MASTER'S THESIS

Security Impact on Embedded System Performance

Ingus Krumins
2014

Master (120 credits)
Master of Science in Information Security

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Acknowledgments

I would like to thank everyone for their support, comments and suggestions from people in and around the university to organisations they represent. Always hard to start with someone in particular, therefore alphabetic arrangement is used. Special thanks to Dr. Ali Ismail Awad for supervision that allowed this thesis to take to correct shape, Toke Herholdt Groth for his review and specially technology expertise, Tero Päivärinta for making sure that this thesis goes along scientific research path in a right direction and Ross Tsagalidis for his outside view on this research project. As well as Luleå Technical University for its effort in information security field and Swedish Armed Forces for its willingness to collaborate with academic field.

Abstract

This research evaluates complexity of proper security implementation in embedded devices that possess additional challenge of constrained operational environment – e.g. limited memory, limited processing power and limited energy consumption. Term *complexity* express ability (or lack of) to implement appropriate security mechanisms, without need for hardware design changes or time-consuming software modifications.

Actual security requirements are analysed according to ISO/IEC 27001:2013 standard and implemented in publicly available open-source embedded devices and prototypes created during this research. Implementation outcomes are evaluated against previously listed complexity attributes.

Second analysis is performed on data gathered through resource measurements during security requirement implementation and additional experiments. It will attempt to identify relation between additional instructions processing (program code execution) against energy consumption.

Important part of this research consists of practical experiments that are performed with both open-source and commercial hardware. Device operation is tested with applied security measures and without them. Energy consumption is measured 100 to 25000 times per second depending on each case. The results give an insight on what is required for security implementation on embedded devices and what are the implications.

Table of contents

List of figures

List of tables

Abbreviations / Definitions

Abbreviation	Definition
ADK	Accessory Development Kit
AES	Advanced Encryption System
Battery gap	Gap between available and necessary time for the device to function on battery power
CPU	Central Processing Unit
IDE	Integrated Development Environment
ISO	International Organization for Standardization
NIST	US National Institute of Standards and Technology
Processing gap	Gap between available and necessary processing power
PCI	Payment Card Industry
PCI DSS	PCI Data Security Standard
RAM	Random-access Memory
SCADA	Supervisory Control And Data Aquision
SMS	Short Message (or Messaging) Service, a system that enables mobile phone users to send and receive text messages
UAV	Unmanned Aerial Vehicle
ZigBee	Communication protocol for low-power radios based on IEEE 802.15 standard

1. Introduction

By definition [1], embedded device is a system designed to perform specific set of tasks. Typically it has a limited processing power (cpu, memory) and in many cases is being run on battery power. For example, Supervisory Control And Data Aquision (SCADA) systems, unmanned aerial vehicle control systems (UAV, also called as "drones") and credit card payment systems at restaurants.

Embedded devices and self-powered devices have become essential part of our everyday life [2], e.g., a step counter connected to an application in our mobile phone. They are also a significant part of a current trend called "Internet of Things" which is believed to be shaping the future of our lives. As for those "things", e.g. a fridge, typically an additional embedded device will perform all the necessary logical operations, e.g. count how much milk there is left, and communication, e.g. sending SMS.

As information technologies are evolving all the time, so do embedded devices. Therefore it has become a bit complicated to draw a line around embedded device definition. For example, an early phone 15 years ago would perfectly qualify being named an embedded device – it was performing only a specific set of functions. Whereas current phones offer nearly all the functions we expect from a personal computer and thus for a good reason are being more and more often called *computers that we occasionally use to make a phone call as well*.

With a term "processing performance" we understand device ability to perform a specified task within expected timeframe. Like, no one would want to buy a calculator that needs 10 seconds to show result for simple tasks. Just as importantly, additional requirements may not affect overall system goals, e.g.:

a. ability to perform other tasks simultaneously;
b. timely response for main (critical) function execution;
c. preserving battery power for a specified time period.

Processing power is limited due to a simple reason – production costs. As every hardware component put into system has a price tag and it's own functioning requirements (voltage, current, etc.), it's easy to see why those systems are stripped down to minimum. Even if cost saving for a single device is just 2$, in large scale operations that sums up into noticeable amount of money.

Consumer devices possess additional requirements of overall size and noise and heat dissipation. By adding additional components to fulfil extra requirements, one of those factors most likely would be influenced as well and device may loose it's immediate business value.

From security perspective, when we say "data transfer", we think "encrypted data transfer". There are numerous security measures that we consider mandatory and that their implementation goes without saying. However, it is not as easy when it comes to embedded devices with their limitations.

For example, embedded devices are often designed to work on battery power. While battery power can be seen as expandable resource, there are limits in terms of cost effectiveness (cost to travel to a device location and replace batteries) and ability to expand (size and weight limits, especially for flying).

Presumably, fulfilling more requirements means more processing to be made. That, however, means more power consumption, which leads to faster battery drain. From practical side, there is a knowledge gap on how to measure security influence on battery life and embedded device performance that could be used by businesses creating new devices.

During past years we have heard rumours of US drones sending data back to controlling station in unencrypted form [3] and seen reports of hard coded passwords and other security vulnerabilities in various SCADA systems[4]. Such security flaws do not seem acceptable from typical view in IT security industry. Use of default passwords is considered as a serious security problem already, but with hard coded passwords it is even worse, as system owner is not able to change those passwords.

Even if it is technically possible to change those passwords at some parts of the system, other system components will stop working.

And security problems are not really a new subject. IT community has been well aware of them for more than a decade. Also proposed security solutions basically remain the same with only difference of possibly being technically more complex due to evolving threats. Which brings us to a question – what is the root cause of these simple and reoccurring problems.

Security of SCADA systems has been analysed before as they are of particular interest in IT security community due to being often used for controlling critical infrastructure in many countries around the world. A recent research [5] in 2013 have also review SCADA security. They listed typical security threats that SCADA systems are facing and looked at ways of securing them.

This research is aimed at targeting security requirements from embedded device builder perspective. ISO/IEC 27001:2013 [6] standard is used as a baseline for security requirements that are expected in today's environment. Presuming that there are reasons why security vulnerabilities within embedded systems have become a norm rather than separate vulnerabilities on case-by-case basis, outcome of this research should provide an alternative viewpoint for a root cause of these vulnerabilities. The aim is to seek answer for a question, if security requirement implementation is as simple as to flip on "secure switch" or are there some sacrifices to be made.

Thus the potential interest for this research is seen around the whole world for people and companies who design new embedded systems or develop solutions that use embedded systems.

2. Literature review

2.1 Review process

As this research focuses on a rather narrow problem set, keyword combinations;

- "embedded device", "security", "performance" and
- "embedded device", "power consumption" were set as initial search targets.

In case of shortage of previous research, it was planned to extend scope to a more general field of knowledge:

- relation between "security" and "performance".

Previous researches, related to a proposed research, use terms:

1. "battery gap" [7] – shortage of available battery power in order to implement desired changes, that directly relates to *energy consumption*;
2. "processing gap" [8] – shortage of available processing power in order to implement desired changes.

While the changes could be technically made, their implementation might affect primary purpose set for the device, e.g.:

1. function on battery power for at least 7 days;
2. response action has to be made within 20ms.

In such case it is considered that desired changes can not be implemented (without significant re-design of hardware, software or both).

First literature search results mainly contained works created during period of year 2001 to year 2005. Additional related literature was found by searching for works that have used literature found in previous step. It contained up to date researches as well, including literature from year 2014. It's analysis allowed to conclude and justify a reason, which is explained further in this chapter, why there is a lack of current literature that would be primarily relevant to this research.

Literature review has been divided in 3 parts:

- battery gap – where subset of these researches are relevant to RQ1 (Research Question 1) and a topic in a whole being relevant also to RQ2 (Research Question 2);
- processing gap – relevant for RQ2;
- general device limitations – relevant for RQ2.

2.2 Battery gap

Even though a reviewed research [9] was focusing on finding encryption algorithms that require less processing power and therefore would be more suitable for use in embedded devices, they also came to conclusion that "encryption key size has an impact on battery consumption".

Results were based on experiments. However, I cannot agree with the method they used. More specifically – there is no information given about measurements of battery consumption, that leaves me to conclude that built-in battery level indicator (percentage) was used, which is not an accurate type of measurement.

Two years later, in year 2003, a similar research was carried out[7]. This time, comparison was made using more common security algorithms (including MD5, SHA1, SSL) and IPSEC protocol, which are known also today, and using a precise energy consumption measurement technique.

While this research confirmed as well that energy consumption depends chosen encryption algorithm, there is also no comparison for equal situations with and without encryption.

A research in 2005 [10] through performed experiments came to conclusion that energy consumption is directly relevant to execution time. This conclusion has gained popularity and has later been used in researches by other persons as well. Which is understandable, as it creates an easy basis for further work.

And I must object to this simple conclusion. They analysed 4 different encryption algorithms and during their execution all algorithms consumed the same amount of energy (0.6W in a test setup) for all their execution time. This could be potentially explained by CPU limits – CPU of their test device must have been utilized at 100% level during their experiments. That, on the other hand, means that on a different setup (e.g. faster CPU) algorithms in question could use varying amounts of total CPU processing time. If algorithm A uses 40% CPU and algorithm B uses 70% CPU, their energy consumption cannot be compared solely by execution time.

Another work in 2013 claimed to design energy efficient security for VoIP[11]. It showed that in some cases optimisation could have too little benefits. Their proposed algorithm claimed to save 13.2 minutes of GSM standby time, which looking from device usage (business) perspective is not sufficient gain for implementation. Besides, there is no trustworthiness in the results, as the energy consumption calculation method is not well sounded.

2.3 Processing gap

Group of researchers have studied topic of embedded device energy consumption for several years from different perspectives. In year 2004 their research [8] only cited other researches proving battery gap and focused on exploring processing gap. Notable part of their research shows analysis of security implementation in commercial devices. It also reveals additional hardware (design changes) being used in order to fulfil security requirements.

Once again, there is no comparison of impact on processing performance with and without added security measures. Which might be explained by their view that “cost/risk analysis is necessary to be made for system-specific case”.

A research [12] during experimental testing proved that added security protection has an impact on both energy consumption and processing power. Their conclusion was that added security needs to be balanced as device power consumption otherwise turns out to be "prohibitively expensive". A processing gap was proved by comparison of no security and added security, which typically resulted in ~30-50% additional

processing time for standard functions performed by the device and sometimes even reached more than 100% of additional processing time.

While their measuring technique is clear and viable, their experimental setup used just one mobile device. Moreover, they implemented operationally expensive security measures (rootkit detection, data integrity checks) that are rarely used even in standalone systems due to increased workload.

Most recent research in this field focusing on designing embedded device specific communication protocols [13], took battery gap and processing gap for granted (only listing references to related researches) and focused on optimising private key, more specifically Advanced Encryption System (AES), and public key algorithms for faster performance and lower energy consumption.

They did achieve remarkable outcomes in terms of processing performance resulting in some cases even in 25 times faster encryption. However, recent years have shown that cryptography is a complicated subject. And small optimizations can lead and have actually led to ability to compromise remote systems - attacks like [14] “padding oracle attack”, “BEAST”, “CRIME”. For that reason, cryptographic algorithms often include code structures that are not optimal on purpose, to protect from side channel attacks. Therefore cryptographic code optimisations may be used only when properly review by cryptographers.

