The full calculations is shown in Appendix B.)

$$t = \frac{C}{I_i + F_e \cdot \left(\sum_{a=0}^{k} (I_a \cdot t_a) - I_i \cdot \sum_{a=0}^{k} t_a\right)} \tag{6.8}$$

This equation works fine in a case where the security is executed each time the origin functionality is executed. An example is the accelerator logger in Figure 8. If we have a battery package of 640mAh and the idle current is measured to be 10mA and the acceleration is fetched one time a second (3600 times an hour) the decrement of charge interval can be calculated as follows:

$$
\begin{aligned}
t_{un-secure} &= \frac{640}{10 + 3600 \cdot \left(\underbrace{\frac{(180 \cdot 6 + 120 \cdot 2 + 200 \cdot 10)}{3600000}}_{\text{Convert ms to h}} - 10 \cdot (\underbrace{\frac{6+2+10}{3600000}}_{\text{Convert ms to h}}) \right)} = 48,71 hours \\
t_{secure} &= \frac{640}{10 + 3600 \cdot \left(\underbrace{\frac{(180 \cdot 6 + 120 \cdot 2 + 130 \cdot 20 + 200 \cdot 10)}{3600000}}_{\text{Convert ms to h}} - 10 \cdot (\underbrace{\frac{6+2+20+10}{3600000}}_{\text{Convert ms to h}}) \right)} = 41,18 hours \\
\Delta t &= 48,71 - 41,18 = 7,52 hours \approx \underline{\underline{7 hours\ and\ 31 minutes}}.
\end{aligned}
$$

In some cases there may be other power consuming activities. To get the charge interval right we need to find the power consumed when the other activities are executed. To describe multiple power consuming activities with different frequencies we use the following notation:

- Every activity is numbered from $a = 0$ to m
- The frequency of activity a is labeled F_a
- The activity's subactivities are numbered from $b = 0$ to n
- A parameter (time or current) of activity a's subactivity b is labeled $t_{a.b}$ or $I_{a.b}$, respectively.

With this notation we can expand Equation 6.8 to Equation 6.9 (See Appendix B for the derivation).

$$
t = \frac{C}{I_i + \sum_{a=0}^{m} \left(F_a \cdot \sum_{b=0}^{n} \left(t_{a.b} \cdot (I_{a.b} - I_i) \right) \right)} \tag{6.9}
$$

This equation gives the charge interval of a given case with given parameters. To find the decrease of charge interval (Δt) when security is implemented we need to calculate the equation two times:

1. t_{before} = (equation without security activities)
2. t_{after} = (equation *with* security activities)
3. $\Delta t = t_{before} - t_{after}$

To state an example of use we can define the following imaginary case:

- We have a battery-powered device that collects and log data from a sensor 20 times each minute.
- Every hour the application sends the logged results by unencrypted WLAN.
- We want the device to sign the logged data.
- We want to apply WEP encryption to the WLAN transmission.

The idle current of the device is 10mA, the battery package has a capacity of 1400mAh and the currents and times are given by Table 8.

Activity	Current	Time
Normal activities		
Fetch data from sensor (20/m)	30mA	10ms
Establish WLAN connection (1/h)	140mA	3s
Send data unencrypted	130mA	20ms
Security activities		
Encrypt and sign (20/m)	25mA	1s
Establish WEP encryption (1/h)	140mA	2s
WEP encryption	130mA	10ms

Table 8: Currents and times of the equation example

Some of the activities consist of several subactivities. E.g. the activity of sending the logged data by WLAN consists of 1: establish the WLAN connection and 2: the transmission of data.

Note that it is important to use the same unit, in this example we convert all units to seconds as follows:

$$\begin{aligned}
1400\text{mAh} &= 1400\cdot 3600\text{mAs} = 5040000\text{mAs}\\
20/\text{m} &= 20/60/\text{s} = 0.33/\text{s}\\
1/\text{h} &= 1/3600/\text{s} = 2.78\text{e-}4/\text{s}\\
10\text{ms} &= 10/1000\text{s} = 0.01\text{s}\\
20\text{ms} &= 20/1000\text{s} = 0.02\text{s}\\
1\text{h} &= 1\cdot 3600\text{s} = 3600\text{s}
\end{aligned}$$

The parameters without security put into the equation gives us:

$$\begin{aligned}
t_{before} &= \frac{1400mAh}{10mA + \underbrace{\frac{20}{m}\cdot(10ms\cdot(30mA-10mA))}_{fetch\ data\ from\ sensors} + \underbrace{\frac{1}{h}\cdot((3s\cdot(140mA-10mA))+20ms\cdot(130mA-10mA))}_{Send\ data}}\\
&= \frac{5040000}{10+6.667e^{-2}+0.109} = 495299s = 137h\ 34m\ 59s
\end{aligned}$$

The parameters with security gives us:

$$\begin{aligned}
t_{after} &= \frac{1400mAh}{10mA + \underbrace{6.667e^{-2}+0.109}_{Before} + \underbrace{\frac{20}{m}\cdot(1s\cdot(25mA-10mA))}_{Enc\ and\ sign} + \underbrace{\frac{1}{h}\cdot((2s\cdot(140mA-10mA))+10ms(130mA-10mA))}_{WEP\ establishment\ and\ WEPencryption}}\\
&= \frac{5040000}{10+6.667e^{-2}+0.109+5+7.256e^{-2}} = 330530s = 91h\ 48m\ 50s\\
\Delta t &= t_{before} - t_{after} = 495299s - 330530s = 164769s = \underline{\underline{45h\ 46m\ 08s}}
\end{aligned}$$

In this case we see that implementation of security in this case costs 45hour and 46 minutes, but is this the final answer to the question of charge interval decrement of the given case?

6.2 Trade-offs

Now that we know the decrement of the charge interval, we can make some trade-offs. Is the decrement acceptable? Is the security acceptable? With smaller keys, the security decrease, however, the charge interval increase. Can the solution be restructured?

We continue with the case from the previous section. When collecting sensor data small values may be collected, and signing these small values may cause much padding. It may be an idea to collect several values and sign a collection of logged values. For instance, a signature is created when 10 blocks of sensor data are collected. This decreases the frequency of the «Encrypt and sign» activity from 20 times each minute to 2 times each minute. A recalculation increases the t_{after} to 130 hours, 15 minutes and 14 seconds. Now we get a Δt of 7 hours, 19 minutes and 44 seconds, a decrement of lost battery time of approximately 38.5 hours.

7 EnergyCalc

7.1 Introduction

As shown in the previous chapter Equation 6.9 may be hard to calculate with a simple calculator or an excel sheet. The problem with an Excel sheet is the complexity of the equation. It is hard to make a general excel sheet that fits all cases when the cases become complex. Also when typing the equation into a calculator it is easy to make typing errors. To make the calculation easier, a java application is created. The application is called EnergyCalc and can be downloaded from `http://www.ztian.org/ec/`.

7.2 Script language

EnergyCalc calculates the decrement of battery lifetime using a script language. To explain the script language we use the case described in Table 8 in the previous chapter.

In the script language we denote a non-security activity with its respective frequency as:

```
A<name> <tab> <frequency> <tab> <domain>
```

Note that the line starts with a capital 'A' and a following name of the activity. However, the name is optional, but gives the reader of the script an overview. Then follows a tabulator to separate the parameters. Further is the frequency followed by a tabulator and the time-domain of the frequency(per day, per hour, per minute, per second or per millisecond). The domain can have the following values:

d	Days
h	Hours
m	Minutes
s	Seconds
ms	Milliseconds

An example is appropriate. «The data is collected 20 times each minutes» is written in the script language as follows:

```
Afetch        20        m
```

When we have defined an activity in the script the following lines are the subactivities of the activity until a new activity is defined. A sub activity has the following syntax:

```
<current in mA> <tab> <time> <tab> <domain>
```

The <domain> may be one of the same domains as the frequency domains above. To state an example of an activity with its subactivities we describe the WLAN connection and sending activity with its subactivities. The execution frequency is one time an hour and consume 140mA in 3 seconds to establish the WLAN connection and 130mA in 20 milliseconds to send the data in plain-text.

```
Awlan      1        h
140        3        s
130        20       ms
```

To add an activity, which has to do with the security, we simply switch the capital 'A' with a capital 'S'.

'#' starts a comment line.

Our example is written in the script language as the following example:

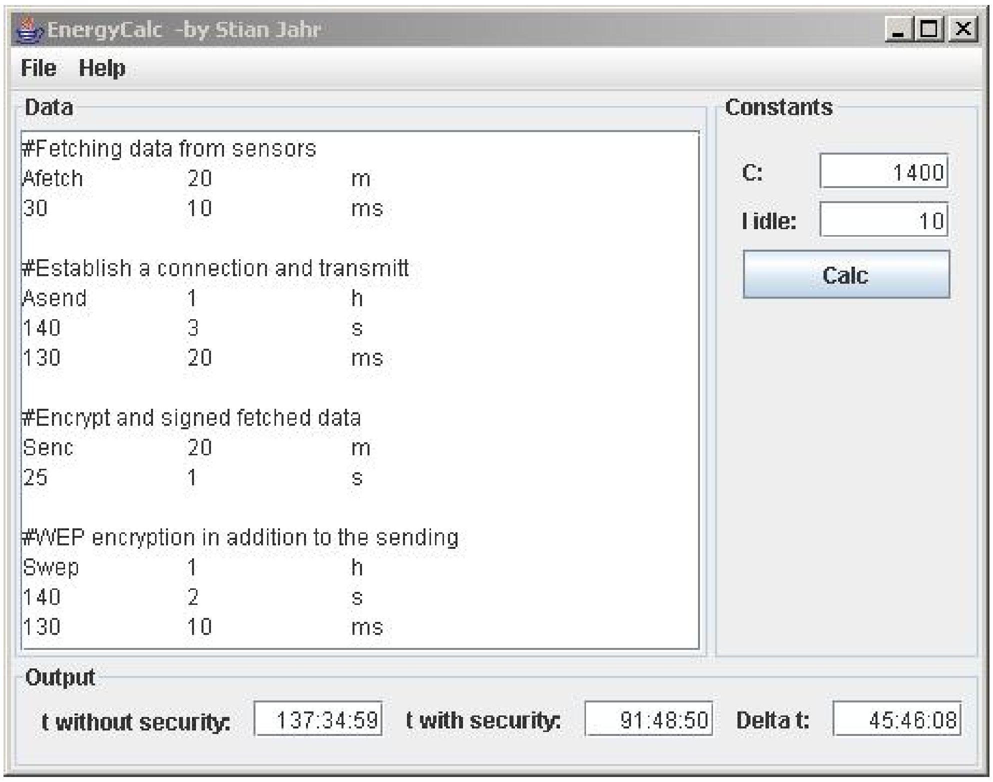
```
#Fetching data from sensors
Afetch    20        m
30        10        ms

#Establish a connection and transmit
Asend     1         h
140       3         s
130       20        ms

#Encrypt and sign fetched data
Senc      20        m
25        1         s

#WEP encryption in addition to the sending
Swep      1         h
140       2         s
130       10        ms
```

When we write this script in EnergyCalc and assign the battery capacity and idle current, EnergyCalc calculates the battery lifetime without security to approximately 137.5h, when security is applied the lifetime is approximately 92h. This gives a decrease of battery lifetime of 45h and 46m (see figure 9). This is a decrement of approximately 66%. To verify the application we see that the times EnergyCalc proposed are equal to the timing calculated in the previous chapter.



```
#Fetching data from sensors
Afetch          20              m
30              10              ms

#Establish a connection and transmitt
Asend           1               h
140             3               s
130             20              ms

#Encrypt and signed fetched data
Senc            20              m
25              1               s

#WEP encryption in addition to the sending
Swep            1               h
140             2               s
130             10              ms
```

C: 1400

I idle: 10

t without security: 137:34:59 t with security: 91:48:50 Delta t: 45:46:08

Figure 9: Screenshot of EnergyCalc with example script

8 Case study: secure SMS

This chapter present an example of use of the model presented in previous chapters. This chapter may contains many details due to try to present the way of thinking when using the model.

8.1 Background

SMS systems of today are not satisfactorily secured, even though they are encrypted with the A5 algorithm. If a phone is stolen, the messages can be read by unauthorized persons. There are also reported weaknesses in the A5 algorithm used in the GSM system[33, 34]. The authentication and integrity is also a concern with the SMS system. Clickatell[1] sells services that makes the customer able to send SMS from a selected sender address (phone number or a self-defined string). If one writes the telephone number of a person known to the receivers address book, the name of the fictive sender will appear in the display. A typing error in the receivers phone number may also have severe consequences when the SMS contains sensitive information.

