WPA password cracking
Parallel Processing on the Cell BE

Master thesis, AAU,

Applied Signal Processing and Implementation

Spring 2009

Group 1045

Martin Daniel

Aalborg University
Institute for Electronic Systems
Fredrik Bajers Vej 7
DK-9100 Aalborg
Phone: (+45) 9635 8080
http://es.aau.dk

Title:

WPA password cracking

Parallel Processing on the Cell BE

Theme:

Applied Signal Processing and Implementation

Project period:

Spring 2009

Project group:

1045

Participant:

Martin Daniel

Supervisors:

Yannick Le Moullec (AAU)

Jes Toft Kristensen

Ole Mikkelsen

(Rohde & Schvartz)

Copies: 5

Number of pages: 60

Attachment: 1 CD-ROM

Completed 12-07-2009

Abstract:

This project deals with the challenges of implementing WPA password cracker on an Cell Broadband Engine processor in PlayStation 3. The WPA security standards were investigated to establish their potential weak points. As a result of investigation was detected that WPA-PSK authentication offers only one known possible weak point how to attack the WPA security during authentication. Further exploration of WPA-PSK authentication led to establish block diagram of designed application. The open source wpa_supplicant code was used to extract the parts of code related to the WPA-PSK authentication. These parts of code were used to design working code on PC. Profiling of code determined possibility of parallel processing and detected the most consuming parts of code. After investigation of the PlayStaion 3 platform with the Cell BE processor, the password cracker was implemented on the PPU with minor changes of the code related to Big-Endian data storing. Then the code was optimised for SPU. The performance of password cracking tool was benchmarked on the PC, PPU, SPU and on the SPU after optimisation. The performance of the cluster of the six SPUs provides almost 30 times more tempts per second than implementation just on the PPU. After second iteration of implementation involving SHA1-SSE2 source code, the cluster of six SPU reached 701 tested passphrases per second.

Preface

This report is the documentation of the last semester Applied Signal Processing and Implementation project at Department of Electronic Systems at Aalborg University, Denmark.

The report is composed of two parts: the main report and an enclosed CD-rom. The CD-rom contains the implemented code, material used for testing and a digital copy of this report.

The report consists of three layers: Chapters, sections and subsections. The chapters and sections are stated in the main table of contents.

Figures, tables and listings are numbered by two numbers separated by a dot. The first number indicates the chapter number and the second number denotes the number of figure, table or listing in the chapter. Equations are numbered in brackets and with the same convention as the figures and tables, e.g. (2.1). Listing environment contains code examples or terminal commands with each line numbered for easier referencing.

The following notation is used throughout the report:

Functions are written in *bold* and variables are written in **italic**.

The words frame and image are used interchangeably.

Citations are written in square brackets with a number, e.g. [3]. The citations are listed in the bibliography on page 60.

Aalborg University, July 13^{th} 2009

_____

Martin Daniel

Contents

Nomenclature

AP Acces Point, page 12

CCMP Counter Mode with Cipher Block Chaining Message Authentication Code Protocol, page 14

Cell BE Cell Broadband Engine, page 3

CUDA Compute Unified Device Architecture, page 19

DoS Denial-of-Service, page 14

EIB Element Interconnect Bus, page 26

ELF Executable and Linking Format, page 28

FPGA Field-Programmable Gate Array, page 19

GPU Graphic Processing Unit, page 19

GTK Group Transient Key, page 16

HW Hardware, page 14

IV Initialisation Vector, page 12

KCK Key Confirmation Key, page 16

LAN Local Area Network, page 1

LSB Least Important Bit, page 41

MAC Media Acces Control, page 12

MFC Memory Flow Controller, page 26

VMX Vector Multimedia Instructions, page 25

WEP Wired Equivalent Privacy, page 2

WiFi Wireless LAN, page 1

WLAN Wireless LAN, page 12

WPA WiFi Protected Acces, page 2

Chapter 1

Introduction

In today's world, wireless communications are more and more common. The whole world of wireless communications, as we know it today, started in 1895, when Guglielmo Marconi transmitted the Morse code for letter "S" (three-dots) over a distance of 3 kms by electromagnetic waves. From this time, wireless communications have grown up into a key element of modern society [13]. In our modern society, wireless communications are hidden in common technologies like radio and TV broadcasting (terrestrial or satellite), remote control, Bluetooth, cell phone, connection to the Internet and many others.
Interesting is a wireless connection to the computer network, because it is a massively used technology. WiFi - Wireless LAN (Local Area Network) is the main technology for wireless connection to computer networks. Portable devices such as laptops, mobile phones and handhelds commonly have the possibility to connect to the Internet via WiFi. WiFi is also a trademark of the WiFi Alliance (logo on figure 2.1), which takes care about certifications of products, security protocols, etc. [2].
Because data of all kinds flow over wireless communications, it is natural that users want to secure at least the most important information. Important information can be industrial secrets, military secrets, political secrets, intelligence agency information, or it can be more personal like bank account details, e-mails, addresses, cell phone numbers or whatever else. In front of those who want to protect their information, there are a lot of persons who want to break this security and steal this information.
Their motivation can be from the need to know what someone told somebody else; industrial, political and military secrets through financial fraud up to simply just wants to enjoy free Internet access. The motivation varies as well as the resoluteness to break in. Probably the highest resoluteness to break in is in army, where it is a question of strate-