2.4 General embedded device limitations

Another research [15], that is not directly relevant to a proposed research, because it was focused on embedded device side channel attacks, also concluded that “constrained resources of embedded devices pose significant new challenges to achieving the desired levels of security”. They expressed opinion that embedded devices should have an advantage in achieving security due to being “application-limited”. But during recent years, time has shown that rising popularity among embedded devices mean that nowadays that are neither “application-limited” nor secure.

Literature review also showed an opinion [16] that adding security to embedded devices can be hard - due to processing limitations and additional financial costs.

Contrary other researches, authors of a theoretical research [17] came to conclusion that "authentication and key exchange protocols using optimized software implementations of public-key cryptography are very viable on small wireless devices". Although technical base for their conclusions can not be seen even at theoretical level.

2.5 Review conclusions

Overall, there is consensus that embedded devices suffer from battery gap and processing gap. Starting from year 2006, researchers take battery gap and processing gap for granted by referencing to previous researches made in year 2005 or before. Typically researches before year 2006 fall into two categories:

1. come to conclusion using questionable methods;
2. perform proper analysis but do not provide baseline measurements (without added security) for comparison.

This leads me to conclude that some of the previous work should be repeated (in a similar setup) and improved in order to answer questions for my research proposal. And community should also benefit from availability of more recent data, not just 9 year old results.

I also did not come over a framework that would allow measuring energy consumption without setting up special equipment. One research claimed to reveal such methodology, but I could not agree with the conclusions (*see Battery gap section*). Interestingly though, this method has gained popularity in further researches.

In recent years, researchers have more focused on designing energy efficient algorithms or optimising existing security algorithms. As I expressed before - *recent years have shown that cryptography is a complicated subject. Small optimizations can lead and have actually led to ability to compromise remote systems. For that reason, cryptographic algorithms often include code structures that are not optimal on purpose.*

As a result, there is a knowledge gap to be filled with a proposed research that would:

1. focus on standard security algorithms (well known algorithms with community approved implementations);
2. give an update on previous researches with up to date technology and provide additional perspectives for proposed measurements;
3. seek if it is possible to estimate energy consumption without laboratory grade equipment.

3. Research questions

Research questions have been developed based on literature review, common assumptions and authors personal experience prior this research. As a result, research questions consist of hypotheses that are set to be tested during the research:

- Research question No 1 – energy consumption:
  - Hypothesis No 1 – embedded device energy consumption is proportional to program execution time [10];

  - Hypothesis No 2 – embedded device energy consumption is proportional to used components and utilisation level of every single component;

  - Hypothesis No 3 – consumed energy can be estimated based on systematic calculations.

    The aim is to seek if an easier way to estimate security influence on battery life (energy consumption) exists, without need of specialized equipment by only using parameters that can be observed in development environment with readily available tools (program execution time, amount of CPU usage, etc.).

- Research question No 2 – embedded device design limitations:
  - Hypothesis No 4, based on literature review – embedded device characteristics (*see "design limitations" in literature review chapter*) allow security requirement implementation at the level required by ISO/IEC 27001:2013 standard.

    Choice of international ISO standard as a benchmark for necessary security mechanisms is explained at chapter 6.1.

Research question will verify if security requirements can be implemented to their full extent and without affecting device ability to perform it's main purpose.

3.1 Scope and limitations

This research will:

1. use existing open source solutions utilizing embedded devices (to have the ability to modify their design) – for example, ArduPilot (unmanned aerial vehicle control system).
2. create simple proof of concept SCADA system (analyse sensor data, perform a simple action based on sensor data, transmit data to a central server), using Arduino open source prototyping platform.

Arduino prototyping platform has been chosen due to it's, arguably, gained popularity and ease of use for Arduino IDE. Ease of use is an important factor for software (in general) to gain popularity and to become widely used around the world. That would result in more embedded device prototypes being built using chosen IDE and compatible hardware platform. An example for it's use is Google Accessory Development Kit (ADK) for Android platform (e.g. creating a custom hardware that connects to Android phone). It is unclear about ADK 2013 version, however, ADK in year 2011 and 2012 was based on Arduino IDE[18].

While end devices that are produced in large quantities are custom build and their software is being rewritten in lower level programming language, it is believed to play significant role in outcome of the final product. Meaning that, if prototype is being built in an unsecure manner, it is likely to be implemented the same way in the final product as well. As we also have to consider cost factor – additional functionality (including security) implementation will take time and resources, will delay start of production that is expected to bring profits and cover research and development costs.

Physical security of embedded devices will not be evaluated during this research. That means that this research will focus more on their communication and secure access.

Purpose of this research is to evaluate how complicated it is (or is not) to implement necessary security solutions; purpose is not to justify lack of proper security. It would give general insight for device builders to support their resource estimations in order to build a secure device.

3.2 Research method

Hypothesis testing [19] through creation of experimental designs is used to disapprove stated research questions.

The main rule in hypothesis testing is that it is not possible to approve hypothesis. As is takes only one sample to disapprove hypothesis, it may be missed during the research or even for a prolonged period of time. Therefore the focus of hypothesis testing is to disapprove it.

As the focus of a research typically is on approving proposed theory rather than disapproving it, in order to comply with hypothesis testing, a null hypothesis (H_0) is put up to disapprove it and an alternative hypothesis (H_A) is given which is considered to be true if H_0 can be proved false.

In order to be able to disapprove null hypothesis, it must be falsifiable[20]. Meaning that it's definition should possibly allow for observing a false situation within planned period of the research.

During the research:

- security measures that should be applied in typical IT environment are identified according to ISO/IEC 27001:2013 standard;
- security measures identified in previous step are implemented;
  Goal of security design implementation is to prove that implementation is (or is not) possible. As soon as the goal is reached, further implementation may be ceased.
- implications of added security measures on embedded device performance are observed;
- it is evaluated if device is still able to perform it's main functions and, if not, could it's hardware be possibly upgraded to satisfy performance needs.

4. Experimental work

Aim of this research was to carry out practical experiments in order to answer research questions derived from theoretical base. Experiment environment consisted of:

- standard personal computer (PC) for communication with embedded devices as a party which performance is significantly higher, thus not affecting embedded device operation in any way;
- XBee radio modules that are used for radio communication between embedded devices;
- Raspberry Pi embedded computer with a flexibility of typical computer system and characteristics of embedded device;
- oscilloscope for energy consumption measurement;
- variable power supply unit to provide power at a level necessary for each device or component.

Table 1: Experiment and research hypothesis relation

Experiment No	Applicable hypothesis	Short description
1	Hypothesis 1, Hypothesis 2	Experiment is carried out to observe energy consumption in different data transfer modes (encrypted and unencrypted)
2	Hypothesis 1, Hypothesis 2, Hypothesis 3	Experiment 1 repeated using wireless data transfer
3	Hypothesis 1, Hypothesis 2	Experiment is carried out to perform the same task (encrypting data) under different conditions
4	Hypothesis 1, Hypothesis 2, Hypothesis 3, Hypothesis 4	Experiment tests energy consumption with and without data encryption for a remote sensor that could be a part of a SCADA system.

4.1 Energy consumption measurement technique

Measurements of embedded device energy consumption that are going to be considered as a true consumption, relies on basic physics – specifically Ohm's law. A low resistance resistor (1 Ω) is being used between the device and power supply unit and voltage drop across the resistor is measured.

Power is supplied using variable power supply at a voltage required by the device. Voltage drop is measured between the variable power supply and the device. Power is supplied:

- directly to device's power pins, if they are easily accessible;
- to device's external power connector, if a particular test device does not have exposed power pins; the device is then powered using it's own internal power regulator.

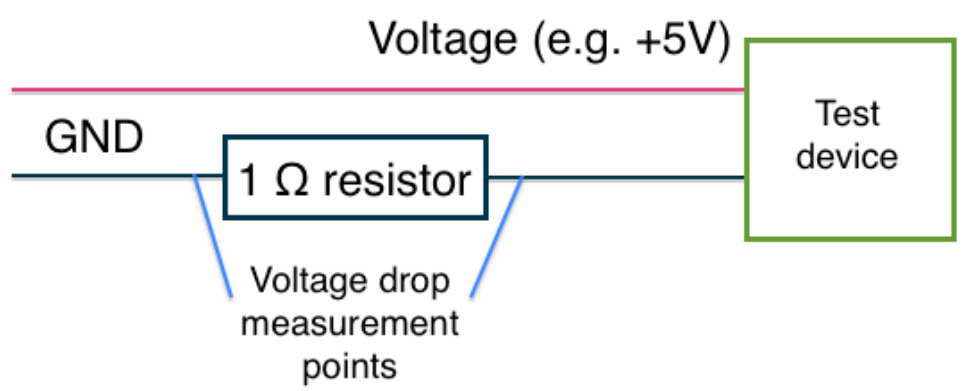

Figure 1: Equipment setup for energy consumption measurement

Voltage drop measurement is performed using an oscilloscope and measurements are made:

- 100 times per second for continuous processes that are executed for more than 10 seconds;
- more than 100 times per second for processes that are executed periods shorter than 10 seconds – shorter period of execution time allows more frequent sampling.

Choice of measurement frequency is limited by available technical equipment for this research. In addition, energy consumption reading (visual) is also observed on variable power supply unit as an additional indicator for measurement accuracy.

Obtained measurements are then summed up and divided by time spent during the test, calculating total consumed energy that is then being expressed in mAh.

4.2 Experiment 1. Data transfer via HTTP vs HTTPS

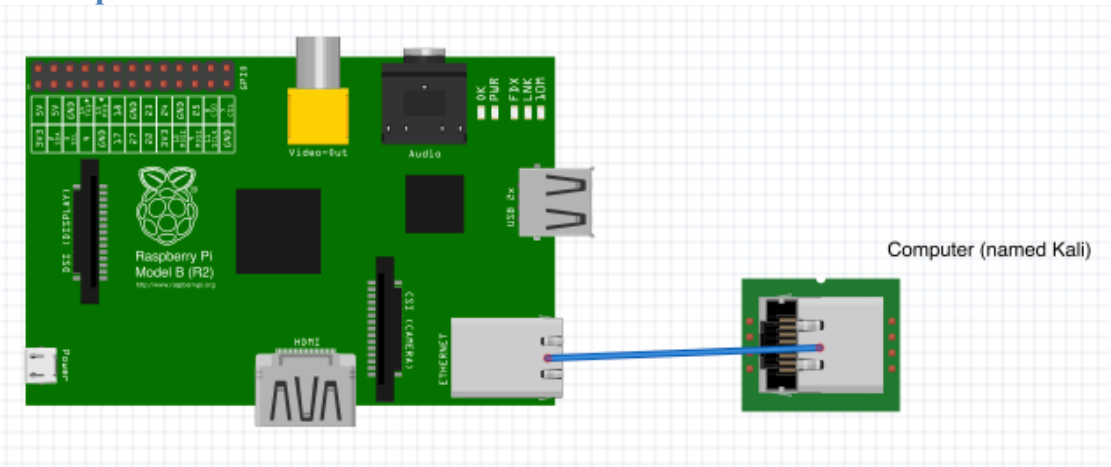

Figure 2: Setup for experiment 1

For this experiment power is supplied at 5.0V to a device's power connector. Data was transferred from Raspberry Pi Ethernet port (which is technically connected through USB interface) directly with a cable connected to a test computer Ethernet port.