8.2 Theory

There are approaches which implement secure SMS. In [35], Hassinen and Markovski present a secure SMS scheme applying quasigroup encryption and Java SMS API. They found quasigroup encryption well suited for applications such as SMS encryption due to the low use of memory and its high speed. One SMS of 160 characters using 16 rounds of encryption uses about 4 seconds, and the encryption is done while the user types the receivers phone number.

Pretty Good Privacy (PGP)[36] was developed by Phil Zimmermann in the late 1980s. Zimmermann's idea was to take the advantage of several cryptographic techniques to create a fast and secure framework. The problem solved with PGP is the fact that RSA is a slow cryptographic algorithm[37]. RSA is a public key cipher that can both be used for encryption and digital signatures. The problem with public key cryptography is that the private key is derived by the public key[38]. The strength of RSA is based on the fact that it is practically impossible or extremely time consuming to factorise a multiple of two large prime numbers[37, 38].

The key generation of an RSA key pair is done by the following algorithm:

1. Select two large random prime numbers, p and q
2. Compute $n = p \cdot q$
3. Randomly choose the encryption key e such that e and $(p-1)(q-1)$ are relatively prime.
4. Finally apply the extended Euclidean algorithm to compute the decryption key $d = e^{-1} \bmod ((p-1)(q-1))$

The encryption and the decryption formula is as the following:
$c_i = m_i^e \bmod n$
$m_i = c_i^d \bmod n$

To do these calculations we can use an algorithm for repeated square and multiply for exponentiation in $\mathbb{Z}_n$[39]. The algorithm is described as follows:

Input: $a \in \mathbb{Z}_n$, and integer $0 \leq k \leq n$ represented binary. ($k = \sum_{i=0}^{t} k_i 2^i$)

[1] http://www.clickatell.com/

```
Output: $a^k \bmod n$.
1 Set $b \leftarrow 1$. If $k = 0$ then return($b$).
2 Set $A \leftarrow a$.
3 If $k_0 = 1$ then set $b \leftarrow a$.
4 For $i$ from 1 to $t$ do the following:
    4.1 Set $A \leftarrow A^2 \bmod n$.
    4.2 If $k_i = 1$ then set $b \leftarrow A \cdot b \bmod n$.
5 Return($b$).
```

The bit complexity of this algorithm is $O((\lg n)^3)$. To give an example we can calculate $4^{423} \bmod 1053$ as in table 9

i	0	1	2	3	4	5	6	7	8
k_i	1	1	1	0	0	1	0	1	1
A	4	16	256	250	373	133	841	718	607
b	4	64	589	589	589	415	415	1024	**298**

Table 9: Steps involved to calculate $4^{423} \bmod 1053 = 298$

As the algorithm and example shows there are a multiplication for each '1' in the binary representation of k. One can use this fact to speed up the RSA calculation by selecting a k with few '1's in the binary representation. The three most common choices are 3, 17 and 65537 ($2^{16}+1$, only two '1's in the binary representation so it takes only 17 multiplications to exponentiate.)[38]. X.509 recommends 65537[40], PEM recommends 3[41] and PKCS #1 recommends 3 or 65537[42]. Note that selecting small values of k does not decrease the security, assuming that we pad messages with random values[38].

With p and q sat to small values, an attacker can find the private key by factoring n and find the secret decryption key the same way the benign person found his key: $d = e^{-1} \bmod ((p-1)(q-1))$. This is why large primes are required. In [39] Menezes et al describes some of the factorization algorithms that exist. The fastest factoring algorithm today was invented in 1988 by John Pollard[43, 44]. The algorithm is called Number Field Sieve[45]. In the RSA Laboratories web site[44] it is announced a challenge in factoring large numbers. The largest key factorized today is a 640bit key. The factorization was reported on November 2, 2005 by F. Bahr, M. Boehm, J. Frankie and T. Kleinjung and took approximately 20 2.2GHz-Opteron-CPU years (over five calendar months). In [43], Lenstra and Verheul suggest using RSA key size of 1191bits in the year 2006 to be safe.

As mentioned the RSA encryption is quite slow due to the large keys. However, symmetric encryption is rather fast according to asymmetric encryption. In comparison the RSA is about 1000 times slower than DES in hardware, and about 100 times slower in software[38]. In PGP Zimmermann suggested a combination of symmetric and asymmetric encryption. The idea was to encrypt the message with a generated symmetric session key, and encrypt the session key with the receivers public key. The symmetric cipher Zimmermann proposed in PGP was the IDEA cipher. The encryption protocol is presented in Figure 10 and in the following protocol:

$$\begin{aligned} A \rightarrow B &: \quad C = E_{KU_b}(K_s) \parallel E_{K_s}(M) \\ B &: \quad K_s = E_{KR_b}(E_{KU_b}(K_s)) \\ & \quad\;\; M = D_{K_s}(E_{K_s}(M)) \end{aligned}$$

The following symbols are used:

A: Sender

B: Receiver

M: Message

C: Cipher

k_s: Symmetric key

E_{k_s}: Symmetric encryption

D_{k_s}: Symmetric decryption

KU_b: Receivers public key

KR_b: Receivers private key

KU_a: Senders public key

KR_a: Senders private key

$E_{KU_b}, E_{KR_b}, E_{KU_a}, E_{KR_a}$: Asymmetric encryption with respective key

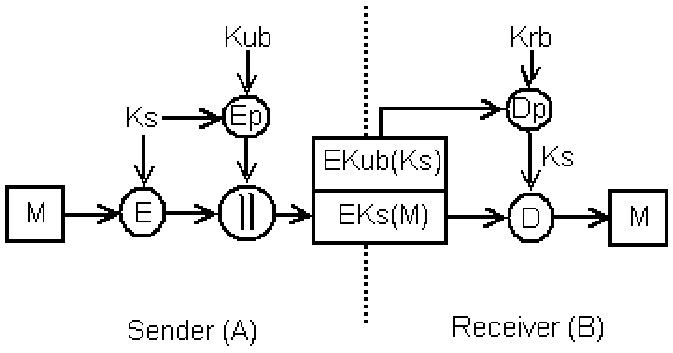


Figure 10: PGP encryption

Above a protocol to ensure confidentiality is presented. As mentioned above, PGP is also used to create digital signatures. To achieve digital signature and still maintain acceptable speed we can apply hashing algorithms. In PGP Zimmermann proposed use of the 128 bits MD5 hashing algorithm. The signature is created by making a hash of the message. This hash is subsequently encrypted with the senders private key and concatenated with the message. Now only the hash of the message is encrypted with a asymmetric cipher. This is more effective than encrypting the whole message. The signing is described in Figure 11 and as the following protocol:

$$\begin{aligned} A \to B &: M \parallel E_{KR_a}(H(M)) \\ B &: E_{KU_a}(H(M)) = H(M) \\ & \text{Yes} : authenticated, \text{No} : failed \end{aligned}$$

The schemes above may also be used in combination to achieve confidentiality, integrity, authentication and non-repudiation. To reduce the size of the message Zimmermann also proposed an optional compression of the message. The compression is performed after the signature and before the encryption. The compression is preferred before the signature because the uncompressed message and the signature can be directly

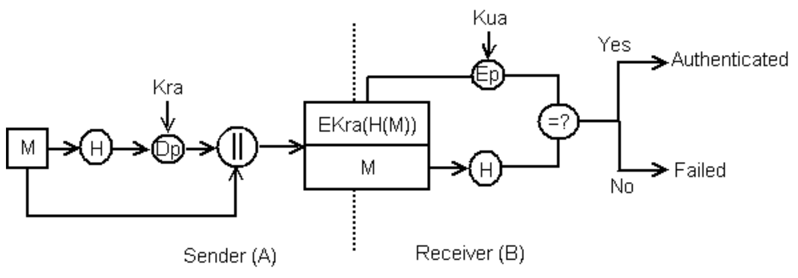


Figure 11: PGP signing

used for future verification[46]. In the encryption case the compression is performed before the encryption because the compression reduces the redundancy of the message[46].

Some mail systems support ASCII text only. To make PGP compatible with these mail systems Zimmermann suggested encoding of the encrypted message or the signature with Radix 64[47, 46]. The Radix 64 encoding divides three 8-bit blocks (24 bits) of raw data to four 6-bits blocks.Each block is encoded into a readable ASCII value. The cipher space consist of small letters (a-z), big letters (A-Z), numbers (0-9) and '+', '/' and '='. This gives us 65 symbols; ($2^6 = 64$) plus a symbol used for padding (=). Each of these 6-bits blocks is then presented as 8-bits ASCII code to be compatible to all mail systems. This encoding leads to an increment of the message by 33%, since three 8-bits blocks are encoded to four 8-bits blocks.

A signed and encrypted PGP message or file is described in [47] and [46] as in figure 12.

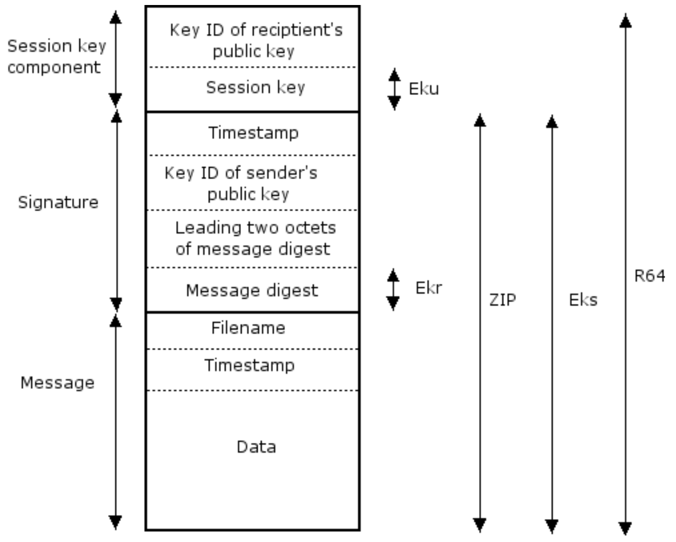


Figure 12: PGP message

As mentioned, compression of the message is optional. Some files are not worth the effort of compression, e.g. JPEG image files[48]. JPEG files are already compressed. When it comes to this case, small text strings are also not suited for compression. To examine the impact of compression in an SMS case we can perform a practical compression test of strings with the length of concatenated SMSes. To automate the compression test one can apply the perl script found in Appendix D. The result of the compression test is found in Table 14 in Appendix A and Figure 13.

The result of the compression test tells us that there is no need to apply compression

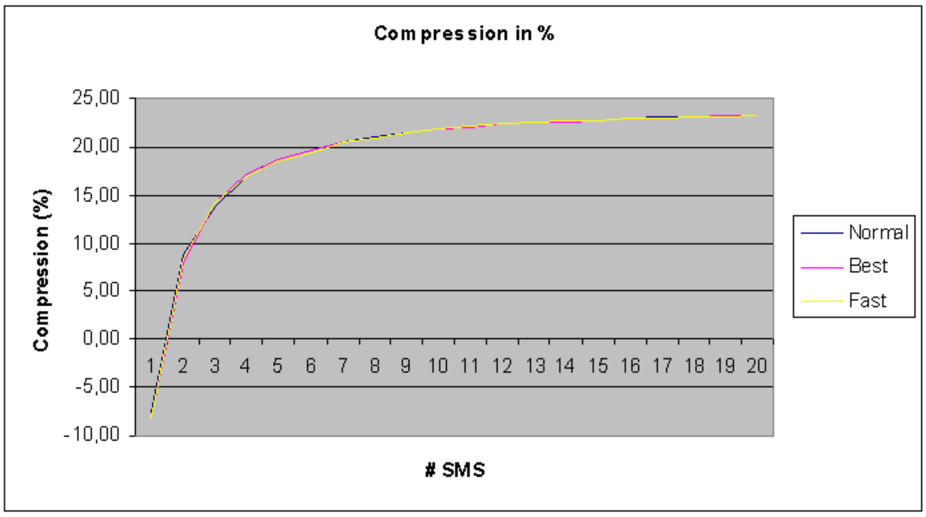


Figure 13: Compression test

if we are going to send short messages. Due to overhead in the gzip algorithm the size of the message actually increase 7,5% with strings as short as 160 characters. The curve in figure 13 shows that the effectiveness of the compression algorithm has a logarithmic increment and stabilize at approximately 24%.

To find the increment of applying a full implementation of PGP as it is implemented in the mail systems we can do a practical approach. In this experiment we use the following software:

- Mozilla Thunderbird v1.5.0.2 (20060308) [49]
- GnuPG v1.4.0 [50]
- OpenPGP provided by Enigmail v0.94.0.0 (20060110) [51]
- GNU Privacy Guard Explorer Extension (GPGee) v1.3.1.192 [52]

We want to test the impact of different key types and key sizes as well. The key types and sizes are respectively RSA vs DSA/ELG and 1024bit vs 2048 bit. In other words, we need to generate four key pairs. To generate the keys we use the enigmail's integrated key management dialog in Thunderbird.

When the keys are made we need to create some test data. To create random SMSes we create strings with length of multiples of 160 characters as we did in the compression test. We use the script described in appendix D without the gzip commands.

When the files are created, we use GPGee to encrypt and sign the files with different keys. We also applied ASCII armor to make the messages applicable by the SMS system (R64 encoding). The result of this experiment can be plotted as the difference between PGP encrypted and signed messages and the messages in plain text as a function of number of concatenated messages. The result is plotted in figure 14

The plot tells us that the amount of extra messages is quite the same with different key types. The amount of messages to encrypt does either affect the number of extra messages to send. However, the key size is an important factor. The difference between 1024- and 2048 bits key are about 2,4 messages (382 bytes).

The increment of cipher text when the key length is expanded has its explanation in how the asymmetric ciphers are calculated. The plain text is divided into blocks with

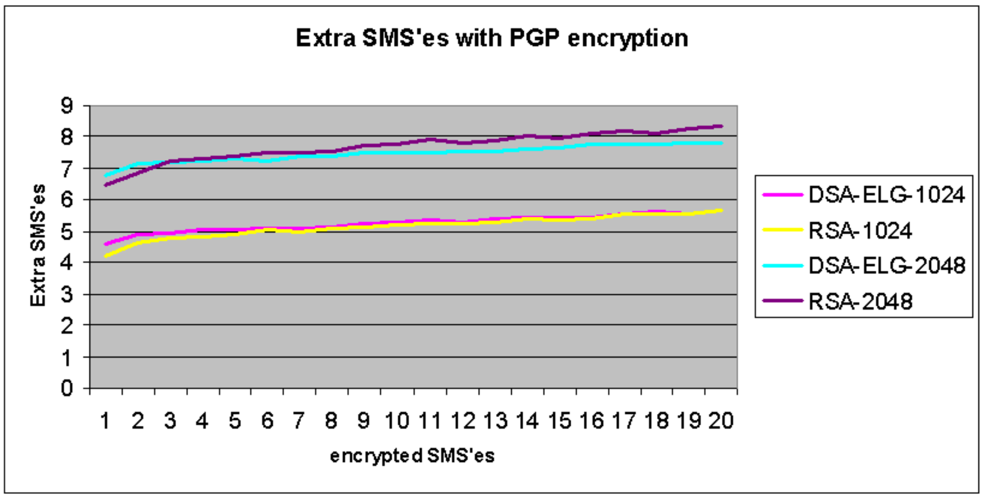


Figure 14: Extra SMSes to send when PGP encryption and signature is applied

the same length as the key. The last block is padded to get the required length. In this case we are encrypting a hash value (160bits, SHA-1) with the senders private key and a symmetric key (128bits) with the receivers public key (see figure 12). Both less that 1024 bits, and therefore padded to fit the key length.

8.3 Experiment

To find the decrement of battery lifetime by sending encrypted and signed SMSes we need to find the current drawn by sending an encrypted SMS. The current drawn in this case may be divided into two parts:

1. the current drawn by the processor during the cryptographic calculations of PGP
2. the current drawn by sending the extra SMSes.

We will try to separate these factors.

To find the current drawn we connected an ammeter between the cellular phone and its battery as shown in figure 15. In this experiment, we used a SonyEricsson Z1010 cellular phone with the specifications in table 10 [53, 54].

Network	UMTS (3G)
Talk time (up to)	3h (UMTS), 4h (GSM)
Standby time	450h
Screen	65k color TFT, 176x220 pixels, illuminated
Batterypack	SonyEricsson Standard Battery BST-15. Li-Polymer 3.6V, 1260mAh
Java virtual processor speed	5.9MHz (5.5 - 6.4)
Heap size (RAM)	511KB
Memory read speed	3907 KB/s (3180 - 4102)
Memory write speed	3813 KB/s (3180 - 4040)

Table 10: SonyEricsson Z1010 specification

The current was measured in different cases to find parameters for our equation. The results of the measurement is described in table 11

To validate these measurements we can compare the measured current with theoretically calculated currents. Table 10 describes a battery package of 1260mAh, and a standby (idle) time of 450 hours. This gives us a theoretically idle current of $1260mAh/450h =$

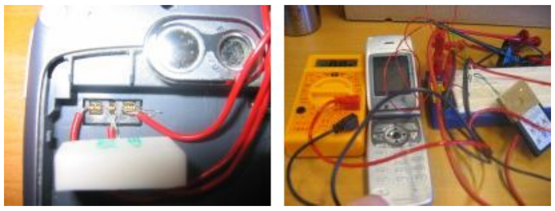


Figure 15: Experiment to find current drawn on a cellular phone

Description	Measured current in mA
Idle. Screen in power-saving mode and no applications running	2.7
Idle with screen active and enlightened	100
Writing an SMS	130
Sending SMS	380
Calling	380
Running a java application with no activity	80
Running an active java application (running encryption test)	160
Opening a WAP page with Opera mini wap browser	410
WAP idle with Opera mini wap browser	220

Table 11: Measurement of current in cellular phone

$2.8mA$, not far from the measured value. Table 10 also mention a talk time of 3h on the UMTS network. This gives us a theoretically calling current of $1260mAh/3h = 420mA$.

The battery-pack is denoted in mAh (milliamperes per hour). To find the decrease of battery lifetime when sending additional messages instead of one we need to find the time of sending SMS. This time is difficult to find with the human eye. To measure this time we can apply video measurement. We can capture the ammeter and the cellular phone screen while sending SMS with a digital camera and afterwards analyze the video to find the time it takes to send SMSes. In this case we captured the mobile screen with a Canon IXUS 400 digital compact camera and analyzed the video in Virtual Dub[2]. The results of the video measurement are denoted in table 12.

Nr. of SMSes	Time in seconds
1	1.6
2	2.1
3	2.7
5	4.3
10	6.5

Table 12: Time of sending SMS

We expect the time to be a linear function. A linear regression of these measured times gives the equation $t_{sms}(x) = 0,554x + 1,112$ with a correlation of 0,993. This means that the connection establishing, initializing and finishing takes 1,112 seconds, and each concatenated SMS takes 0,554 seconds.

To find the power consumption of the cryptographic calculations we implement the algorithms in JavaME. In this experiment we use a free cryptographic library for Java called Bouncy Castle[55]. Bouncy Castle is a widely used API to implement cryptography

[2]http://www.virtualdub.org/

in Java and C# applications. A wide specter of ciphers and hash functions are supported.

In PGP the choice of algorithms and key sizes are optional. In this experiment, we create a test application to time several optional algorithms, modes and key sizes. To get the time we use the build-in timer in Java (System.currentTimeMillis()). To avoid noise we run the cryptographic algorithm several times and calculate the average time. Selected parts of the source code of this application are found in appendix C.

We expect the hash algorithms as the fastest in the test, and the MD5 algorithm as the fastest of the hash algorithms. We expect the time to increase when the size of the hash value increase (higher security). Of the symmetric ciphers, we expect RC4 to be the fastest. This is a stream cipher and is well suited for low memory environment[39]. After the RC4 cipher we expect the DES cipher as the second fastest cipher. AES and IDEA ciphers are expected to be approximately the same time and the 3DES as the slowest symmetric cipher. When it comes to AES BouncyCastle provide a trade-off between speed an memory. The name of the high speed AES is called AES-FAST and the slower, but less memory consuming AES version is called AES-LIGHT. The slowest encryption algorithm is expected to be RSA. We also expect the public encryption to be faster than the private encryption. This because the exponent of the public key is smaller than the exponent of the private key (see appendix C).

The result of running the application on the cellular phone is described in table 15 in appendix A. Remark that the asymmetric encryption is not tested with longer than one message. This because the asymmetric encryption is applied to encrypt the symmetric key and sign the hash value. In this approach we run each algorithm 20 times and calculate the average as described in appendix C. Table 16 in appendix A compares key sizes of the AES algorithm.

To compare the results the results are plotted in figure 16, 18 and 20. To make the plots more comparable 17, 19 and 21 shows the linear regression of the plots. These histograms shows the initialization time and the time of encrypting/hashing one SMS. This gives us the formula: $t = n \cdot \text{time per SMS} + \text{initialization}$.

Some of these results are surprising. We expected all the hash algorithms to be the fastest algorithms, however, RC4 was surprisingly fast and only beaten by the MD5 hash, which was expected to be the fastest algorithm of them all. The explanation of the speed of RC4 is the type of the cipher: Stream cipher. These ciphers are fast and require low memory. Another surprisingly fast cipher was the AES-FAST cipher. Due to its high use of memory it is as fast as the SHA-1 hash algorithm in CBC, CFB and OFB and even faster in ECB mode. The high usage of memory did not seem to be a problem in a J2ME environment with small data blocks to encrypt/decrypt.

Another remarkable result is the speed of the different sizes of the same hash algorithms. As we can see from the plot the SHA-512 is faster than SHA-256. The same way are RIPEMD-256 faster than RIPEMD-160. The logical result should be the other way around. The explanation of this phenomenon is found in the source code of BouncyCastles API. The block size of SHA-256 and SHA-512 are respectively 512bit and 1024bit.

Some of our expectations fitted the results of the test. The 3DES algorithm was the slowest. The 3DES cipher is actually three rounds of the DES cipher with three different keys. This will result in a time of 3DES about three times longer than the DES cipher, which is the result we got. We also see a increment of time consumption when the ciphers runs in different modes of operations. The ECB mode was the fastest and CBC, OFB, and CFB takes approximately the same time. The trade-off between speed and memory (AES-FAST and AES-LIGHT) had also an impact of the encryption time. In ECB mode and ten messages there is a difference of 43 milliseconds. This may be a trade-off to consider, however in this case the choice of cipher does not make a big impact to the decrease of charge interval due to the long time of RSA encryption/signing. The symmetric encryption and hash is to small in proportions to RSA encryption (8ms « 14047ms).

We expected the AES and IDEA to be approximately the same, which we found as a fact. We expected the DES algorithm to be some faster than the AES and IDEA, however, this was not true in this case, they all three executed in pretty much the same speed. This means that the choice between these three ciphers in the BouncyCastle API does not affect the charge interval.

Since the time of symmetric encryption and hash algorithms are so small in proportions to asymmetric encryption (RSA) we can exclude the symmetric encryption and hash calculations. In a PGP encryption we encrypt the symmetric key with the public key and sign the hash of the message with the private key. This gives us a cryptographic calculation of (14047ms + 4170ms=) 18,2 seconds with 1024bit key and (104354ms + 16337ms=) 2minutes with 2048bit key. This may be a long time to wait for the user anyway.

By combining the results of these experiments, we can define a case to insert into our equation. We can imagine the following case:

- We have a SonyEricsson Z1010 with the specifications of table 10 and 11
- We make 2 phone calls of 5 minutes each day
- We use 30 seconds to write an SMS
- We want to encrypt our SMSes with PGP encryption
- We want to see how the decrement of charge interval increase with an increasing amount of SMSes sent each day (A function: $\Delta t(\#SMS)$).
- We want to compare 1024bit key with 2048bit key.

In other words, we want to put the following case into our equation:

Description	Current (mA)	Time
Non-security activities		
Call, 2 times each day		
Calling	380	5min
Send SMS, 1-20 times each day		
User types the SMS	130	30sec
Send SMS	380	1.6sec
Security activities (1024bit)		
PGP encryption and sending, 1-20 times each day		
PGP encryption	160	18.2sec
Send extra bits (5 extra SMSes)	380	2.77sec*
Security activities (2048bit)		
PGP encryption and sending, 1-20 times each day		
PGP encryption	160	2min
Send extra bits (7 extra SMSes)	380	3.88sec**

* $5 \cdot 0.554$

** $7 \cdot 0.554$

Table 13: Example case

From the table we can create the following script to EnergyCalc:

```
Acall     2         d
380       5         m

Asms      1         d
130       30        s
```

```
380       1.6       s

Spgp1024  1         d
160       18.2      s
380       2.77      s

#Spgp2048 1         d
#160      2         m
#380      3.88      s
```

We set the battery charge (C) to 1260 and the idle current (I_i) to 2.7. If we use this script and change the frequency of messages from 1 though 20 and calculate the time of each value we get the charge intervals plotted in figure 22.