Data for transfer was used from a pseudo file (Linux OS feature, contents of file are generated by CPU on request) in order to minimize impact of delay that can be caused for reading the data from storage device. Data consisted of "0"s and was 200MB (200000000 bytes) in size.

In order to minimize result interferance by other processes it was decided to ommit headers of HTTP protocol. That way it is possible to perform data transfer with minimal software interferance and therefore providing results based on hardware capabilities. Overall HTTP headers would add a constant size of data overhead during all transfer modes.

HTTP data transfer was executed using following system commands:

```
root@kali:~/ltu# socat TCP4-LISTEN:80,fork,reuseaddr
FILE:/dev/null
root@raspberrypi:~# dd if=/dev/zero bs=500000 count=400|socat
STDIN TCP:10.0.1.9:80
```

HTTPS data transfer with compression was executed using following system commands:

```
root@kali:~/ltu# socat OPENSSL-
LISTEN:443,fork,reuseaddr,cert=cert.pem,verify=0 FILE:/dev/null
root@raspberrypi:~# dd if=/dev/zero bs=500000 count=400|socat
STDIN OPENSSL:10.0.1.9:443,verify=0
```

HTTPS data transfer without compression was executed using following system commands:

```
root@kali:~/ltu# socat OPENSSL-
LISTEN:443,fork,reuseaddr,cert=cert.pem,verify=0 FILE:/dev/null
root@raspberrypi:~# dd if=/dev/zero bs=500000 count=400|socat
STDIN OPENSSL:10.0.1.9:443,verify=0,compress=none
```

Experiments were carried out without any other user processes running on both systems and they were started with system load average reading of 0.0 (zero).

Table 2: Experiment 1 results

Transfer mode	Energy consumed (mAh)	Time (sec)	Data packets sent	Transfer rate (MB/s)
HTTP	2.798256	23.34	58761	8.6
HTTP	2.801122	23.25	57931	8.4
HTTPS compressed	5.377167	43.24	48389	4.6
HTTPS compressed	5.495011	44.17	48681	4.5
HTTPS	6.848933	57.21	149017	3.5
HTTPS	7.025511	58.45	148179	3.4

Measurements of this experiment mainly focus on consumed energy (mAh) and execution time (seconds) in order to calculate ratio of consumed energy per 1 second. This data is then used later in this work for hypothesis analysis. Other parameters – data transfer rate and data packets sent – are recorded as well, to ensure that experiment results are not affected by unnoticed factors (e.g. reaching limits of network interface theoretical speed).

Overall results show that plain HTTP data transfer is the fastest data transfer method in the experiment setup. HTTPS with compression is limited by CPU speed – as transfer rate is only half of what was achieved with plain HTTP and other factors are the same. HTTPS without compression mainly suffers of increased amount of data packets (nearly 3 times more) that is caused by encryption process, as input data (size and content) is the same for all 3 modes.

4.3 Experiment 2. Data transfer via HTTP vs HTTPS using wireless

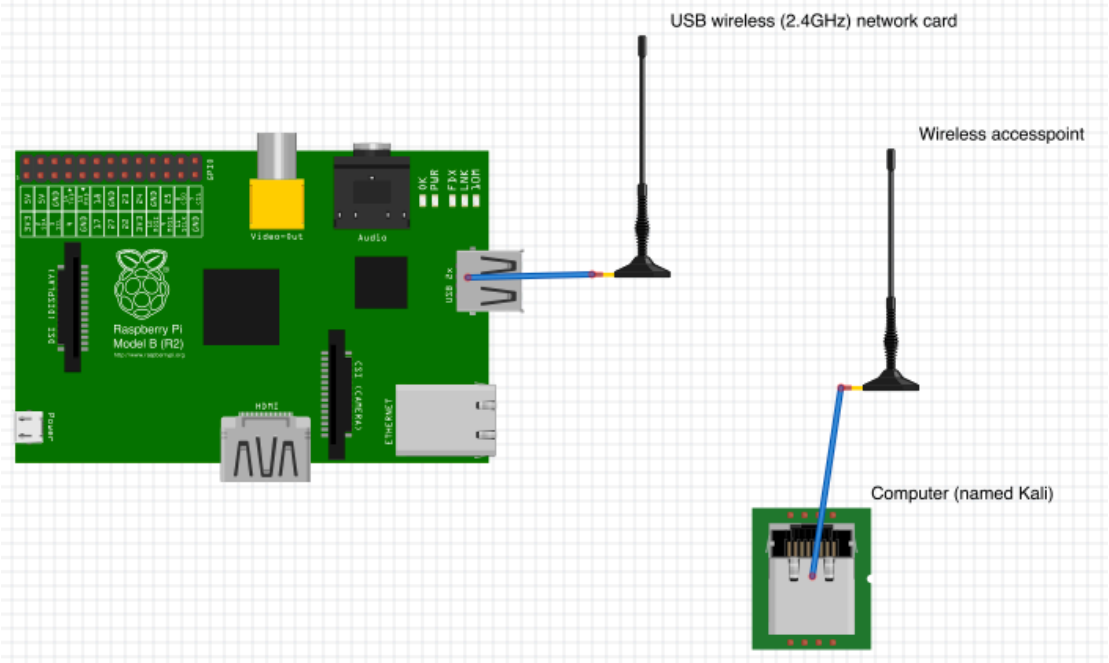

Figure 3: Setup for experiment 2

For this experiment power is supplied at 5.4V to a device's power connector. Raise in power was necessary for the wireless card to function properly. Data was transferred from Raspberry Pi USB wireless card to a test computer Ethernet port that was connected directly to the wireless access point (WPA2 in PSK mode) with Ethernet cable.

Data for transfer was used from a pseudo file (Linux OS feature, contents of file are generated by CPU on request) in order to minimize impact of delay that can be caused for reading the data from storage device. Data consisted of "0"s and was 200MB (200000000 bytes) in size.

As experiment unfolded unexpected results, it was repeated with other data source modes:

- a file on device storage (SD card) cached in memory;
- a file on device storage (SD card) read directly from storage avoiding memory cache.

File was created with random characters generated by pseudo file /dev/urandom (Linux OS feature) with size of 200MB (200000000 bytes).

Data transfer was executed using the same system commands from Experiement 1. For follow-up measurements, input file (*if=/dev/zero*) was substituted with:

- *"if=200MB_file"* for cached file, energy consumption measured for 2nd consecutive data transfer;
- *"if=200MB_file iflag=nocache"* for file red directly from storage device (SD card).

Table 3: Experiment 2 results

Transfer mode	Energy consumed (mAh)	Time (sec)	Data packets sent	Transfer rate (MB/s)
HTTP (/dev/zero)	30.165713	145.246	210652	1.4
HTTP (cached file)	30.390653	146.076	210938	1.4
HTTP (file not cached)	30.481328	145.862	211954	1.4
HTTPS compressed (/dev/zero)	13.469259	65.934	46374	3.0
HTTPS compressed (cached file)	13.943216	67.621	45887	3.0
HTTPS compressed (file not cached)	14.100565	68.437	45795	3.0
HTTPS (/dev/zero)	38.983740	189.275	218040	1.1
HTTPS (cached file)	39.767077	192.559	217755	1.0
HTTPS (file not cached)	40.076089	193.933	217844	1.0

Measurements of this experiment mainly focus on consumed energy (mAh) and execution time (seconds) in order to calculate ratio of consumed energy per 1 second. This data is then used later in this work for hypothesis analysis. Other parameters – data transfer rate and data packets sent – are recorded as well, to ensure that experiment results are not affected by unnoticed factors (e.g. reaching limits of network interface theoretical speed).

Results from this experiment contradict results from previous experiment. As they are also against initial expectations, measurements were repeated several times to ensure their validity. Reason behind compressed HTTPS being fastest data transfer method in this setup has to be significantly smaller amount (nearly 5 times) of data packets that are being sent out by network interface. It must be affected by data encapsulation specifics into wireless network data frames by the wireless network card drivers. It could also be affected by the wireless network configuration (WPA2 in PSK mode) in experiment environment, however, reasons behind this phenomena were not explored as they were outside the scope and timeframe of this research.

4.4 Experiment 3. Data encryption under various factors

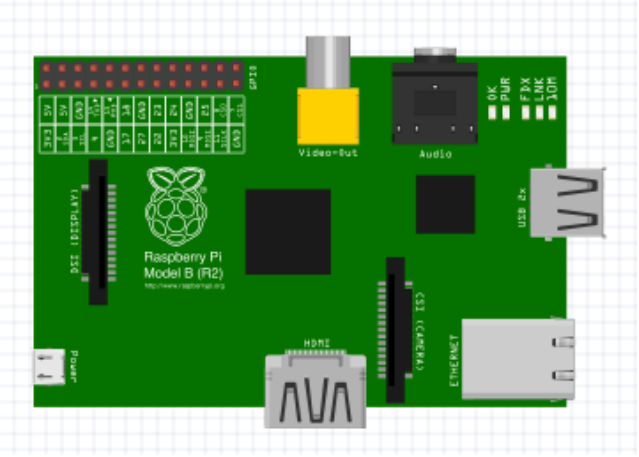

Figure 4: Setup for experiment 3

For this experiment power is supplied at 5.4V to a device's power connector. Raised power level (from 5.0V) was not technically necessary for this experiment, however, it was left unchanged from previous experiment to provide greater consistency for measurement environment.

This experiment was set to measure the same process – data encryption using AES algorithm with 256bit encryption key in CBC mode – under different conditions:

- source data taken from pseudo file (CPU generated);

  ```
  dd if=/dev/zero bs=1000000 count=200 | openssl aes-256-cbc -a -salt -out /dev/null -k 1
  ```

- source data taken from cached file (RAM), 1st run not measured to allow system to cache file in memory;

  ```
  dd if=200MB_file| openssl aes-256-cbc -a -salt -out /dev/null -k 1
  ```

- source data read from storage device (SD card).

  ```
  dd if=200MB_file iflag=nocache| openssl aes-256-cbc -a -salt -out /dev/null -k 1
  ```

Encryption output was discarded to avoid additional interference for test data. Idle power consumption for 25 second period was measured and calculated to be 0.183592 mAh per second.

Table 4: Experiment 3 test results

Input data	Device's consumed energy (mAh)	Test process consumed energy (mAh)	Test process energy consumption (mAh per second)
Pseudo file	5.336022	0.487358	0.018454
Cached file	4.696978	0.373386	0.015855
Cached file	6.141306	0.486672	0.015801
From storage	7.061161	0.688683	0.019841
From storage	8.045628	0.804759	0.020405

Measurements of this experiment still focused on ratio of consumed energy per 1 second. This data is then used later in this work for hypothesis analysis. In order to minimize device ambient (idle) energy usage impact on test process ratio calculation, average idle power is deducted from total energy consumption recorded during execution of test process.