8.4 Results and discussion

In this implementation the problem with the user-friendliness is hardly the decrement of the charge interval, especially with 2048bit key. The calculations of RSA takes intolerable long time. 18.2 seconds may be an acceptable time to wait, however 2 minutes is a long time to wait to send a message. In [18] Potlapally et al found the power consumption of RSA-1024 encryption to be 546,5mJ and 15,97mJ with respectively private and public key. The time of the encryption is not mentioned. However we can calculate our results to be comparable with equation 5.1. The energy consumed by an RSA encryption with respectively private and public key are ($3,6V \cdot 160mA \cdot 4,2s =$)2419,2mJ and ($3,6V \cdot 160mA \cdot 14s =$) 8064mJ. The explanation of higher power consumption in our case is the clock frequency of the processor. The frequency of the SonyEricsson Z1010 is 5,9MHz[54]. The processor in Compaq iPAQ H3670 is 206MHz[56].

To see how our times are affected by a faster processor we ran the application on a SonyEricsson P910i. This smart-phone has a processor of 54,5MHz[54]. The results are found in table 18 in appendix A. We see that the RSA calculations is significant faster in P910i than Z1010.

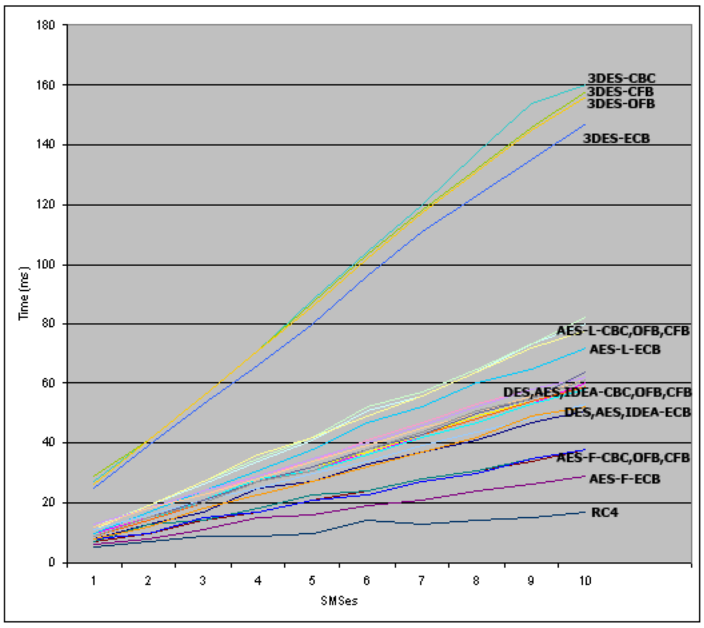


Figure 16: Time of symmetric encryption on SonyEricsson Z1010

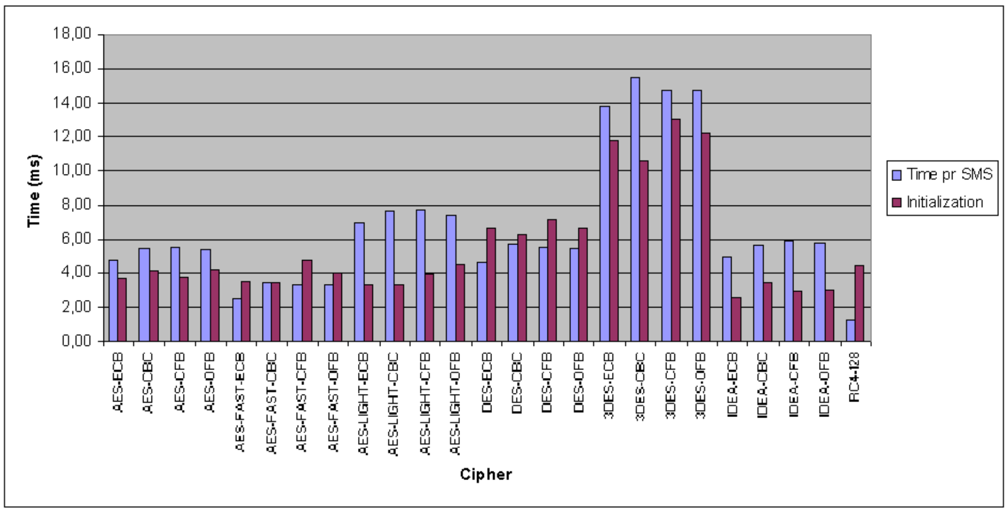


Figure 17: Linear regression of time of symmetric encryption on SonyEricsson Z1010

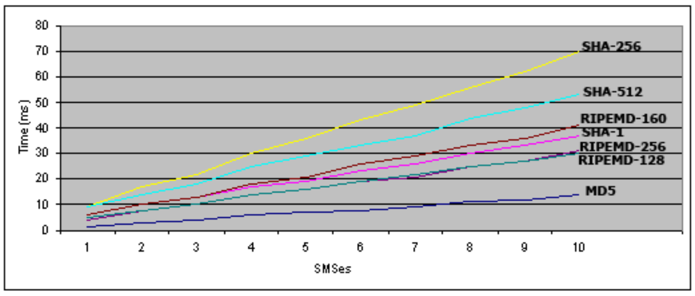


Figure 18: Time of hash algorithms on SonyEricsson Z1010

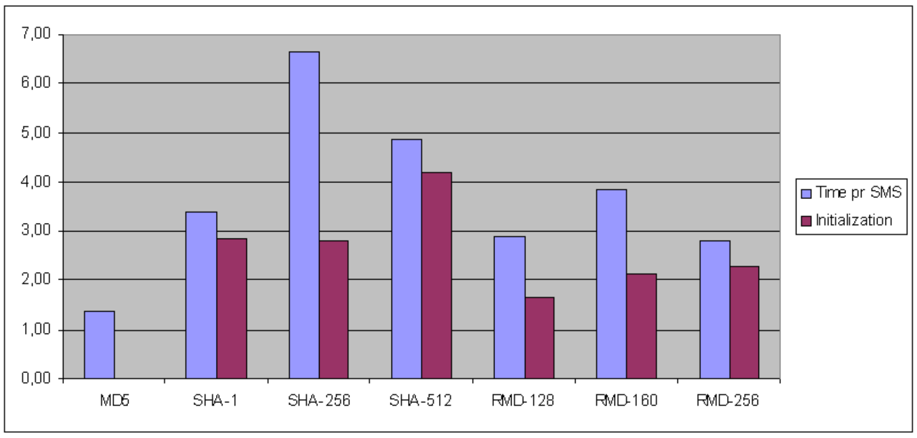


Figure 19: Linear regression of time of hash algorithms on SonyEricsson Z1010

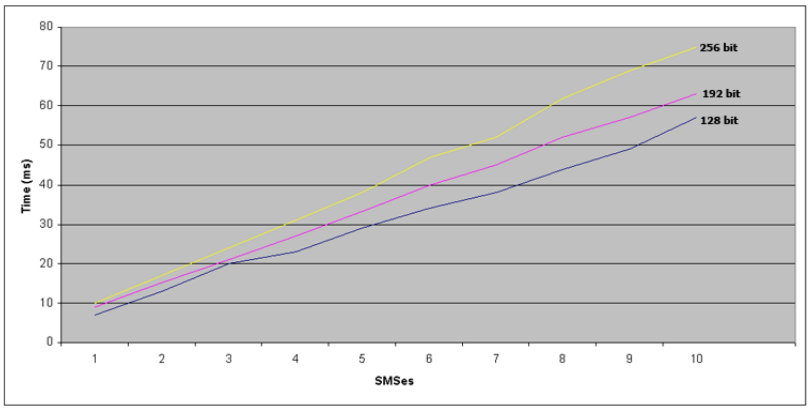


Figure 20: Time of AES with different key sizes on SonyEricsson Z1010

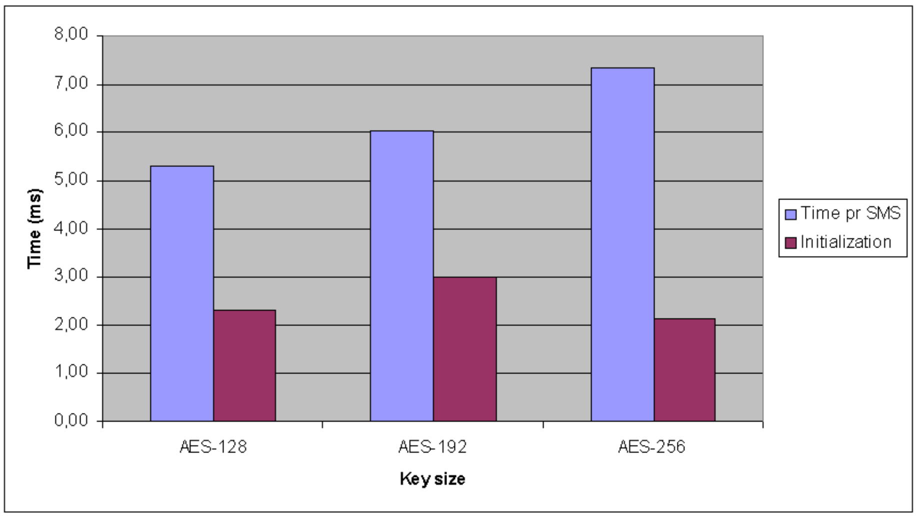


Figure 21: Linear regression of time of AES with different key sizes on SonyEricsson Z1010

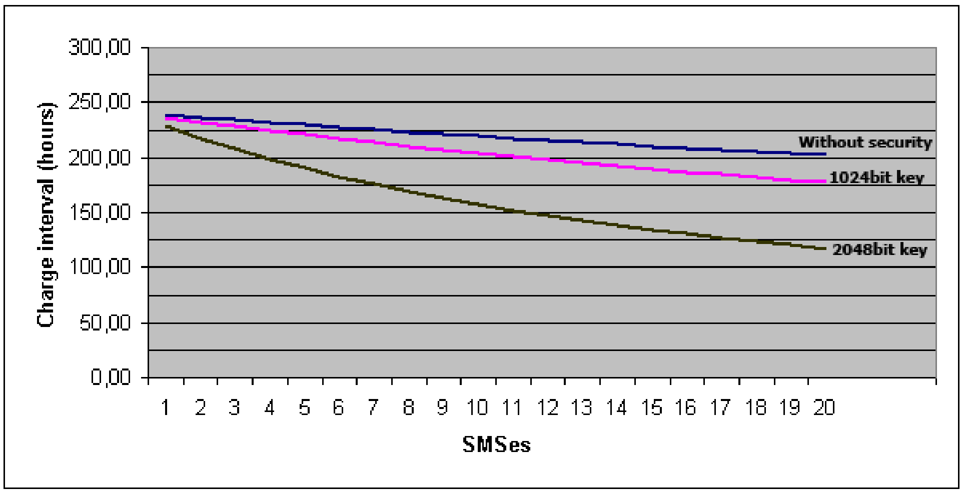


Figure 22: Charge interval as a function of sent SMSes each day

9 General results and discussion

There is no definite answer to the question «How does this security implementation affect on the charge interval?» The impact is dependent of the specific case and how the design. This chapter will try to summarize and draw general conclusions to the question.

9.1 Impacts on the charge interval

9.1.1 Choice of algorithm

The choice of cipher is important to the power consumption. This is tested in the experiments in this thesis and several articles[57, 58, 59]. As expected, the RSA algorithm was the most power consuming algorithm. In our secure SMS case, the power-consumption of RSA was so high that the symmetric encryption and hash algorithm did not play a significant part of the power consumption. In other words the choice of symmetric cipher is more or less irrelevant.

When we do not use asymmetric encryption, we may reduce the decrement of charge interval by selecting the «right» cipher. In our experiment the encryption of one SMS takes 50% more time with AES-ECB-128 than RC4. A difference between the modes of operations is also present. The ECB mode is about a half millisecond faster than the other modes of operation each SMS. Hardly noticeable with few SMSes, however, if the data become large there will be significant differences.

9.1.2 Increased bits to send in communication settings

The expansion of the data to send due to security implementation may play a significant role. Even if the time of the sending is short it may consume much energy[60]. In our case, the current drawn while encrypting the message was 160mA. The current drawn while sending the SMSes was 380mA, 2,4 times more than the encrypting current. The time of encrypting one SMS was 18,2 seconds due to the long time of RSA, and the extra messages was sent in 2,77 seconds. The charge used of each encrypted SMS of the respectively encryption and sending is ($160mA * 18,2s/3600 =$) 0,81mAh and ($380mA * 2,77s/3600 =$) 0,29mAh. This gives an encryption charge of only 2,77 times more than the sending charge. To reduce the power consumption of communication one should consider a compression of the data before sending[14, 18]. Optimize the handshake protocol may also be important to the power consumption[14, 18].

9.1.3 Hardware versus software

In dedicated hardware the encryption routines executes faster than in software and consumes less energy[23, 61, 62]. This will help us reduce the decrement of charge interval. However, the time we will save due to hardware integration is not proportional with the energy savings. The new hardware also consumes power[23].

9.1.4 Design challenges

Low idle current causes the charge interval more vulnerable to activities. The decrement of charge interval is as mentioned dependent of the specific case. When designing battery powered devices with requirements to both security and lifetime. We want secure devices that last as long as possible. Making the idle current as low as possible is important. Today we can buy micro controllers that draw current down to a few μA in sleep mode[63, 64]. This makes the charge interval vulnerable for power consuming activities. E.g. in the following case:

- Functionality is executed once a second

- It takes 130ms to execute
- The encryption takes 130ms
- The working current is 130mA

This makes the following script:

```
Anormal   1        s
130       130      ms

Senc      1        s
130       130      ms
```

We set the battery charge (C) to 640mAh. With I_{idle} sat to 10mA the time without security is 24h, 59m and 59s. The Δt is calculated to be 9h, 27m and 57s. The charge interval is decreased by 38%. If we set the I_{idle} to 0,1mA we get the time without security of 37h, 40m and 33s. The charge interval is reduced by 18h, 46m and 56s. This is 50% decrement.

Even fast encryption may affect the charge interval if frequently executed. If we got a fast processor that encrypts the data in 1ms we may not be concerned about the charge interval. 1ms is fast encryption, however it may affect the charge interval significantly, if it is executed frequently. E.g. 100 times each second as the following script describes:

```
Anormal   100      s
130       1        ms

Senc      100      s
130       1        ms
```

If we set the battery charge to 640mAh and the I_{idle} to 10mA we get a charge interval of 29h, 5m and 27s. The charge interval is decreased with 10h, 16h and 2s. This is a decrement of 35,3%.

Accumulation before encryption/communication If we need to encrypt and send small amount of data each execution, it may be valuable to accumulate collected data before encryption and/or sending. E.g. the following case:

- Idle current is 10mA
- Working current is 130mA
- Data is collected once a second and takes 1ms
- Encrypt one dataset takes 1ms
- Cipher setup and initialization takes 5 milliseconds

We set the charge interval to 640mAh and the idle current to 3mA. The following script initializes and encrypts the data each time a data is collected. The encryption takes (5ms+1ms=) 6ms.

```
Anormal   1        s
130       1        ms

Senc      1        s
130       6        ms
```

Without encryption this device has a charge interval of 204h, 40m and 8s. The encryption decrease the charge interval by 40h, 6m and 8s. If we collect data in a minute before encryption we get an encryption time of (5ms+60*1ms=) 65ms. This gives us the following script.

```
Anormal    1          s
130        1          ms

Senc       1          m
130        65         ms
```

The charge interval without encryption is the same. Now the decrement of charge interval is reduced to 8h, 37m and 32s.

9.1.5 New technology

New technology affects the importance of the design challenges presented above. Faster processors may draw more current, but the time of the encryption is reduced a lot. Running the test application described in appendix C on a SonyEricsson P910 with a Java virtual processor speed of 54.5MHz[54] the RSA-1024 encryption with private key took 1,8 seconds and public key encryption took 0,5 seconds (see table 18 in appendix A). When the encryption time become that much lower in this case, the most power consuming activity due to the security is to send the message expansion due to PGP.

As the processor becomes faster the communication networks also increase in speed. The UMTS (also known as 3G) network offers teleservices (like speech or SMS) with a significant higher speed than the GSM network. The data rate is up to 2048kbits/s[65]. In comparison the GPRS network has a maximum speed of 171,2kbits/s[66].

10 Conclusions

10.1 Summary

The theoretical equation of the decrement of charge interval is given in chapter 5 as the following equations:

$$\begin{aligned}
\Delta t &= \frac{C}{I_{before}} - \frac{C}{I_{after}} \\
t &= \frac{C}{I_i + F_e \cdot \left(\sum_{a=0}^{k} (I_e \cdot t_e) - I_i \cdot \sum_{a=0} kt_e\right)} \\
t &= \frac{C}{I_i + \sum_{a=0}^{m} \left(F_a \cdot \sum_{b=0}^{n} (t_{a.b} \cdot (I_{a.b} - I_i))\right)}
\end{aligned}$$

The two last one gives the charge interval in a given setting, and to find the decrement of charge interval one needs to do the calculations twice, with security activities and without, and find the difference.

10.2 Equation accuracy

The equations summarized in the previous section are based on current and time measurements. We have assumed the voltage delivered by the battery to be stable. The correct model of the voltage shows a decreasing curve over time[67, 68]. Including this phenomenon makes the equation more complicated.

In the experiments presented in chapter 8 we have measured the currents by using an ammeter. There are doubts in using a regular ammeter to measure the current. We cannot create a accurate current profile of the current consumption. A more appropriate way of finding the current consumption is to use an advanced oscilloscope with logging or connection to a computer. Some devices used to find an accurate current consumption might be:

- Mobile PC evaluation board[6, 7]
- Tektronix TDS 3054 Oscilloscope with Tektronix TCP 202 current probe[14]
- SCB-68 I/O connector and computer with LabVIEW[18] from National Instruments[69].

When the current profile is found we can calculate an average current that fits into our equation.

Another drawback with *calculating* the charge interval is the complexity of existing hardware. We may find it difficult to pinpoint all activities that are important to the charge interval with the correct current. E.g. in communication devices: The signal strength may affect on the current drawn while sending data over a wireless channel[70].

The specification of the battery may also vary from the real world. Batteries are assumed to provide a stable voltage and are therefore denoted in mAh (milliampere hours) instead of mwh (milliwatt hours = mAh· voltage). What is denoted in the specification may differ from the really specification, the maximum battery charge may also vary over time as the battery gets older. Since the max battery charge is proportional with the charge interval we may get a bias in the calculated charge interval.

Take these drawbacks in consideration the equation may be a useful tool to make a trade-off between power consumption and security. Former research (as described in chapter 2) has focused on measuring power consumption in mJ ($P \cdot t$). This does not give us an image of the consequences of the power consumption. It gives the power consumption of one activity due to security. In this thesis we have focused on a user-friendliness aspect. The user-friendliness is not measured in amperes, effect or energy. The equation presented in this thesis gives the decrement of charge interval, a result people can deal with and understand the consequences.

There is no concrete answer to the question «Does security decrease the charge interval?». It is dependent of the given case and strength of security. In the case studied in this thesis, the decreased charge interval when we implement a 1024bit PGP encrypted SMSes is not significant. However, if the key increase to 2048bit the charge interval is considerable decreased. The decrement of charge interval is dependent of...

- how long time the security is running each execution,
- the current drawn while the security is executed,
- the frequency of the execution and
- the proportions of the previous parameters in comparison to the normal activity.

According to [1] the power consumption of modern electronics is dependent of how many switches between '0' and '1' in the memory needed to execute, and not the time. This actually means that increasing the clock frequency with result in faster execution does not affect the power consumption. That is a fact with modifications. As the technology evolves, the processors also become more power saving.

11 Further work

This thesis has presented a theoretically calculation of the decrement of charge interval when security is implemented. We have also presented some points to consider before designing the security based on measurements done in chapter 8. The validity of the equation is not tested. How the result of the equation agree with the reality. This needs to be tested.

In the feasibility study of this thesis, we planed to test several platforms such as embedded platforms (micro controllers, FPGA etc) and in a laptops. However, due to time limit this thesis has focused on mobile phones. It should be interesting to see the differences in different platforms.

In this thesis we have found the power consumptions of the ciphers as they are implemented in BouncyCastle[55]. We do not know how optimized these ciphers are implemented and it would be interesting to test other implementations of the same algorithm to see if there is significant differences from the different vendors.

In our case where we have tested how PGP encrypted SMS impact the charge interval. We have only tested when the encrypting is introduced. However, in PGP there may be possible to include key and certificate validation by online revocation lists. This involves more power consuming communication and higher reduction of the charge interval.

The test application for j2ME described in appendix C is tested on two cellular phones: SonyEricsson Z1010 and SonyEricsson P910i (the results are found in appendix A). To reduce the risk of destroying the phone we have not measured the current of P910i. It would have been interesting to test several phones with both times and currents to see who is the most appropriate to use with security and charge interval in mind.

Bibliography

[1] Osmulski, T., Muehring, J. T., Veale, B., Li, J. M. W. H., Vanichayobon, S., Ko, S.-H., & Dhall, J. K. A. S. K. 2000. A probabilistic power prediction tool for the Xilinx 4000-Series FPGA. *Lecture Notes in Computer Science*, 1800, 776–783.

[2] Najm, F. N. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[3] Jagau, U. 1990. Simcurrent-an efficient program for the estimation of the current flow of complex cmos circuits. *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, 396–399.

[4] Veendrick, H. August 1984. Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits. *IEEE Jour. of SSC* , SC-19(4), 468–477.

[5] Bhargava, R., Kargupta, H., & Powers, M. 2003. Energy consumption in data analysis for on-board and distributed applications.

[6] Tiwari, V., Malik, S., Wolfe, A., & Lee, M. 1996. Instruction level power analysis and optimization of software.

[7] Tiwari, V., Malik, S., & Wolfe, A. 1994. Power analysis of embedded software: a first step towards software powerminimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4), 437–445.

[8] Prasithsangaree, P. & Krishnamurthy, P. 2003. Analysis of energy consumption of RC4 and AES algorithms in wireless LANs. In *Global Telecommunications Conference*, volume 3, 1445–1449.

[9] IEEE-P802.11-working-group. 1999. IEEE 802.11 Standard Part 11: Wireless LAN Medium Access Control(MAC) and Physical Layer (PHY) Specifications.

[10] Wi-Fi-Alliance. 2002. Wi-Fi Protected Access (WPA) Standard.

[11] IEEE-P802.11i-working-group. 2002. IEEE 802.11i Draft Supplement to Standard for Telecommunications and InformationExchange Between Systems- LAN/MAN Specific Requirements - Part 11: WirelessMedium Access Control (MAC) and physical layer (PHY) specifications: Specificationfor Enhanced Security.

[12] *The OpenSSL project*. http://www.openssl.org (Last visited Jan 2006).

[13] Prasithsangaree, P. & Krishnamurthy, P. Sept 2004. Analysis of tradeoffs between security strength and energy savings in securityprotocols for WLANs. In *Vehicular Technology Conference*, volume 7, 5219–5223.

[14] Karri, R. & Mishra, P. April 2003. Optimizing the Energy Consumed by Secure Wireless Sessions - Wireless Transport Layer Security Case Study. *Mobile Networks and Applications*, 8(2), 177–185.

[15] Wireless Application Protocol Forum Ltd. Wireless Transport Layer Security. Version 06-Apr-2001.

[16] Karri, R. & Mishra, P. 2003. Analysis of energy consumed by secure session negotiation protocols in wirelessnetworks. *Lecture Notes in Computer Science*, 2799, 358–368.

[17] The Internet Engineering Task Force. IP Security Protocol (IPSec). http://www.ietf.org/html.charters/OLD/ipsec-charter.html (Last visited Mar 2006). (last visited mar 2006).

[18] Potlapally, N. R., Ravi, S., Raghunathan, A., & Jha, N. K. 2003. Analyzing the energy consumption of security protocols. 30–35.

[19] VectroTEL. http://www.vectrotel.ch/ (Last visited May 2006).

[20] Diffie, W. & Hellman, M. E. 1976. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), 644–654.

[21] Federal Information Processing Standards Publication 197. November 2001. Announcing the Advanced Encryption Standard (AES).

[22] Vectrotel. Vectrotel X8 flyer. http://www.vectrotel.ch/files/vectroTEL_S3_FLYER.pdf (Last visited May 2006).

[23] Thull, D. & Sannino, R. 2005. Performance considerations for an embedded implementation of oma drm 2. *Design, Automation and Test in Europe, 2005. Proceedings*, 3, 46–51.

[24] Microsoft. Digital Rights Management (DRM). http://www.microsoft.com/windows/windowsmedia/forpros/drm/default.mspx (Last visited May 2006).

[25] MP3.com Staff. March 13 2006. MP3 Insider: The truth about your battery life. http://www.mp3.com/features/stories/3646.html (Last visited May 2006).

[26] Sun Developer Network. Java Platform, Micro Edition (Java ME). http://java.sun.com/javame/index.jsp (Last visited May 2006).

[27] Leedy, P. D. & Ormrod, J. E. 2005. *Practical Research. Planning and design*. Pearson Education, 8th edition.

[28] The ACM Digital Library. ACM Portal. http://portal.acm.org (Last visited Jun 2006).