Overall results showed that accessing random access memory (RAM) has the least impact on energy consumption compared to pseudo file (CPU generated) access or accessing file on storage device (SD card). These results support Hypothesis 2, by showing that overall energy consumption may be affected by different components used during program execution process.

4.5 Experiment 4. Data sampling and transfer using XBee radio modules

Figure 5: Experiment setup for wireless (ZigBee protocol) data transfer

Power is supplied at 3.0V directly to power pins of XBee device. Energy consumption measurement is performed using oscilloscope at a rate of 25KHz (25000 times per second).

Xbee (Router AT) with directly attached analogue temperature sensor performs data sampling and sends samples to receiving Xbee (Coordinator API). Data is delivered from receiving Xbee to a computer (USB serial cable) at position number 1250 (50ms from start of measurement). One position equals 0.00004 seconds.

A temperature sensor was connected to XBee radio transmission module. XBee module is configured to:

- sleep for 28 seconds (consuming minimal power while in sleep mode);
- sample sensor data (read a temperature measurement from directly attached thermo resistor);
- send data to a central radio module which is connected to a central data collection device.

For this experiment, central radio module was connected to a computer USB serial port. Sleep mode, according to module technical specification consumes less than 45microampers. Such energy consumption level provides negligible impact on overall consumption and is technically complicated to measure therefore it was not measured during this experiment.

Data transmission was performed in unencrypted mode and encrypted mode. Measurements were made at 25KHz rate (25000 measurements per 1 second) on:

- transmitting radio module energy consumption;
- time taken for cycle execution (marked by rise and drop of energy consumption);
- moment in time for receiving radio module starting to output data to a computer.

Table 5: Energy consumption, execution time, data delivery latency measurements without encrypting transfered data

Start time (ms)	Finish time (ms)	Total time (ms)	Consumed energy (A)	Consumed energy (mAh)	Time from start to data output on receiving module (ms)
0.00732	0.04992	42.64	20.5472	0.000228302	33.72
0.01108	0.04984	38.80	20.6600	0.000229556	29.96
0.01144	0.04988	38.48	20.4808	0.000227564	29.60
0.00972	0.04988	40.20	21.9272	0.000243636	31.32
0.01032	0.04992	39.64	21.7512	0.000241680	30.72
0.00920	0.04972	40.56	22.3048	0.000247831	31.84
	Average:	**40.05**	**21.2785**	**0.000236428**	**31.19**

Table 6: Energy consumption, execution time, data delivery latency measurements with encrypted data transfer

Start time (ms)	Finish time (ms)	Total time (ms)	Consumed energy (A)	Consumed energy (mAh)	Time from start to data output on receiving module (ms)
0.00260	0.04812	45.56	27.2888	0.000303209	38.44
0.00332	0.04832	45.04	26.9160	0.000299067	37.72
0.00692	0.04828	41.40	23.4984	0.000261093	34.12
0.00472	0.04816	43.48	25.3744	0.000281938	36.32
0.00456	0.04832	43.80	25.6400	0.000284889	36.48
0.00336	0.04812	44.80	26.9376	0.000299307	37.68
0.00504	0.04816	43.16	24.8560	0.000276178	36.00
0.00404	0.04816	44.16	26.2792	0.000291991	37.00
0.00316	0.04812	45.00	27.0200	0.000300222	37.88
0.00360	0.04824	44.68	26.4800	0.000294222	37.44
0.00484	0.04800	43.20	25.1792	0.000279769	36.20
	Average:	**44.03**	**25.9517**	**0.000288353**	**36.84**

Measurements of this experiment focused on ratio of consumed energy per 1 second, which is used later in this work for Hypothesis 1 analysis. Second measurement group focuses on execution time and latency between start of execution cycle and data delivery on central computer, which is used for Hypothesis 4 analysis.

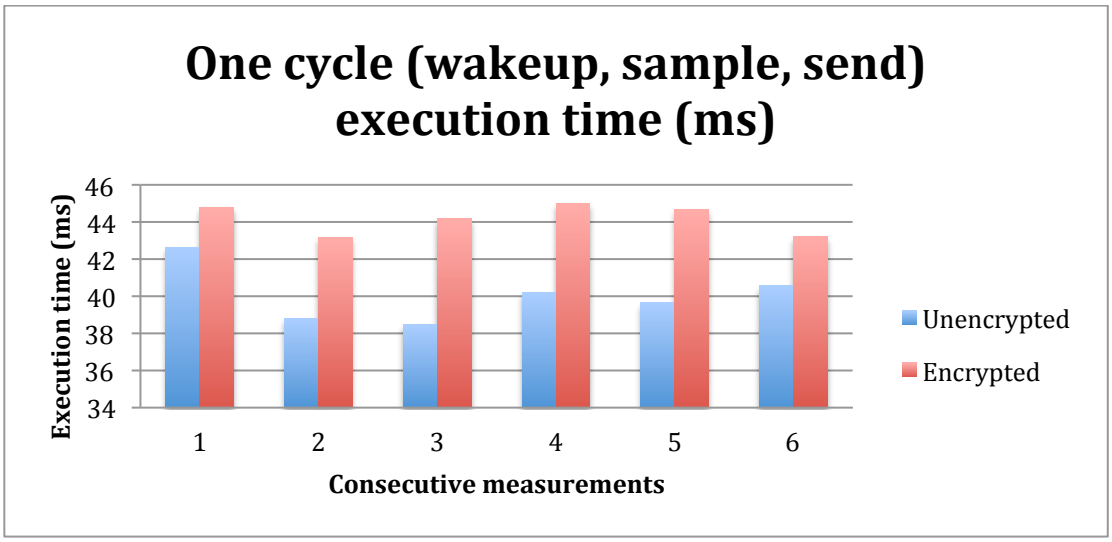

Figure 6: Comparison of one cycle execution time in different modes for 6 consecutive measurements

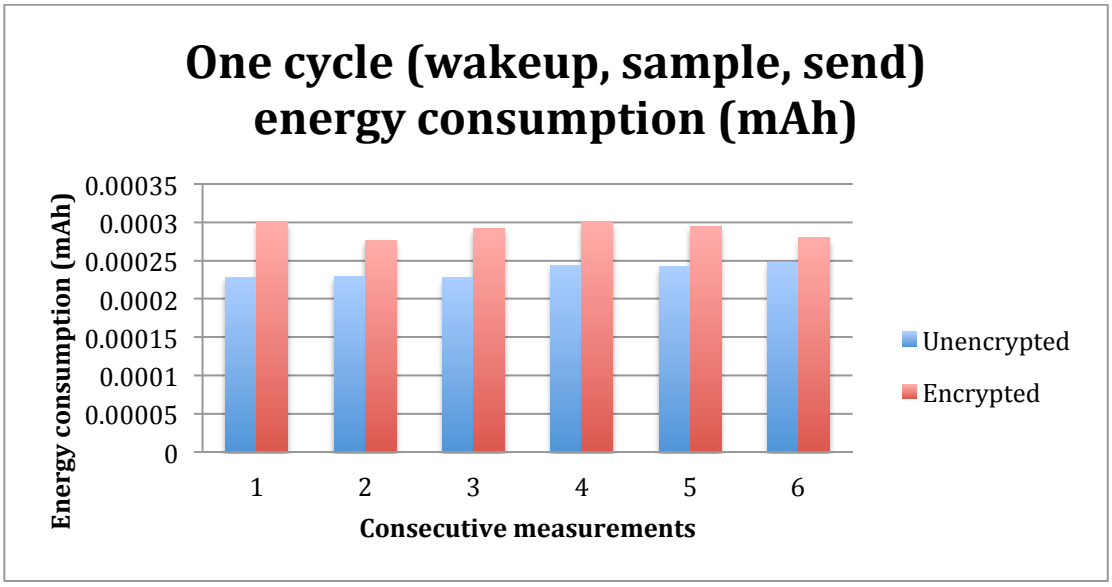

Figure 7: Comparison of one cycle energy consumption in different modes for 6 consecutive measurements

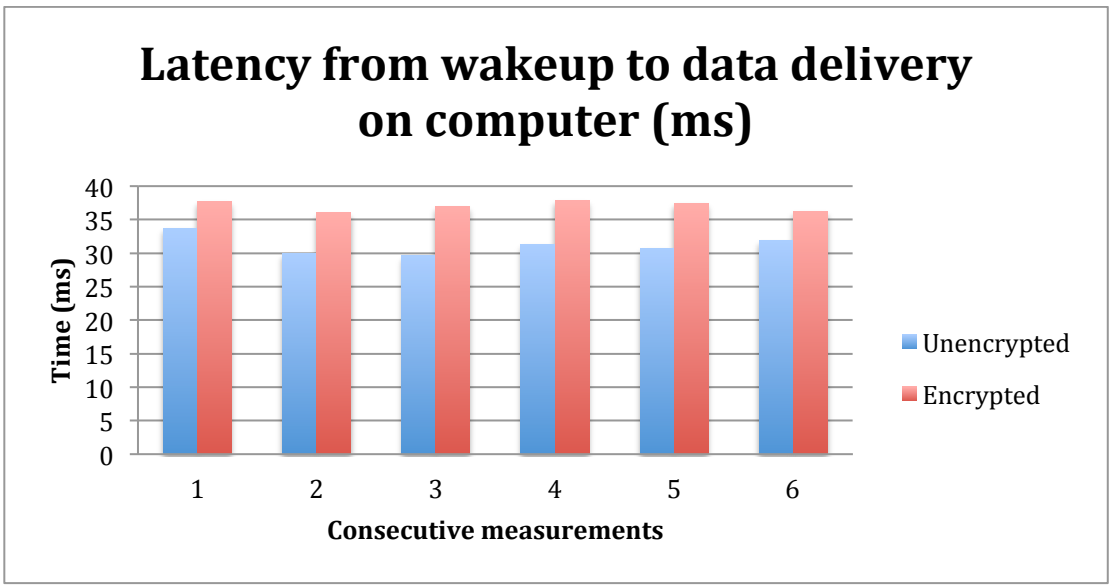

Figure 8: Comparison of latency from wakeup to data delivery on computer in different modes for 6 consecutive measurements

5. Energy consumption

5.1 Energy consumption relation to program execution time

A previous research [10] through performed experiments came to conclusion that energy consumption is directly relevant to execution time. Though this observation can be true with various limitations, it is not believed to be true in this research.

Hypothesis 1. Energy consumption is directly relevant to execution time.
Hypothesis 2. Energy consumption is proportional to used components and utilisation level of every single component.

Results from Experiment 1 show that different data transfer techniques (with and without encryption) result in different execution times. Initially execution time relevance to energy consumption (mAh per second) seem to be questionable – as it is within 3-5% range. However, looking at results graphically show that difference (mAh per second) in results is clearly related to chosen data transfer technique.