[29] Springer. SpringerLink. http://www.springerlink.com (Last visited Jun 2006).

[30] Elsevier. Science Direct. http://www.sciencedirect.com/ (Last visited Jun 2006).

[31] IEEE. IEEE Xplore. http://ieeexplore.ieee.org/ (Last visited Jun 2006).

[32] CiteSeer. Scientific Literature Digital Library. http://citeseer.ist.psu.edu/ (Last visited Jun 2006).

[33] Goldberg, I., Wagner, D., & Green, L. 1999. The (real-time) cryptanalysis of a5/2. In *Rump Session of Crypto'99*.

[34] Barkan, E., Biham, E., & Keller, N. 2003. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *Lecture Notes in Computer Science*, 2729, 600–616.

[35] Hassinen, M. & Markovski, S. Secure SMS messaging using Quasigroup encryption and Java SMS API.

[36] *The International PGP Home Page*. http://www.pgpi.org/ (Last visited Apr 2006).

[37] Salomon, D. 2003. *Data Privacy and Security*. Springer professional computing.

[38] Schneier, B. 1996. *Applied Cryptography*. John Wiley and Sons, Inc.

[39] Menezes, A., van Oorschot, P., & Vanstone, S. 1996. *Handbook of Applied Cryptography*. CRC Press.

[40] CCIT, Recommendation X.509. 1989. The Directory-Authentication Framework. In *Consulantion Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva*.

[41] Balenson, D. *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes and Identifiers*. RFC 1423.

[42] RSA Laboratories. *PKCS #1: RSA Encryption Standard*. version 1.5, Nov 1993.

[43] Lenstra, A. K. & Verheul, E. R. 1999. Selecting cryptographic key sizes.

[44] RSA Security. *RSA Laboratories home page*. http://www.rsasecurity.com/ (Last visited Apr 2006).

[45] Buhler, J., Lenstra, H., & Pomerance, C. 1994. The development of the number field sieve. *Lecture Notes in Computer Science*, 1554.

[46] Rhee, M. Y. 2003. *Internet security: cryptographic principles, algorithms and protocols*. John Wiley & Sons Ltd.

[47] Stallings, W. 2003. *Network security essentials*. Pearson Education Inc, 2nd edition edition.

[48] Wallace, G. Feb 1992. The JPEG still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1), xviii–xxxiv.

[49] Mozilla Corporation. Thunderbird. http://www.mozilla.com/thunderbird/ (Last visited May 2006).

[50] The GNU Privacy Guard. GnuPG. http://www.gnupg.org/(en)/download/index.html (Last visted May 2006).

[51] Mozdev.org. Enigmail. http://enigmail.mozdev.org/download.html (Last visited May 2006).

[52] Exelica.org. GNU Privacy Guard Explorer Extension. http://gpgee.excelcia.org/ (Last visited May 2006).

[53] *Sony Ericsson Z1010 Specification*. http://www.sonyericsson.com/z1010/ (Last visited Feb 2006).

[54] Le Club-Java. TastePhone Server, list of MIDP Java phones. http://www.club-java.com/TastePhone/J2ME/MIDP_Benchmark.jsp (Last visited Apr 2006).

[55] The Legion of the Bouncy Castle. Bouncycastle homepage. http://www.bouncycastle.org/ Last visited Jun 2006.

[56] HP. *QuickSpecs, Compaq iPAQ Pocket PC H3600 Series*. http://h18000.www1.hp.com/products/quickspecs/10632_na/10632_na.HTML (Last visited May 2006).

[57] Gebotys, C. H. 2004. Low energy security optimization in embedded cryptographic systems. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 224–229, New York, NY, USA. ACM Press.

[58] Keeratiwintakorn, P. & Krishnamurthy, P. An energy efficient security protocol for ieee 802.11 wlans. In Press, Corrected Proof, Available online 6 January 2006.

[59] Kiratiwintakorn, P. Energy efficient security framework for wireless local area networks. Master's thesis, University of Kansas, 2000.

[60] Mattisson, S. 1997. Minimizing power dissipation of cellular phones. In *ISLPED '97: Proceedings of the 1997 international symposium on Low power electronics and design*, 42–45, New York, NY, USA. ACM Press.

[61] Karri, R. & Mishra, P. May 2003. Modeling energy efficient secure wireless networks using network simulation. *Communications, 2003. ICC '03. IEEE International Conference on*, 1, 11–15.

[62] Ravi, S., Raghunathan, A., Kocher, P., & Hattangady, S. 2004. Security in embedded systems: Design challenges. *Trans. on Embedded Computing Sys.*, 3(3), 461–491.

[63] Microchip Technology Inc. *PIC18F2455/2550/4455/4550 Data Sheet*, 2004.

[64] Microchip Technology Inc. *Power Management for PIC18 USB Microcontrollers with nanoWatt Technology*, 2004.

[65] UMTS World. 2002. Overview of The Universal Mobile Telecommunication System. http://www.umtsworld.com/technology/overview.htm (Last visited May 2006).

[66] GSM World. GPRS Platform. http://www.gsmworld.com/technology/gprs/index.shtml (Last visited May 2006).

[67] *Evaluating Battery Run-down Performance Using the Agilent 66319D or 66321D with Option #053 14565A Device Characterization Software*. Application Note 1427.

[68] Brorein, E. Using battery drain analysis to improve mobile-device operating time. Technical report, Agilent Technologies, 2002.

[69] National Instruments. LabVIEW. http://www.ni.com/labview/ (Last visited May 2006).

[70] Zhao, Y. & Hsiao, M. 2002. Reducing power consumption by utilizing retransmission in short range wireless network. *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, 6-8, 527–533.

A Tables

A.1 Practical compression test results

SMS	bytes	GZIP normal			GZIP best (-9)			GZIP fast (-1)		
		zipped	delta	%	zipped	delta	%	zipped	delta	%
1	160	172	-12	-7,50	173	-13	-8,13	173	-13	-8,13
2	320	292	28	8,75	294	26	8,13	293	27	8,44
3	480	414	66	13,75	412	68	14,17	412	68	14,17
4	640	533	107	16,72	530	110	17,19	533	107	16,72
5	800	651	149	18,63	651	149	18,63	652	148	18,50
6	960	772	188	19,58	772	188	19,58	773	187	19,48
7	1120	890	230	20,54	890	230	20,54	891	229	20,45
8	1280	1011	269	21,02	1012	268	20,94	1013	267	20,86
9	1440	1132	308	21,39	1130	310	21,53	1131	309	21,46
10	1600	1251	349	21,81	1252	348	21,75	1251	349	21,81
11	1760	1372	388	22,05	1373	387	21,99	1370	390	22,16
12	1920	1491	429	22,34	1489	431	22,45	1491	429	22,34
13	2080	1611	469	22,55	1610	470	22,60	1612	468	22,50
14	2240	1733	507	22,63	1734	506	22,59	1729	511	22,81
15	2400	1852	548	22,83	1853	547	22,79	1853	547	22,79
16	2560	1971	589	23,01	1972	588	22,97	1973	587	22,93
17	2720	2091	629	23,13	2094	626	23,01	2094	626	23,01
18	2880	2213	667	23,16	2213	667	23,16	2212	668	23,19
19	3040	2334	706	23,22	2333	707	23,26	2334	706	23,22
20	3200	2453	747	23,34	2455	745	23,28	2453	747	23,34

Table 14: Practical compression test results

A.2 Times of cryptographic calculations

Algorithm	Time in ms with # concatenated SMSes									
	1	2	3	4	5	6	7	8	9	10
Hash algorithms										
MD5	1	3	4	6	7	8	9	11	12	14
SHA-1	6	10	13	17	19	23	26	30	33	37
SHA-256	9	17	22	30	36	43	49	56	62	70
SHA-512	9	14	18	25	29	33	37	44	48	53
RMD-128	4	8	10	14	16	19	21	25	27	31
RMD-160	6	10	13	18	21	26	29	33	36	41
RMD-256	5	8	10	14	16	19	22	25	27	30
Symmetric encryption in different modes										
AES-ECB	8	13	17	25	27	33	37	41	47	51
AES-CBC	10	15	20	27	31	37	42	47	53	60
AES-CFB	9	15	20	27	32	37	43	49	54	59
AES-OFB	9	14	21	28	31	36	42	47	53	58
AES-FAST-ECB	6	8	11	15	16	19	21	24	26	29
AES-FAST-CBC	7	10	14	17	21	24	28	31	34	38
AES-FAST-CFB	7	13	14	18	23	24	28	31	35	38
AES-FAST-OFB	8	10	15	17	21	23	27	30	35	38
AES-LIGHT-ECB	10	17	24	31	38	47	52	60	65	72
AES-LIGHT-CBC	11	19	26	34	41	51	56	64	73	80
AES-LIGHT-CFB	12	19	27	35	42	52	57	65	73	82
AES-LIGHT-OFB	11	19	27	36	42	49	56	64	72	78
DES-ECB	11	16	21	25	30	34	39	44	48	53
DES-CBC	12	18	23	29	34	41	47	53	58	62
DES-CFB	13	18	24	29	35	40	46	52	58	62
DES-OFB	12	18	23	28	34	39	45	50	56	61
3DES-ECB	25	39	53	66	80	96	111	123	135	147
3DES-CBC	27	41	56	71	88	104	120	137	154	160
3DES-CFB	29	41	56	71	87	103	118	132	146	158
3DES-OFB	26	41	56	71	86	102	117	131	145	156
IDEA-ECB	8	12	18	23	27	32	37	42	49	52
IDEA-CBC	9	14	20	27	31	38	43	48	54	59
IDEA-CFB	9	15	21	27	32	38	43	50	55	64
IDEA-OFB	9	15	20	27	31	38	44	51	55	61
RC4-128	5	7	9	9	10	14	13	14	15	17
Asymmetric encryption (private key / public key)										
RSA-1024	14047 / 4170									
RSA-2048	104354 / 16337									

Table 15: Ciphers compared in SonyEricsson Z1010

Key size	Time in ms with # concatenated SMSes									
	1	2	3	4	5	6	7	8	9	10
128	7	13	20	23	29	34	38	44	49	57
192	9	15	21	27	33	40	45	52	57	63
256	10	17	24	31	38	47	52	62	69	75

Table 16: AES key sizes compared in SonyEricsson Z1010

Cipher	Time per SMS (ms)	Initialization time (ms)	Correlation
MD5	1,36	0,00	0,996463296
SHA-1	3,37	2,87	0,999134193
SHA-256	6,65	2,80	0,999582214
SHA-512	4,87	4,20	0,998684563
RMD-128	2,88	1,67	0,997967433
RMD-160	3,85	2,13	0,999098499
RMD-256	2,79	2,27	0,999075009
AES-ECB	4,77	3,67	0,997869171
AES-CBC	5,47	4,13	0,999162807
AES-CFB	5,58	3,80	0,999597056
AES-OFB	5,40	4,20	0,998506905
AES-FAST-ECB	2,54	3,53	0,997661466
AES-FAST-CBC	3,44	3,47	0,999615873
AES-FAST-CFB	3,33	4,80	0,995884878
AES-FAST-OFB	3,35	4,00	0,997274104
AES-LIGHT-ECB	6,95	3,40	0,999164995
AES-LIGHT-CBC	7,67	3,33	0,999450531
AES-LIGHT-CFB	7,72	3,93	0,999391835
AES-LIGHT-OFB	7,43	4,53	0,999383171
DES-ECB	4,62	6,67	0,999787071
DES-CBC	5,70	6,33	0,999102264
DES-CFB	5,56	7,13	0,999617292
DES-OFB	5,44	6,67	0,999846266
3DES-ECB	13,76	11,80	0,999364245
3DES-CBC	15,49	10,60	0,998532884
3DES-CFB	14,75	13,00	0,999469296
3DES-OFB	14,71	12,20	0,999456501
IDEA-ECB	4,98	2,60	0,99890956
IDEA-CBC	5,61	3,47	0,999364813
IDEA-CFB	5,90	2,93	0,998676573
IDEA-OFB	5,82	3,07	0,999223446
RC4-128	1,24	4,47	0,974501478
AES-128	5,28	2,33	0,99779266
AES-192	6,04	3,00	0,999752112
AES-256	7,34	2,13	0,999269819

Table 17: Linear regression of times in SonyEricsson Z1010

Algorithm	Time in ms with # concatenated SMSes									
	1	2	3	4	5	6	7	8	9	10
Hash algorithms										
MD5	0	1	1	0	1	2	1	1	1	2
SHA-1	0	2	1	2	3	3	3	4	4	5
SHA-256	2	3	5	6	4	5	6	7	7	9
SHA-512	1	2	3	3	4	6	6	7	8	8
RMD-128	1	3	6	5	5	6	6	7	7	7
RMD-160	2	3	4	5	7	7	10	10	10	12
RMD-256	1	2	3	3	4	4	7	5	7	7
Symmetric encryption in different modes										
AES-ECB	0	1	7	3	4	3	4	3	5	6
AES-CBC	1	2	3	3	4	4	4	5	6	7
AES-CFB	1	2	3	3	3	4	5	6	7	7
AES-OFB	1	3	3	3	4	4	5	6	6	7
AES-FAST-ECB	1	1	1	1	1	2	2	2	3	2
AES-FAST-CBC	2	1	1	2	2	3	3	3	3	3
AES-FAST-CFB	0	1	1	2	2	2	3	3	3	3
AES-FAST-OFB	1	1	1	2	2	2	3	3	3	3
AES-LIGHT-ECB	1	2	3	3	5	7	7	7	7	8
AES-LIGHT-CBC	2	3	3	5	6	7	7	8	8	9
AES-LIGHT-CFB	1	3	3	5	6	7	7	7	8	9
AES-LIGHT-OFB	1	3	3	3	6	7	7	7	8	8
DES-ECB	0	0	1	2	3	3	3	3	4	4
DES-CBC	1	2	1	3	3	3	3	9	4	5
DES-CFB	1	1	2	3	3	3	3	4	5	6
DES-OFB	1	1	2	3	3	3	3	4	5	6
3DES-ECB	3	3	5	6	7	8	9	10	11	12
3DES-CBC	2	3	5	6	7	9	10	11	12	13
3DES-CFB	3	3	5	7	6	8	10	12	13	13
3DES-OFB	3	3	5	6	7	8	9	11	12	13
IDEA-ECB	0	1	1	2	3	3	3	4	5	6
IDEA-CBC	0	3	3	3	3	3	4	6	6	7
IDEA-CFB	0	1	1	3	3	3	5	5	7	7
IDEA-OFB	0	1	2	3	3	3	5	6	7	7
RC4-128	0	0	0	0	1	1	1	0	1	2
Asymmetric encryption (private key / public key)										
RSA-1024	2297 / 578									
RSA-2048	13422 / 2062									

Table 18: Ciphers compared in SonyEricsson P910

	Charge interval in hours		
SMSes	Without security	1024bit key	1028bit key
1	238	236	228
2	236	232	217
3	234	228	208
4	232	224	198
5	230	221	190
6	227	217	183
7	225	214	176
8	223	210	169
9	221	207	163
10	219	204	157
11	218	201	152
12	216	198	147
13	214	195	143
14	212	192	138
15	210	189	134
16	208	187	130
17	207	184	127
18	205	182	123
19	203	179	120
20	202	177	117

Table 19: Charge interval as a function of sent SMSes each day

B Formula equations

B.1 Variables

This appendix shows the full calculations of the equation 6.8 in section 6.1. The equations consist of the following variables:

Symbol	Variable	Description
t	Battery lifetime/ Charge interval	This is the variable we are interested in. It describes the time the device can run with the given condition before it needs to be recharged. The total time is given by the equation $t = t_i + t_e$.
C	Battery capacity	The decrement is dependent of the battery package specification. We need to know the electrical energy stored in the battery ($E = U \cdot I \cdot t$). If we assume the voltage to be stable, the energy can be denoted as $I \cdot t$. Batteries are often denoted with milliamperes per hour (mAh) so this notation is used. Since the voltage is assumed to be stable and we actually do not use the formula for energy, the energy is from now denoted as C (As *Capacity* shorted)
I_i	Idle current	The current drawn while the device is idle is important to find the decrement of battery lifetime. This is for example the current drawn while a cellular phone do not run any java application or illuminated the display.
t_i	Idle time	The time the device is idle. This is unknown value and needs to be calculated in combination with the other parameters.
I_e	Execution current	The current drawn while the device is not idle. For example the current drawn while fetching and storing data in an accelerator logger. This variable also includes all activities to make a user-profile of the device. We need to find all currents drawn in different stages and activities, including security activities. For example the current drawn while calculating the MD5 sum, encrypting with AES or sending extra bits to ensure confidentiality and integrity.
t_e	Execution time	The time of all the activities each time the device is executing. Each execution current(I_e) needs to be timed to find how long time the current is drawn.
F_e	Frequency of execution	The battery lifetime is dependent on the frequency of execution. For example when implementing fingerprints of the stored value in an acceleration logger we need to know how often we log the acceleration. How many MD5 values do we need to calculate each second? Another example: approximately how many secure SMSes do we send each day.

Table 20: Variables of the model

B.2 Equation calculations

The equation is a result of a combination of the two following equation:

$$C = \overbrace{I_i \cdot t_i}^{\text{Idle charge}} + \overbrace{\underbrace{F_e \cdot t}_{\#\text{executions in time } t} \cdot \underbrace{\sum_{a=0}^{k} (I_a \cdot t_a)}_{\text{Sum of the sub-activities charges}}}^{\text{Activity charge}} \tag{B.1}$$

$$t_i = t - \overbrace{\left(F_e \cdot t \cdot \sum_{a=0}^{k} t_a\right)}^{\text{Total execution time}} \tag{B.2}$$

Equation B.1 says that the total capacity of the battery package is equals to the energy consumed when the device is idle ($I_i \cdot t_i$) plus the sum of every activity energy each execution ($\sum (I_e \cdot t_e)$) multiplied with the number of executions ($F_e \cdot t$).

Equation B.2 says that the time the device is idle (t_i) equals the total battery lifetime minus the sum of all execution times.

These two equations can be combined to find an expression for t as the following:

$$\begin{aligned}
C &= I_i \cdot \overbrace{\left(t - \left(F_e \cdot t \cdot \sum_{a=0}^{k} t_a\right)\right)}^{t_i} + F_e \cdot t \cdot \sum_{a=0}^{k} (I_a \cdot t_a) \\
&= I_i \cdot t - I_i \cdot F_e \cdot t \cdot \sum_{a=0}^{k} t_a + F_e \cdot t \cdot \sum_{a=0}^{k} (I_a \cdot t_a) \\
&= t \cdot \left(I_i + F_e \cdot \sum_{a=0}^{k} (I_a \cdot t_a - I_i \cdot t_a)\right) \qquad \text{(B.3)} \\
t &= \frac{C}{I_i + F_e \cdot \sum_{a=0}^{k} (I_a \cdot t_a - I_i \cdot t_a)} \qquad \text{(B.4)}
\end{aligned}$$

This equation works fine in a case where the security is executed each time the origin functionality is executed. For example in an accelerator logger. However, in some cases there may be other power consuming activities. For example in a cellular phone situation. We need to find power consumed when we are calling to make the total battery lifetime correct. This can be included by the following equation:

$$C = I_i \cdot t_i + \overbrace{\sum_{a=0}^{m} \left(\underbrace{\underbrace{F_a \cdot t}_{\#\text{ executions of activity } a} \cdot \underbrace{\sum_{b=0}^{n} (I_{a.b} \cdot t_{a.b})}_{\text{Energy consumed by activity } a \text{ each execution}}}_{\text{Total energy consumption of activity } a} \right)}^{\text{Energy sum of all activities}}$$

$$t_i = t - \sum_{a=0}^{m} \left(\underbrace{F_a \cdot t}_{\#\text{ of executions of activity } a} \cdot \underbrace{\sum_{b=0}^{n} t_{a.b}}_{\text{Sum of all executiontimes in activity } a} \right)$$

These equations can be combined as in equation B.1. The result of this combination will give us the following equation:

$$\begin{aligned}
C &= I_i \cdot \overbrace{\left(t - \sum_{a=0}^{m}\left(F_a \cdot t \cdot \sum_{b=0}^{n} t_{a.b}\right)\right)}^{t_i} + \sum_{a=0}^{m}\left(F_a \cdot t \cdot \sum_{b=0}^{n}(I_{a.b} \cdot t_{a.b})\right) \\
&\quad \text{Multiply } I_i \text{ with the inserted } t_i \\
&= I_i \cdot t + \sum_{a=0}^{m}\left(F_a \cdot t \cdot \sum_{b=0}^{n}(I_{a.b} \cdot t_{a.b})\right) - I_i \cdot \sum_{a=0}^{m}\left(F_a \cdot t \cdot \sum_{b=0}^{n} t_{a.b}\right) \\
&\quad \text{Pulling } t \text{ out of the addends} \\
&= t \cdot \left(I_i + \sum_{a=0}^{m}\left(F_a \cdot \sum_{b=0}^{n}(I_{a.b} \cdot t_{a.b})\right) - I_i \cdot \sum_{a=0}^{m}\left(F_a \cdot \sum_{b=0}^{n} t_{a.b}\right)\right) \\
&\quad \text{Applying } k \cdot \sum(a) + k \cdot \sum(b) = k \cdot \sum(a+b) \\
&= t \cdot \left(I_i + \sum_{a=0}^{m}\left(F_a \cdot \sum_{b=0}^{n}(I_{a.b} \cdot t_{a.b}) - I_i \cdot F_a \cdot \sum_{b=0}^{n} t_{a.b}\right)\right) \\
&\quad \text{Applying } k \cdot \sum(a) + k \cdot \sum(b) = k \cdot \sum(a+b) \\
&= t \cdot \left(I_i + \sum_{a=0}^{m}\left(F_a \cdot \sum_{b=0}^{n}(I_{a.b} \cdot t_{a.b} - I_i \cdot t_{a.b})\right)\right) \qquad \text{(B.5)}
\end{aligned}$$

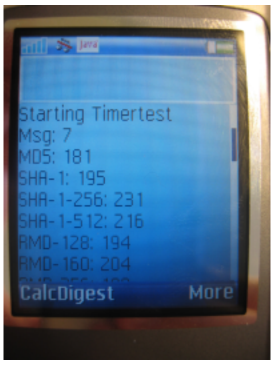
This equation gives us an equation for C. By reformatting the equation and applying the fact that $a \cdot b + a \cdot c = a \cdot (a + c)$ we finally get equation B.6

$$t = \frac{C}{I_i + \sum_{a=0}^{m}\left(F_a \cdot \sum_{b=0}^{n}(t_{a.b} \cdot (I_{a.b} - I_i))\right)} \qquad \text{(B.6)}$$

C J2ME source code

C.1 Introduction

This appendix describes selected parts of the application used to find the time of cryptographic algorithms. The application prints the time of the respective cryptographic algorithm to screen as in figure 23.



```
Starting Timertest
Msg: 7
MD5: 181
SHA-1: 195
SHA-1-256: 231
SHA-1-512: 216
RMD-128: 194
RMD-160: 204
```

Figure 23: CryptotestME screenshot

The time is captured by using the Java integrated clock (System.currentTimeMillis()). The following attributes and methods are defined to take care of the timing:

```
private long lstart;
private long lstop;

public void startTimer()
{
    lstart = System.currentTimeMillis();
}

public void stopTimer()
{
    lstop = System.currentTimeMillis();
}

public long getTime()
{
    return lstop - lstart;
}
```

To make the time more accurate one can run the cryptographic algorithm several times and calculate the average time. To make this one can implement the following methods and attributes:

```
public long getAvg(long[] longs)
{
    long lSum = 0;

    for(int i = 0; i< longs.length; i++)
    {
        lSum += longs[i];
    }

    return lSum / longs.length;
}
```

There are interesting to see how much the time increase when we want to encrypt concatenated SMSes. To increase the number of SMSes one can apply the following code where staIn is a string with a number of how many SMSes we want to encrypt:

```
try
{
    int iIn = Integer.parseInt(staIn);
    if (iIn < 1) appln("To low");
    else {
        String oneSMS = input;
        for (int i = 1; i < iIn; i++) input += oneSMS;
        appln("Msg: " + staIn);
    }
}

catch(Exception e)
{
    appln("No number");
}
```

The helping methods we need are now defined. To apply these help methods and find the average time of the cryptographic algorithms

C.2 Hash algorithms

This section describes the source code to test different hash algorithms with different security. As mentioned in the previous section the time is calculated by an average of different tests. The number of tests is variable and is sat by the `iNumTest` attribute. The `app` and `appln` methods prints the result to the screen. The `update` method calculates the digest of a block of bytes, and the `doFinal` method closes the digest and fills a byte array with the calculated digest[55].

```
int iNumTest = 10;
private String input = "This is a message to test the PGP " +
    "implementation on a mobile phone. This message should " +
    "be 160 chars so I will just write some words, almost " +
    "there, finish. Bye!";

public void digestTest()
{
        app("MD5: ");
        long[] hashTimes = new long[iNumTest];
        byte[] baIn = input.getBytes();

        for(int i=0; i< iNumTest; i++)
        {
                startTimer();
                MD5Digest digest = new MD5Digest();
                digest.update(baIn,0, baIn.length );
                byte[] baOut = new byte[digest.getDigestSize()];
                digest.doFinal(baOut, 0);
                stopTimer();
                hashTimes[i] = getTime();
        }
        appln(Long.toString(getAvg(hashTimes)));


        app("SHA-1: ");
        hashTimes = new long[iNumTest];

        for(int i=0; i< iNumTest; i++)
        {
                startTimer();
                SHA1Digest digest = new SHA1Digest();
                digest.update(baIn,0, baIn.length );
                byte[] baOut = new byte[digest.getDigestSize()];
                digest.doFinal(baOut, 0);
                stopTimer();
                hashTimes[i] = getTime();
        }
        appln(Long.toString(getAvg(hashTimes)));

        [...]
}
```

These two examples of MD5 and SHA-1 continue by replacing the Digest class (MD5Digest and SHA1Digest respectively) with corresponding class for hash algorithm. The classes tested in this experiment are the following:

- MD5Digest;
- SHA1Digest;
- SHA256Digest;
- SHA512Digest;
- RIPEMD128Digest;
- RIPEMD160Digest;
- RIPEMD256Digest;

All classes is part of the `org.bouncycastle.crypto.digests` package of BouncyCastle[55].