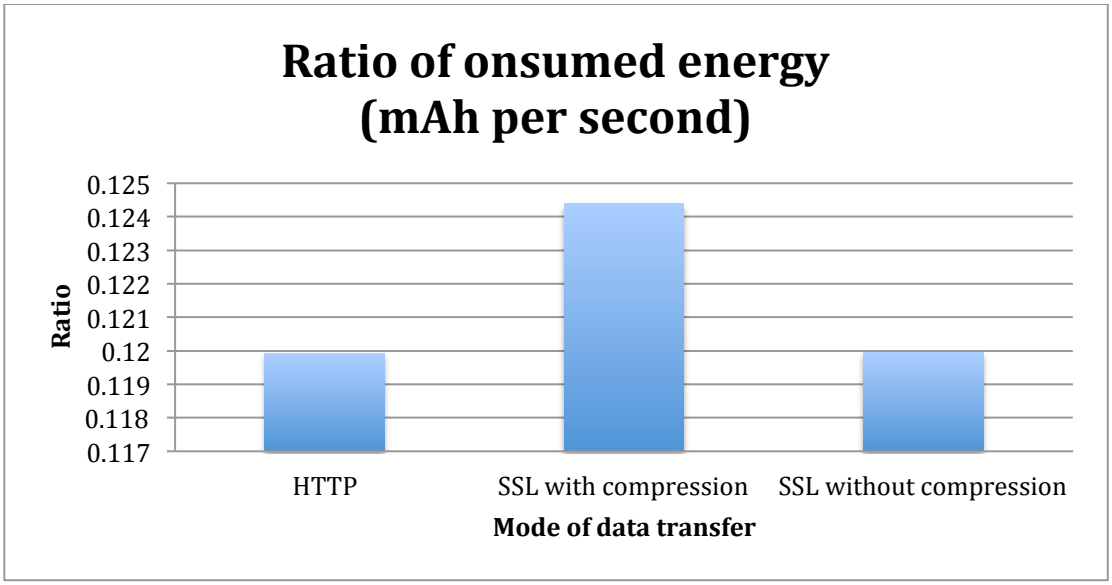

Figure 9: Total mAh consumed during testing process

Experiment 2 was carried out performing data transfer in the same setup as Experiment 1, and this time using wireless network interface for data transfer. Results, however, were even closer matched if expressed as mAh per second.

As proposed Hypothesis 2 states that energy consumption consists of more factors, Experiment 3 was set to measure energy consumption (mAh per second) with different factors taking part in a similar task. Specifically, 200MB of data was being encrypted (output being discarded) using:

a. pseudo file as an input, thus generating CPU workload;
b. normal file (SD card being data storage device) that is being cached in memory as a standard Linux feature;
c. normal file (SD card being data storage device) with instructions to avoid cached file and read it specifically from data storage device.

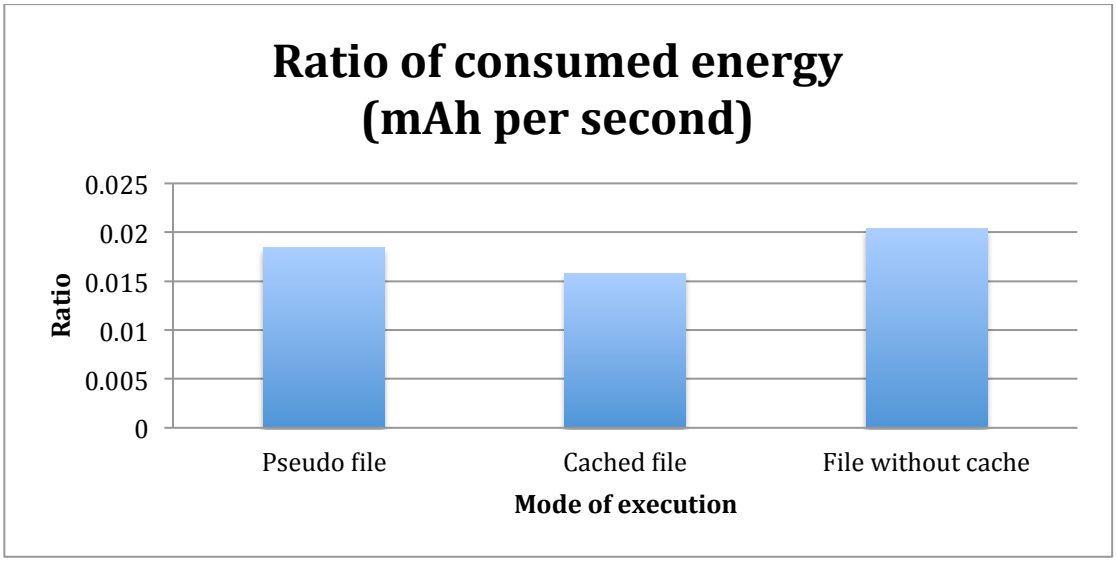

Figure 10: mAh consumed per second during testing process

While absolute values in this test case are relatively closed, if cached file test is taken as a basis for comparison, then pseudo file encryption uses 16.58% more energy per second and non-cached file uses 23.35% more energy per second. Which clearly disapproves Hypothesis 1 and shows that execution time cannot be used as a sole comparison for energy consumption between two processes.

Further during the research, another experiment was carried out for a different part of this research, but it provided valuable results as well.

A temperature sensor was connected to XBee radio transmission module. XBee module was configured to:

- sleep for 28 seconds (consuming minimal power while in sleep mode);
- sample sensor data (read a temperature measurement from directly attached thermo resistor);
- send data to a central radio module which is connected to a central data collection device.

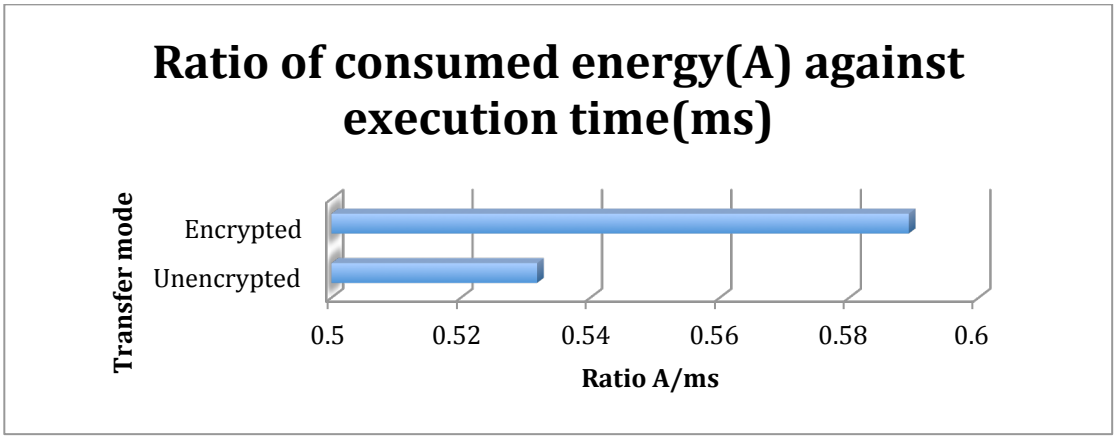

Figure 11: Ratio of consumed energy(A) against execution time(ms)

Measurements obtained during this experiment show that different states of execution cycle (with and without encryption of transmitted data) result in different ratio of consumed energy against execution time – 10% difference for results obtained during the experiment. These results once again disapprove Hypothesis 1 by proving that energy consumption is not directly relevant to execution time in general case.

As Hypothesis 2 is set as an alternative (opposite) to Hypothesis 1, all results disapproving Hypothesis 1, at the same time support Hypothesis 2.

5.2 Energy consumption estimation

Initially it was planned for this research to confirm (not approve) statements set by Hypothesis 2. It was known, prior to the research that measuring energy consumption of different embedded system components is complicated as those components have to be physically separated. Technically, Arduino prototyping platform with it's modular approach would allow separating at least some components.

Though complexity of logical (software) part has been underestimated. Results of Experiment 2 provided unbelievable results to an extent that it was repeated for a few times, caused to implement unplanned design changes to verify results and even repeat Experiment 1.

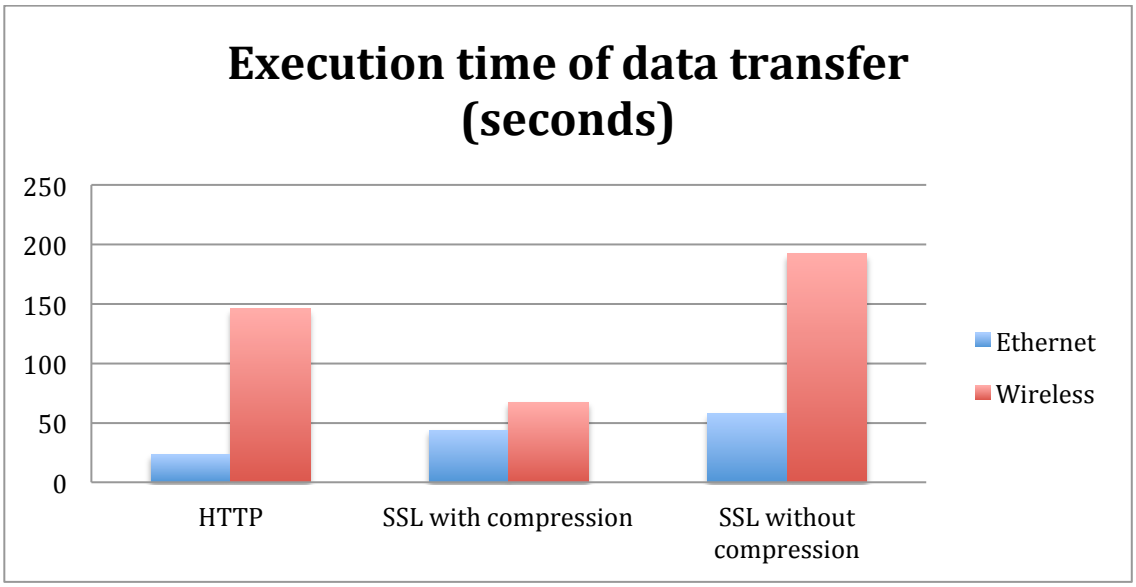

Figure 12: Experiment 1 and 2 execution time comparison

Experiment 1 provided straightforward results for data transmission execution time – the fastest execution is for unencrypted data, as SSL encryption creates data overhead. Compressed encrypted data is faster to transmit due to it's smaller size compared to uncompressed data.

Experiment 2, however, showed significant increase in transfer speed for compressed SSL transmission. As mentioned, wireless access point for this experiment was operating in WPA2 PSK encryption mode. It could potentially be explained by data size of transmission packets – compressed SSL data fitting perfectly into data packet for WPA2 wireless transmission compared to unencrypted data packet possibly being split for transmission and therefore creating significant overhead. Reasons behind this paradox were not explored as they were out of primary scope for this research.

Experiment 4 also showed complexity of calculating energy consumption for data transmission as for tested radio modules it is also organised around data packets with maximum size. Once that size is reached, data packet is split into two or more packets

sent separately. Each packet contains data overhead (e.g. source address, destination address, checksum) and as a result there will always be a point where a 1 byte increase in data size will cause significant increase in consumed time and energy.