C.3 Symmetric encryption

This section describes the methods used to take the time of symmetric encryption. All cipher but one is block ciphers. These block ciphers are tested in different modes of operations. The ciphers tested in the experiment are listed below:

- AES
- AES-Fast
- AES-Light
- DES
- 3DES
- IDEA
- RC4 (stream cipher)

These ciphers are all found in the `org.bouncycastle.crypto.engines` package. As the list describe there are three implementations of AES in BouncyCastle, fast, light and «normal». The difference between these implementations is a trade-off of speed vs memory.

To use a block cipher one need to pad the message to make the length of the message divisible by the block size. The message also needs to be divided into blocks of the given block size. Bouncy Castle contains a class that takes care of both padding and message splitting called `org.bouncycastle.crypto.paddings.PaddedBufferedBlockCipher`. This class pads the message with PKCS7 padding by default.

To apply different modes of operations we use the classes in the `org.bouncycastle.crypto.modes` package. The modes tested in this experiment are ECB[1], CBC[2], CFB[3] and OFB[4].

The code for the encryption/decryption is implemented as the following:

```java
// Generate key
SecureRandom sr = new SecureRandom();
byte[] baKey= new byte[16];
sr.nextBytes(baKey);
byte[] baOut;
byte[] baIn = input.getBytes();

app("AES-ECB: ");
long[] encTimes = new long[iNumTest];
for(int i=0; i< iNumTest; i++)
{
        startTimer();
        // Create encryption object
        PaddedBufferedBlockCipher pbcAes =
                new PaddedBufferedBlockCipher(new AESEngine());
        // Initialize and set the generated key
        pbcAes.init(true, new KeyParameter(baKey));
        // Create a byte array to put the output
        baOut = new byte[baIn.length + 16 - baIn.length%16 ];
        // Encrypt the message
        int outlen = pbcAes.processBytes(baIn,0,baIn.length,baOut,0);
        // Finishing up
        pbcAes.doFinal(baOut, outlen);
        stopTimer();
        encTimes[i] = getTime();
}
appln(Long.toString(getAvg(encTimes)));

app("AES-CBC: ");
encTimes = new long[iNumTest];
for(int i=0; i< iNumTest; i++)
{
        startTimer();
        PaddedBufferedBlockCipher pbcAes =
                new PaddedBufferedBlockCipher(
                        new CBCBlockCipher(new AESEngine()));
        pbcAes.init(true, new KeyParameter(baKey));
        baOut = new byte[baIn.length + 16 - baIn.length%16 ];
        int outlen = pbcAes.processBytes(baIn,0,baIn.length,baOut,0);
        pbcAes.doFinal(baOut, outlen);
        stopTimer();
        encTimes[i] = getTime();
}
```

[1]Electronic CodeBook
[2]Cipher Block Chaining
[3]Cipher FeedBack
[4]Output FeedBack

```
appln(Long.toString(getAvg(encTimes)));

[...]
```

This code continues with different block ciphers and modes described above.

RC4 differs from the other symmetric ciphers owing to the fact that it is a stream cipher. The code to test RC4 is listed below:

```
// Generate key
SecureRandom sr = new SecureRandom();
byte[] baKey= new byte[16];
sr.nextBytes(baKey);
byte[] baOut;
byte [] baIn = input.getBytes();

app("RC4: ");
long[] rc4encTimes = new long[iNumTest];
for(int i=0; i< iNumTest; i++)
{
        startTimer();

        RC4Engine rc4 = new RC4Engine();
        rc4.init(true, new KeyParameter(baKey));
        baOut = new byte[baIn.length];
        rc4.processBytes(baIn,0,baIn.length, baOut,0);
        stopTimer();
        rc4encTimes[i] = getTime();
}
appln(Long.toString(getAvg(rc4encTimes)));
```

The asymmetric encryption algorithm tested in this application is the RSA algorithm, both signing (private key) and encryption (public key). The application also test both 1024- and 2048 bits key. The keys are hardcoded. The source code of the RSA encryption is listed below. Note that the RSA keys are shortened in this listing. The keys are described in table 21

```
app("RSA-1024(r/u): ");

// Signing with private key
startTimer();
BufferedAsymmetricBlockCipher asymChipher =
        new BufferedAsymmetricBlockCipher(new RSAEngine());
// Init with private key
RSAKeyParameters rsapriv = new RSAKeyParameters(true,
        //1024
        new BigInteger("10103640997585173195 [...continue...] 6677357"),
        new BigInteger("37500107161497186367 [...continue...] 67627941")
        );

asymChipher.init(true, rsapriv);
asymChipher.processBytes(input.getBytes(),0,input.length());
byte[] baOut = null;
try{ baOut = asymChipher.doFinal();} catch(Exception e)
{appln("Exeption: " + e.getMessage()); }
stopTimer();
app(Long.toString(getTime()));

// Encryption with public key
```

```
startTimer();
RSAKeyParameters rsapub = new RSAKeyParameters(false,
        //1024
        new BigInteger("10103640997585173459 [...continue...] 6677357"),
        new BigInteger("54398563824747239473 [...continue...] 75734953")
        );

asymChipher.init(false, rsapub);
asymChipher.processBytes(baOut, 0, baOut.length);
byte[] baDec = null;
try{ baDec = asymChipher.doFinal();} catch(Exception e)
{appln("Exeption: " + e.getMessage()); }
stopTimer();
appln("/" + Long.toString(getTime()));
```

1024 bit private key	
Modulus	10103640997585173459354181954885537754643917772167928442842878672 31372280679973947926425556159241864134583650365412754447853901015 04374984761623475486888508165714664614744125762664629576989046970 05023244819633836263447582748114723300266225755273361526155721735 4413966972379946778826350507138317230667166677357
Exponent	37500107174099585614971863679082912619825241670654814715540727378 37887489411333359074358914047747852787795805694740251124566727348 96610947606908156455697625517885615708144872356380701333202988142 91890116773033744176496000374536898426168604445271706949426380329 748251414877995281160739996668486656735367627941
1024 bit public key	
Modulus	10103640997585173459354181954885537754643917772167928442842878672 31372280679973947926425556159241864134583650365412754447853901015 04374984761623475486888508165714664614744125762664629576989046970 05023244819633836263447582748114723300266225755273361526155721735 4413966972379946778826350507138317230667166677357
Exponent	54398563824747239473583242315368435307439873489753425345534535372 4723947237575734953

2048 bit private key	
Modulus	21958957355986203816421592845012462872227401431865614549977041029 54453391192868964517531457797267964105001386488765353794893643808 85154477814474200002994739648263060853848155968420286758840370373 22342266654580094687381362474308259901154285053407058749534509474 29085462559452912629010692056437165521820246095738195040443789216 12353424851938045588952728232701703318388764471314093357082544770 53812503957347566842409125741230513266270470813078957616692511162 04655413661920229811187311271255116690022561940675955710069051095 20466580335398335956844514400186543648604218219541824415378044565 96508244767776375988291553381317
Exponent	72651399238070475385645934046617885155436867617176112224990385006 48525608511949208578195329750756109330265626529002540788600318854 29768074071987069256192212074720895424144312327297700926990133143 47593442609870605002483104490061345017066032171965209275793930754 63381619229418794227273661103170278707139075917262765638130775989 60581204809172795576968949830987005320971915677131363029651640644 25143889737636032167375486981304348445688685344208823629439123513 36126866269232181759368888913448209969419605779767495218858207844 08623058991605290838958137897561917943283558772176476259795805029 0586020958273665711376966718489
2048 bit public key	
Modulus	21958957355986203816421592845012462872227401431865614549977041029 54453391192868964517531457797267964105001386488765353794893643808 85154477814474200002994739648263060853848155968420286758840370373 22342266654580094687381362474308259901154285053407058749534509474 29085462559452912629010692056437165521820246095738195040443789216 12353424851938045588952728232701703318388764471314093357082544770 53812503957347566842409125741230513266270470813078957616692511162 04655413661920229811187311271255116690022561940675955710069051095 20466580335398335956844514400186543648604218219541824415378044565 96508244767776375988291553381317
Exponent	54398563824747239473583242315368435307439873489753425345534535372 4723947237575734953

Table 21: RSA key components

D Perl script for practical compression test

This appendix describes a perl script to make a practical compression test of SMS. The script creates 20 strings of random characters with lengths of respectively 1 to 20 SMSes (160, 320, 480, ...). The strings are stored in files and at last zipped by using gzip. The script generates a `ls -l` list of the compressed files to view the size of the compressed SMSes. This output may be used in combination with awk to filter out only the size of the files (`perl smsgzip.pl |awk 'print $6'`).

```
#!/usr/bin/perl

#############################################################
# Practical comression test of SMS
# Written by Stian Jahr
# Tnx to Guy Malachi http://guymal.com
# 22 Apr, 2006
#############################################################

# This function generates random strings of a given length
sub generate_random_string
{
my $length_of_randomstring=shift;# the length of
 # the random string to generate

my @chars=('a'..'z','A'..'Z','0'..'9',' ');
my $random_string;
foreach (1..$length_of_randomstring)
{
# rand @chars will generate a random
# number between 0 and scalar @chars
$random_string.=$chars[rand @chars];
}
return $random_string;
}

#Generate the random string and save them to files
my $random_string=&generate_random_string(1*160);
open OUTPUT, ">01.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(2*160);
open OUTPUT, ">02.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(3*160);
open OUTPUT, ">03.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(4*160);
open OUTPUT, ">04.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(5*160);
open OUTPUT, ">05.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(6*160);
open OUTPUT, ">06.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(7*160);
open OUTPUT, ">07.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(8*160);
open OUTPUT, ">08.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(9*160);
```

```
open OUTPUT, ">09.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(10*160);
open OUTPUT, ">10.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(11*160);
open OUTPUT, ">11.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(12*160);
open OUTPUT, ">12.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(13*160);
open OUTPUT, ">13.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(14*160);
open OUTPUT, ">14.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(15*160);
open OUTPUT, ">15.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(16*160);
open OUTPUT, ">16.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(17*160);
open OUTPUT, ">17.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(18*160);
open OUTPUT, ">18.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(19*160);
open OUTPUT, ">19.txt";  print OUTPUT $random_string;  close OUTPUT;
my $random_string=&generate_random_string(20*160);
open OUTPUT, ">20.txt";  print OUTPUT $random_string;  close OUTPUT;

#gzip the files
`gzip -f 01.txt`;
`gzip -f 02.txt`;
`gzip -f 03.txt`;
`gzip -f 04.txt`;
`gzip -f 05.txt`;
`gzip -f 06.txt`;
`gzip -f 07.txt`;
`gzip -f 08.txt`;
`gzip -f 09.txt`;
`gzip -f 10.txt`;
`gzip -f 11.txt`;
`gzip -f 12.txt`;
`gzip -f 13.txt`;
`gzip -f 14.txt`;
`gzip -f 15.txt`;
`gzip -f 16.txt`;
`gzip -f 17.txt`;
`gzip -f 18.txt`;
`gzip -f 19.txt`;
`gzip -f 20.txt`;

#Printing output of ls -l. Use this in combination with awk
#to get the size only ( perl smsgzip.pl |awk '{print $6}' )
open (ZIPSIZE, "ls -l *.gz|");
print <ZIPSIZE>;
close (ZIPSIZE);
```