Identification of all factors that could affect various software execution processes was not feasible for this research. It would also require extensive testing to ensure that there are no unidentified factors left. Possibility of miscalculation rises proportionally to number of arguments used for estimation. In the end it will be arguable - is better to spend time and effort on estimation or spend money on an equipment for exact measurements.

6. Security implementation in embedded devices

In order to evaluate security implementation, it is first necessary to define security requirements. In general, security requirements are derived from:

- sensitivity of data processed;
- company/user requirements in order to achieve desired security level;
- legal requirements;
- independent compliance audits (PCI DSS, ISO);

As this research is not company or device specific and not country specific, it is left with global standards describing minimal set of security requirements. As PCI DSS is designed specifically for payment system security, it cannot be applied for a wide range of possible uses of embedded devices.

Therefore ISO/IEC 27001:2013 international standard for information security management systems is chosen as a benchmark. It is also approved as national standard in at least several countries across Europe, including Sweden and is also commonly referred to by EU institutions. That means that research results will also contribute to a legal requirement field.

6.1 Statement of ISO/IEC 27001:2013 control applicability

ISO/IEC 27001 standard lists requirements for information security management, not technical requirements for hardware or software. However it is necessary for the equipment, used in organization, to provide technical means for implementation of those requirements.

As this research focuses on embedded devices in general, all organization specific controls are left out of scope. Those controls, which should be implemented in a typical general case, are listed as applicable, meaning that embedded devices should have some form of technical solution for implementation of security management controls.

During security control evaluation, following controls set by ISO/IEC 27001:2013 standard were determined as applicable for this research:

- A.9.2.1 User registration and de-registration;
- A.9.4.1 Information access restriction;
- A.9.4.2 Secure log-on procedures;
- A.9.4.3 Password management system;
- A.9.4.4 Use of privileged utility programs;
- A.10 Cryptography;
- A.12.3 Backup;
- A.12.4 Logging and monitoring;
- A.13.1.1 Network controls;
- A.13.2.3 Electronic messaging;
- A.14.1.2 Securing application services on public networks;
- A.14.1.3 Protecting application services transactions.

Full statement of applicability is available at Annex A.

To summarise, main areas applicable to embedded devices in general are:

- user management and related tasks, e.g. user access control;
- cryptography – from key management to protection of confidentiality and integrity, to data transfer encryption;
- logging;
- software updating;
- support for monitoring.

To identify a starting point of security requirement implementation, applicable areas were connected together with description of their relationship.

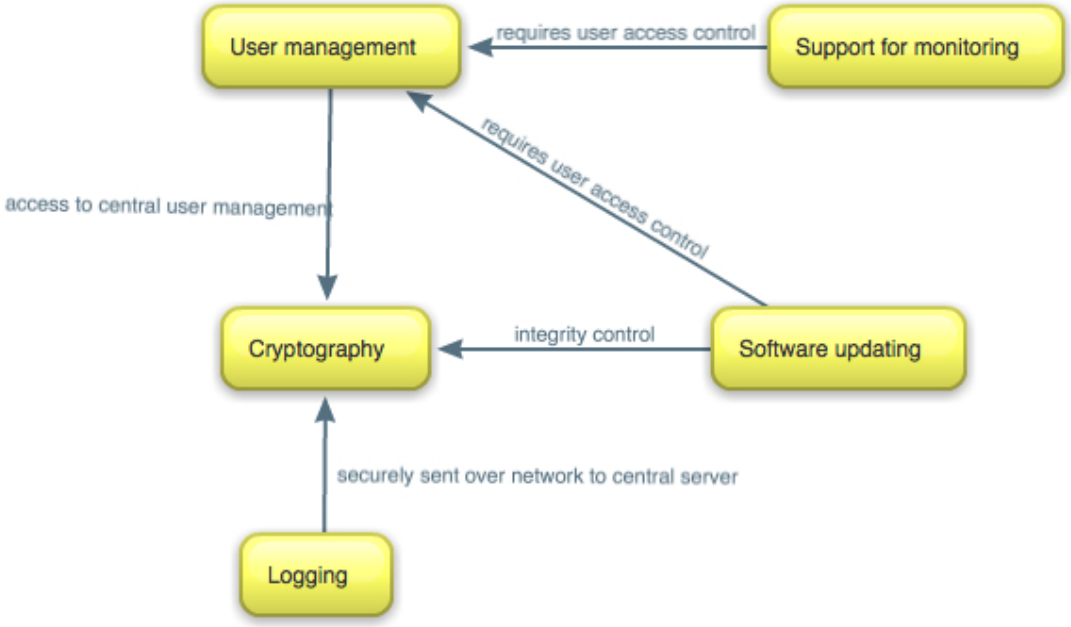

Figure 13: Relationships between identified security requirements

As a result, Cryptography is clearly the first area to explore, as all other functions depend on it in some way.

6.2 HTTPS implementation

As general technique for data transfer encryption across standard IT systems is HTTPS protocol, it was set as first technique to be implemented on sample embedded systems.

Experiment 1 was initially carried out on Raspberry Pi platform hardware which is actually a low performance computer typically running Linux OS. Although it is frequently used to perform a single task and thus effectively being used as an embedded device.

According to research objectives, next step for Experiment 1 would be to repeat tests on Arduino embedded device prototyping platform. However, first obstacle is lack of HTTPS data transmission library availability.

If we take away hardware limitations, logical structure of HTTPS requests also possess implementation challenges. Data encryption for HTTPS uses server

certificates and rely on chain of trust by verifying if chain of certificates that starts with a trusted Root Certification Authority (Root CA -> Intermediate CA -> Webserver certificate).

In a typical computer system environment, list of trusted Root CA's is being maintained by OS manufacturers and updated via typical OS security updates. A number of trusted Root CA's in Debian based linux, iOS 7 [21], Windows 8 is ranging from 160 to 310. Size of a single certificate stored on disk is slightly above 4K (having a public key of 4096bits). In order to store 200 certificates, device would need to have 800KB of memory.

Arduino boards, which were chosen for this research, typically have just 32KB of Flash memory and 2KB of SRAM, with only one board having 512KB Flash memory and 96KB of SRAM available. That is also the only Arduino board having more than 8KB of SRAM, that would allow loading at least a single certificate (with potential future requirements in mind) in memory. Therefore, without an additional storage device or large size memory module, it is not possible to create full SSL/TLS implementation.

6.3 AES implementation

Initially it was planned for the research to use commercially or community approved algorithm implementations, e.g. OpenSSL library. However, it was not available for use with Arduino IDE and possibility of its use due to memory limits is also questionable.

Search for readily available AES encryption implementation for Arduino platform revealed three resources:

- AES-library [22] (shared on Arduino forum post [23]);
- AESlib [24];
- AES256 [25].

A closer look at AES-library implementation shows that it has a serious security flaw – a static initialization vector (IV). NIST requirements [26] state that IV "**must** be

unpredictable". Having a static IV value contradicts this requirement. Therefore this library should **not** be used.

Using sample codes provided with each library, their encryption speed was tested using Arduino Uno board (same CPU speed as Ardupilot board).

Table 7: Measured encryption speeds for different AES libraries on Arduino platform

Library	128-bit encryption time	112.5 -Kbit encryption time (calculated)	256-bit encryption time	112.5-Kbit encryption time (calculated)
AES-library	0.00580 s	5.22 s	0.00820 s	3.69 s
AESlib	0.00288 s	2.59 s	0.00400 s	1.80 s
AES256	-	-	0.04754 s	21.39 s

Theoretically it means that data could be encrypted at 44.44Kbit/sec rate which is less than full speed serial port (115200bit or 112.5Kbit). As this research was set to use practical approach instead of purely theoretical - Ardupilot is used as embedded system for AES implementation evaluation, specifically air to ground communication. Currently this communication is sent in clear text, therefore it is susceptible to data integrity violation.

Ardupilot software sends radio messages to ground control at 50Hz rate which is every 0.02seconds and message transmission is expected to finish within 720 to 950 microseconds for general messages and within 150 microseconds for heartbeat messages.

Heartbeat message size is 9 bytes=72bit which is one 128bit encryption loop. It would mean that time for sending heartbeat message would increase from 150 microseconds to 3030 microseconds (20.2 times longer). It might be argued that a factor of 20.2 might be incorrect for longer messages that will fit into one encryption loop – that would make impact factor smaller. On the other hand, it could be also larger for packets with total size slightly above a factor of 128bit (e.g. 136bit, 264bit, etc.).

In order to avoid potential dis-belief in the results and their conclusions, further analysis will use impact factor calculated per data bit:

- 2880 microseconds for 128bit block encryption, equals 22.5 microseconds per bit;
- 72bit encryption will require additional 1620 microseconds to initially reserved 150 microseconds for message transmission;
- total time for heartbeat message transmission would be 1770 microseconds instead of 150 microseconds – a factor of 11.8.

Other communication tasks have reserved processing times of 720 and 950 microseconds. If we increase those times proportionally (by a factor of 11.8) that results in 8'496 and 11'210 microseconds accordingly. But 50Hz loop has just 20'000us of maximum available time, which must be shared with other tasks as well:

- 800us are reserved for 100Hz loop;
- 619us are reserved (calculated proportionally with increased value of heartbeat message processing time) for 10Hz, 33Hz and 1Hz processing loops;
- and 3340us are reserved for other tasks in 50Hz loop.

Totalling those values means that there could be 15'241 microseconds of possibly "spare" time and it is estimated that data encryption would require at least 19'706 microseconds.

Overall it means that encrypting data communication on main CPU for Ardupilot cannot be achieved with current design. More importantly, it has been demonstrated with just a lack of CPU power for the actual encryption without analysing related implementation requirements:

- generation of non-predictable random values for IV used for data encryption;
- impact of disturbed air-to-air communication – defective and lost data packets.

Defective or lost data packets have a considerable effect on data communication reliability. One defective 128-bit block for AES encryption in CBC mode would mean 256 bits of lost (corrupted) data. If a defective block happens to be a final block for transmitted message, it results in 2 corrupted data messages. In CTR encryption mode for AES, defective block would not affect a following data block. However, it is not supported by libraries currently available for Arduino platform and it has stricter requirements for IV generation compared to CBC mode.

Though it definitely does not mean that communication can not be encrypted. Generally there are two methods (hardware with software support) for wireless communication with Ardupilot – 3DR radio module and XBee radio module. 3DR module is designed specially for Ardupilot communication and therefore claimed to better suited for this purpose [27] and XBee is a general purpose radio communication module. XBee usage is evaluated in next chaper.

6.4 Commercial devices

According to XBee specification [28] data rates up to 1Mbps are achiveable for some device models and 128-bit AES encryption is supported. Their own testing [29] has shown that enabling encryption does slow down encryption rate by 26 to 50 percent.

Initially it was set for this research to create a proof-of-concept SCADA system:

- analyse sensor data;
- perform an action based on this data;
- transmit data to a central server.

It was demonstrated in previous chapters that encrypted data transmission is near impossible to implement using Arduino platform itself. As impact on system performance and energy consumption will consist of added energy consumption characteristics and performance limitations of XBee device it self, the initial plan is adjusted to connect sensor directly to XBee device and instruct XBee to perform sensor measurement and transmit the measurement to a 2nd XBee module connected to a central data collector device.

This is possible due to ZigBee protocol on which XBee operation is based. In short, ZigBee protocol is designed for low-cost, low-power wireless sensor and control networks[30]. A typical example of ZigBee network is building security system using wireless motion detectors and wireless smoke detectors. Thus it will demonstrate another aspect of embedded device usage.

6.4.1 XBee encryption setup

A distinct feature of ZigBee protocol is a “sleep” mode – ability for device to sleep for a period of time while consuming minimal amount of power and processing all incoming and outgoing messages on a cycle basis. This feature allows cutting power

usage to minimum – to levels that can be powered for a year or more using just battery power. Therefore it is a very useful for remote sensors as their placement is not directed by ability to hardwire them and they can be relocated, if necessary (within limits of overall network coverage area).

Factors that impact device performance with added encryption:

- data transfer throughput rate;
- latency in data delivery;
- added energy consumption.

As discussed previously, data transfer throughput rate requirements will be unique to every device purpose. They are well within requirements (also with 50% decrease in encrypted mode) for a remote sensor application tested during this research. Technical specifications for certain device model data transmission rates would also satisfy requirements for UAV air-to-air communication. However, it's real usage in such scenario was not tested as 3DR radio module and XBee radio module are not directly comparable due to different radio frequencies being used. Which also affects communication range, an important factor for UAV.

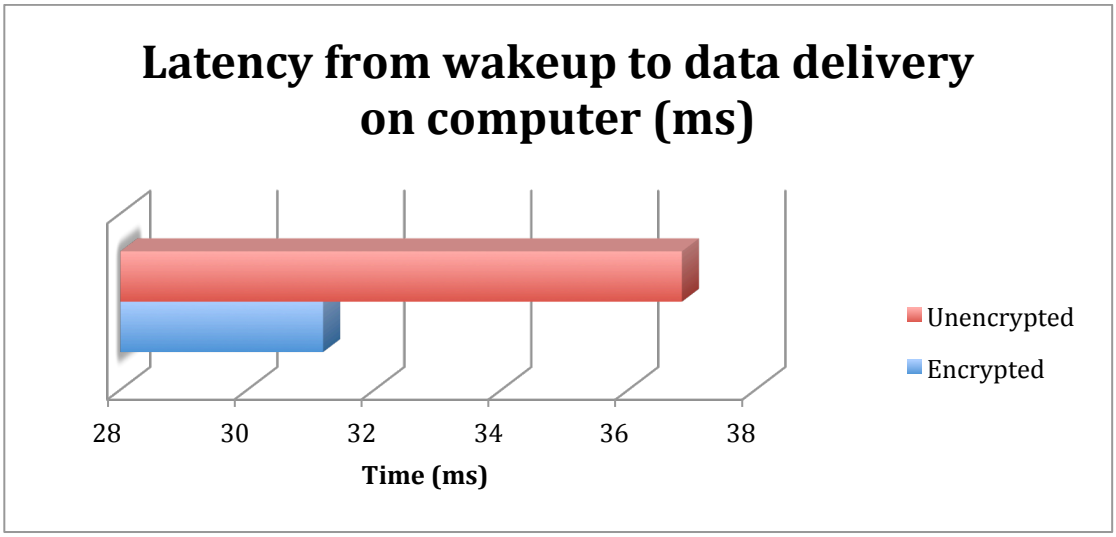

Figure 14: Latency from wakeup to data delivery on computer (ms)

Results from Experiment 4 show that latency from wake up of transmitting sensor node to start of data delivery on receiving computer was measured to be on average

31.93ms in unencrypted mode and 36.84ms for encrypted mode. Additional 5ms of latency are considered to be acceptable for a remote sensor data delivery, but acceptance could vary on different usage cases. For a UAV usage 5ms additional latency should be acceptable for both telemetry data delivery and remote control. US military UAV's are reported [31] to have maximum speed of 135mph (217.26 km/h), which results in 30cm in 5ms. Whilst a military grade encryption system could have smaller latency for data encryption, 30cm are believed to be a reasonable distance that can be either tolerated or calculated (compensated) for.

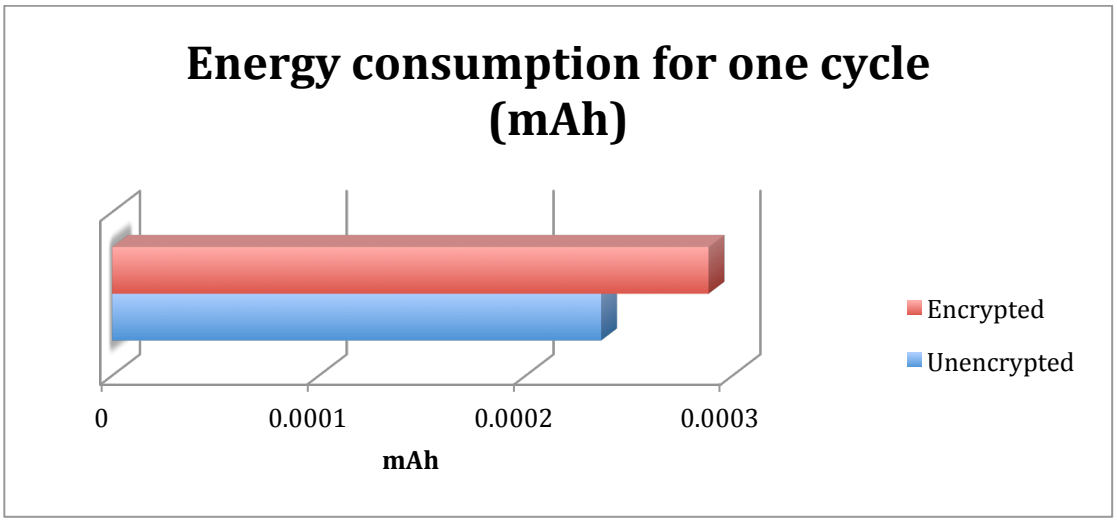

Figure 15: Energy consumption for one encryption cycle (mAh)

Energy consumption absolute values for one execution cycle (sample sensor data, send it to central device) are small and unrepresentative. A button size (20mm diameter) battery or two AA type batteries with 1Ah capacity would be able power sensor reading and data transmission for more than 6 years.

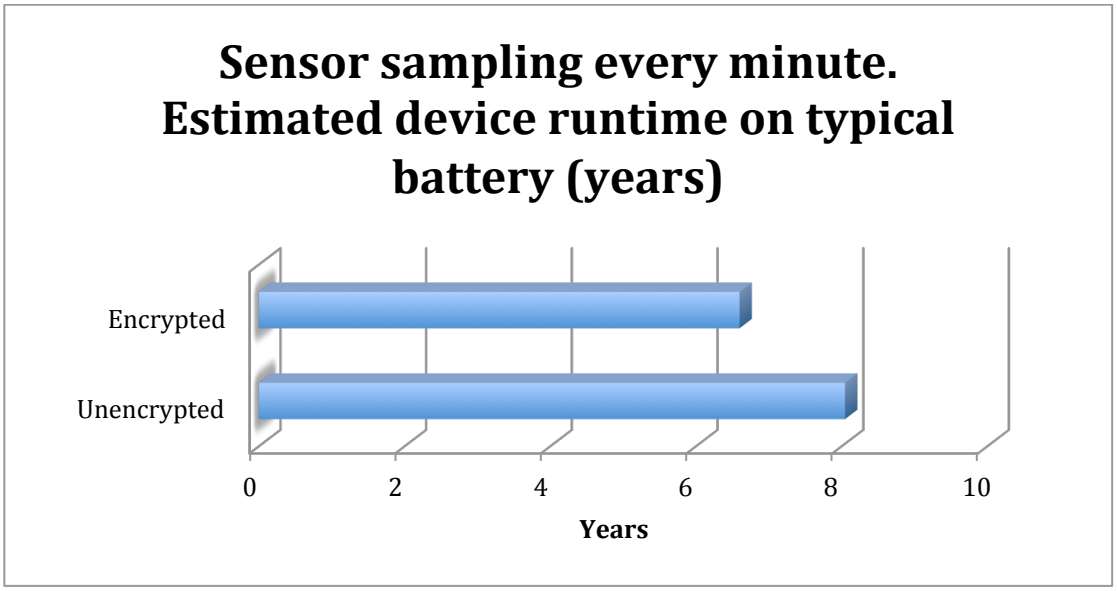

Figure 16: Estimated device runtime on typical battery with sensor sampling every minute

Estimated runtime is calculated without taking into account battery characteristics of varying power level over time. An 18% impact (measured for this case) on energy consumption becomes visible but well within acceptable range for a remote sensor. In different circumstances (e.g. more frequent sampling rate) absolute times will vary, however, impact level would remain at an acceptable level – e.g. 100 days (slightly more than 3 months) versus 82 days (slightly less than 3 months).

7. Discussion

Research results supported main views by previous researches that embedded devices possess design limitations and processing gap. Even with technology of today. This research has showed and explained why exactly those limitations prevent adequate security implementation. It has also demonstrated that in order to implement security, an existing design will most likely need additional hardware just as it was observed 10 years ago in commercial device designs[8].

During testing of Hypothesis 1 – a conclusion from previous research – "Energy consumption is directly relevant to execution time" [10] was proved wrong in general case. It might be true, if specific conditions are met, but it is false in general case. The same results support an alternative conclusion – Hypothesis 2.

Initially it was planned that during design limitation research, various security requirements will be implemented and then, knowledge gained from RQ1 would be used to evaluate impact of implemented security requirements. And it turned out that security implementation in existing embedded devices is cumbersome, that limited battery gap evaluation regarding security for this research to results of only one experiment.

It was also demonstrated that in general case there would be too many factors to account for in order to estimate security impact on energy consumption based on systematic calculations. Although it goes well along chosen research method and disapproves Hypothesis 3, it also means that creating a framework for energy consumption estimation is not unfeasible.

Expectations from this research were different at the beginning. It was thought to be possible to estimate added workload and increase in data size due to necessary changes for security requirement implementation. However, early results from Experiment 2 proved these assumptions to be wrong. If not wrong, the initial assumptions were at least incomplete – being too focused on embedded device hardware environment.

Results from Experiment 2 showed significantly faster data transmission rate for encrypted (SSL) data with compression. First results were discarded as invalid and measurements being repeated. Over period of time with adjusted testing methods and received data integrity checks, results were confirmed as valid in every case. One of the measured parameters – data packets transmitted on network interface – showed that this method used least amount of data packets by a magnitude of 5 times.

That can only be explained by data encapsulation process, which is done by 3rd party software (network drivers) and can depend on vast amount of factors. Software impact was the part that was overlooked during initial assumptions and played main role in disapproving Hypothesis 3.

Results from Hypothesis 4 testing show that it is possible to practically implement some security features without impact on system performance (throughput, latency, battery life) that would render the system useless. Most likely, though, it will require buying a separate hardware module to implement particular security functions.

As a result, potential for future work in this field is seen on designing necessary hardware and support that would allow rapid embedded device prototype development in future to possess necessary security attributes as well. Another field of future research could be independent evaluation of available hardware (commercial or open-source) security implementations.

8. Conclusions

In the wake of Internet of Things, that consists of embedded devices, it is important to implement adequate security in those devices in order to protect people privacy and safety. As internet keeps expanding, it often brings online systems that have previously been protected by isolation – SCADA systems for power plants, oil platforms in open sea, marine control and many more.

A knowledge gap of added security impact on embedded device performance and of security implementation complexity level in embedded devices was found from literature review. Therefore this research was set to test through practical experiments following hypothesis:

- embedded device energy consumption is proportional to program execution time;
- embedded device energy consumption is proportional to used components and utilisation level of every single component;
- consumed energy can be estimated based on systematic calculations;
- embedded device characteristics allow security requirement implementation at the level required by ISO/IEC 27001:2013 standard.

It was demonstrated that even with today's technology security implementation can be complicated unless it is possible to find a readily available hardware module that performs necessary functions at an expected performance rate. It was also shown that with such hardware availability, functions can be secured. Generally security impact (tested on a proof-of-concept setup used for experiment) was found to be at an acceptable level performance wise – throughput, latency, additional energy consumption. Some aspects were also estimated to be theoretically acceptable for applications with high performance requirements like UAV's flying at high speed.

Final impact though has to be tested for each device (product) specific requirements. It was initially expected fill a knowledge gap and provide a framework for security impact estimation, however, energy consumption was found to be affected by too many factors. Identification of those factors was not feasible for this research and

their use would be arguable as well – possibility of miscalculation rises proportionally to number of arguments used for estimation. In the end it will still be an estimation and it becomes questionable if it is better to spend time and effort on estimation or spend money on an equipment for exact measurements.

Opposed to expectations at the beginning of this research that battery gap would be main deterrent for security implementation in embedded devices, it turned out that without hardware re-design, security implementation can even be impossible with available technology today. Hardware redesign, though, is time consuming and expensive process, therefore a path to secure embedded devices is seen by enabling easy security implementation during prototyping phase.

On a final note to answer a side question – is lack of security in embedded devices a simple negligence. While implementation certainly is not easy and straightforward, lack of it still can not be justified. And it also leaves us with a new question – what kind of security level we should realistically expect from SCADA systems that control critical infrastructure and have been created with technology that was available at least a decade ago.

References

[1] Heath, S. (2003). *Embedded systems design*. Oxford: Newnes.

[2] Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer networks*, *54*(15), 2787-2805.

[3] Greene, T. (2009, December 17). *Unencrypted drone video intercepted by militants*. Retrieved January 9, 2014, from http://www.networkworld.com/news/2009/121709-drone-intercept-encryption.html

[4] Zetter, K. (2010, July 19). *SCADA System's Hard-Coded Password Circulated Online for Years*. Retrieved January 9, 2014, from http://www.wired.com/threatlevel/2010/07/siemens-scada/

[5] Gao, J., Liu, J., Rajan, B., Nori, R., Fu, B., Xiao, Y., ... & Philip Chen, C. L. (2014). SCADA communication and security issues. *Security and Communication Networks*, *7*(1), 175-194.

[6] International Organization for Standardization, & International Electrotechnical Commission (2013). *Information technology - Security techniques - Information security management systems - Requirements* (2013 ed.). Geneva: ISO.

[7] Potlapally, N. R., Ravi, S., Raghunathan, A., & Jha, N. K. (2003, August). Analyzing the energy consumption of security protocols. In *Proceedings of the 2003 international symposium on Low power electronics and design* (pp. 30-35). ACM.

[8] Ravi, S., Raghunathan, A., Kocher, P., & Hattangady, S. (2004). Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, *3*(3), 461-491.

[9] Ruangchaijatupon, N., & Krishnamurthy, P. (2001, September). Encryption and power consumption in wireless LANs. In *The Third IEEE Workshop on Wireless LANs-September* (pp. 27-28).

[10] Hager, C. T., Midkiff, S. F., Park, J. M., & Martin, T. L. (2005, March). Performance and energy efficiency of block ciphers in personal digital assistants. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on* (pp. 127-136). IEEE.

[11] Dermanilian, H. M., Saab, F., Elhajj, I. H., Kayssi, A., & Chehab, A. (2013, August). Energy-Efficient Security for Voice over IP. In *International Journal of Network Security*, Vol.17, No.1, PP.11-26, Jan. 2014

[12] Bickford, J., Lagar-Cavilla, H. A., Varshavsky, A., Ganapathy, V., & Iftode, L. (2011, June). Security versus energy tradeoffs in host-based mobile malware detection. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (pp. 225-238). ACM.

[13] Mlynek, P., Misurec, J., Koutny, M., & Raso, O. (2013, October). Design of secure communication in network with limited resources. In *Innovative Smart Grid Technologies Europe (ISGT EUROPE), 2013 4th IEEE/PES* (pp. 1-5). IEEE.

[14] Al Fardan, N. J., & Paterson, K. G. (2013, May). Lucky thirteen: Breaking the TLS and DTLS record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on* (pp. 526-540). IEEE.

[15] Kocher, P., Lee, R., McGraw, G., Raghunathan, A., & Moderator-Ravi, S. (2004, June). Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference* (pp. 753-760). ACM.

[16] Koopman, P. (2004). Embedded system security. In *Computer, 37(7), 95-97.*

[17] Sedghi, A. R., & Kaghazgaran, M. R. (2013). Data Security via Public-Key Cryptography in Wireless Sensor Network. In *International Journal on Cybernetics & Informatics (IJCI)* Vol.2, No.3, June2013.

[18] Accessory Development Kit 2012 Guide. (n.d.). *Android Developers*. Retrieved April 25, 2014, from http://developer.android.com/tools/adk/adk2.html

[19] Dodig-Crnkovic, G. (2002, April). Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia* (pp. 126-130).

[20] Popper, K. (2005). *The logic of scientific discovery*. Routledge.

[21] iOS: List of available trusted root certificates. (2013, September 18). *iOS: List of available trusted root certificates*. Retrieved April 19, 2014, from http://support.apple.com/kb/HT5012?viewlocale=en_US&locale=en_US

[22] Tillotson, M. (2012, January 23). *AES library*. Retrieved May 4, 2014, from http://utter.chaos.org.uk:/~markt/AES-library.zip

[23] Tillotson, M. (2012, January 25). *new AES library*. Retrieved May 4, 2014, from http://forum.arduino.cc/index.php?topic=88890.0

[24] Landman, D. (2013, October 3). *Arduino AESLib*. Retrieved May 4, 2014, from https://github.com/DavyLandman/AESLib

[25] Marle, C. (2014, January 29). *ArduinoAES256*. Retrieved May 4, 2014, from https://github.com/qistoph/ArduinoAES256

[26] Dworkin, M. (2001). *NIST Special Publication 800-38: Recommendation for Block Cipher Modes of Operation* (800-38). Retrieved May 11, 2014, from US National Institute of Standards and Technology website: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[27] *Using wireless data modules for telemetry and in-flight commands*. (2013, February 19). Retrieved May 25, 2014, from https://code.google.com/p/ardupilot-mega/wiki/Telem

[28] Digi International Inc. (n.d.). *Product datasheet: XBee® & XBee-PRO® ZB*. Retrieved May 25, 2014, from http://ftp1.digi.com/support/documentation/90000976_S.pdf

[29] Digi International Inc. (2014, March 5). *Product Manual: XBee / XBee-PRO ZB RF Modules*. Retrieved May 25, 2014, from http://ftp1.digi.com/support/documentation/90000976_S.pdf

[30] ZigBee Aliance (n.d.). *ZigBee Technology*. Retrieved May 27, 2014, from http://www.zigbee.org/About/AboutTechnology/ZigBeeTechnology.aspx

[31] Lister, T. (2014, November 9). *Drone incident over the Gulf: A sign of the times*. Retrieved May 27, 2014, from http://security.blogs.cnn.com/2012/11/09/drone-incident-over-the-gulf-a-sign-of-the-times/

Annex A. Statement of ISO27001:2013 Controls Applicability

Control	Applicability
A.5 Information security policies	Not Applicable Organisation specific requirements
A.6 Organization of information security	Not Applicable Organization specific requirements
A.7 Human resource security	Not Applicable Organization specific requirements
A.8 Asset management	Not Applicable Organization specific requirements
A.9 Access control	Partially Applicable (see subsections)
A.9.1 Business requirements for access control	Not Applicable
A.9.2 User access management	Partially Applicable (see subsections)
A.9.2.1 User registration and de-registration	**Applicable**
A.9.2.2 User access provisioning	Not Applicable
A.9.2.3 Management of privileged access rights	Not Applicable
A.9.2.4 Management of secret authentication information of users	Not Applicable
A.9.2.5 Review of user access rights	Not Applicable
A.9.2.6 Removal or adjustment of access rights	Not Applicable
A.9.3 User responsibilities	Not Applicable
A.9.4 System and application access control	Partially Applicable

	(see subsections)
A.9.4.1 Information access restriction	**Applicable**
A.9.4.2 Secure log-on procedures	**Applicable**
A.9.4.3 Password management system	**Applicable**
A.9.4.4 Use of privileged utility programs	**Applicable**
A.9.4.5 Access control to program source code	Not Applicable
A.10 Cryptography	**Applicable**
A.11 Physical and environmental security	Not Applicable Out of scope for this research, specific to device function and location, and specific to organization requirements
A.12 Operations security	Partially Applicable (see subsections)
A.12.1 Operational procedures and responsibilities	Not Applicable
A.12.2 Protection from malware	Not Applicable
A.12.3 Backup	**Applicable**
A.12.4 Logging and monitoring	**Applicable**
A.12.5 Control of operational software	Not Applicable *Procedural controls*
A.12.6 Technical vulnerability management	Not Applicable *Procedural controls*
A.12.7 Information systems audit considerations	Not Applicable
A.13 Communications security	Partially Applicable (see subsections)
A.13.1 Network security management	Partially Applicable (see subsections)
A.13.1.1 Network controls	**Applicable**

A.13.1.2 Security of network services	Not Applicable
A.13.1.3 Segregation in network	Not Applicable
A.13.2 Information transfer	Partially Applicable (see subsections)
A.13.2.1 Information transfer policies and procedures	Not Applicable
A.13.2.2 Agreements on information transfer	Not Applicable
A.13.2.3 Electronic messaging	**Applicable**
A.13.2.4 Confidentiality or non-disclosure agreements	Not Applicable
A.14 System acquisition, development and maintenance	Partially Applicable (see subsections)
A.14.1 Security requirements of information systems	Partially Applicable (see subsections)
A.14.1.1 Information security requirements analysis and specification	Not Applicable *Procedural controls*
A.14.1.2 Securing application services on public networks	**Applicable**
A.14.1.3 Protecting application services transactions	**Applicable**
A.14.2 Security in development and support processes	Not Applicable *Procedural controls*
A.14.3 Test data	Not Applicable
A.15 Supplier relationships	Not Applicable
A.16 Information security incident management	Not Applicable *Procedural controls*
A.17 Information security aspects of business continuity management	Not Applicable *Procedural controls*
A.18 Compliance	Not Applicable *Country specific or procedural controls*