gical advantage or a question of survival. The same level of resoluteness is in politics and intelligence. With them comes the ability to provide Hi-Tech equipment (like supercomputers) and the best human resources. Also in the industry it is possible to find high level of resoluteness, especially rich companies could pay a lot for information, without caring about how it was obtained. On the other side, someone, who just want use free Internet access, typically has a much lower resoluteness. But the quantity of hackers like this one is the highest. When an ordinary owner of the WiFi connection is attacked, it is higly probable that it is by hackers like this one. And to avoid attack is easy - it is enough to have a better security than neighbours, because it is most likely the weakest network that will be attacked by such hackers. But this is not true in the case of a directed attack to a specific network.
There exist several standards for secure communication protocols in WiFi like WEP (Wired Equivalent Privacy), WPA (WiFi Protected Access) and WPA2 [25]. But development is so fast that codes and standards become obsolete very rapidly and offers weak points that the hackers can exploit. Therefore it is important to put efforts for protecting and proof the security of the networks - including WiFi. Nowadays there already exist tools and programs for cracking codes in the older standards. And security teams, cryptography experts, and hackers continuously try to find the weaknesses of currently used security standards: the two first mentioned ones for proofing the security, the second for their own interest. Therefore the whole process of security is a circle, which typically consists of the next steps:

- Developing a new security standard.
- Testing and then using the new standard.
- Looking for weakneses of the standard by experts and hackers, while simultaneously developing newer standard.
- Using the current security standard until it becomes vulnerable and then start using a newer one.
- Then the circle is repeated.

A proof of the importance of security testing could be, that companies like Lockheed Martin spends a lot in it. Weapons company Lockheed Martin recently opened Wireless Cyber Security Lab. It is one of the few facilities capable of testing systems in classified environment [21]. One of the projects is cracking WPA-encrypted networks with 8

clustered Sony PlayStations. The PS3 (Play Station 3) hardware is used to build supercomputing environments [14]. The advantage of using the PS3 is that no-one need to buy equipment from the black market, one can buy it in a toyshop and build a supercomputer at home [29]. And it is a security issue, mainly for the army. "*You break the passphrase, you break the network, and that has disastrous implications for some of the military-based projects that Lockheed is working on. According to Crawford, "The military has a vision of having an IP address for every soldier and weapon," Morrison says. "They're not going to be trailing wires around on the battlefield, but that can lead to some vulnerabilities.*" [14]
Lockheed Martin did not published any other information about the project. But according to an unconfirmed report, Lockheed Martin used pre-configured system of 8 PS3's. The pre-configured system of 8 PS3's offers theoretical performance of greater than 1 TFlop [10].

1.1 Scope of the Project

The scope of this project is to evaluate the suitability of the Cell BE (Cell Broadband Engine) for WPA cracking and evaluate whether the WPA encryption is strong enough and offers enough security. Therefore it is tried to crack WPA by using brute force password cracking method, because it is the only known way to crack WPA [23] (WPA cracking is investigated in next chapter). A brute force password cracking is generally time consuming and multi-core platforms should be well suited.
Based on research by Nick Breese about using PS3 with a Cell BE processor to crack eight-character password [3] and on research by Lockheed Martin [14], it is decided to verify that this platform can offer enough computational performance to implement brute force password cracking of WPA. This investigation consists in exploring brute force password cracking techniques, exploring the algorithm for WPA cracking, analysing the Cell BE architecture and implementation techniques, using profiling and metrics for evaluating the algorithm. And it is investigated how to modify the algorithm for parallel execution on the Cell BE processor.

1.2 Problem Definition

The project problem definition is formulated as follows:

How efficient is the Cell BE processor for cracking the WPA standard using brute force password cracking?

In order to solve this problem, several subgoals have been identified as can be seen in this list:

- Study of WPA and weaknesses...
- Study brute force algorithms and existing algorithms for WPA cracking...
- Study platform, tools and implementation techniques...
- Evaluate existing algorithms for chosen platform...
- Modify algorithm for Cell BE...
- Implement modified algorithm...
- Benchmark of implementation...
- Optimise implementation...
- Benchmark of optimised implementation...
- Evaluate, based on the previous sub-goals, Cell BE for brute force password cracking...

1.3 Development Model

The project development model is adapted from the A^3 model (A cube) used at Aalborg University. This A^3 concept provides a good design trajectory for designing and implementing signal processing systems. The original A^3 model divides development into three different domains: Application, Algorithm and Architectute (therefore A^3) [5]. A generic diagram of the A^3 model is shown in figure 1.1.

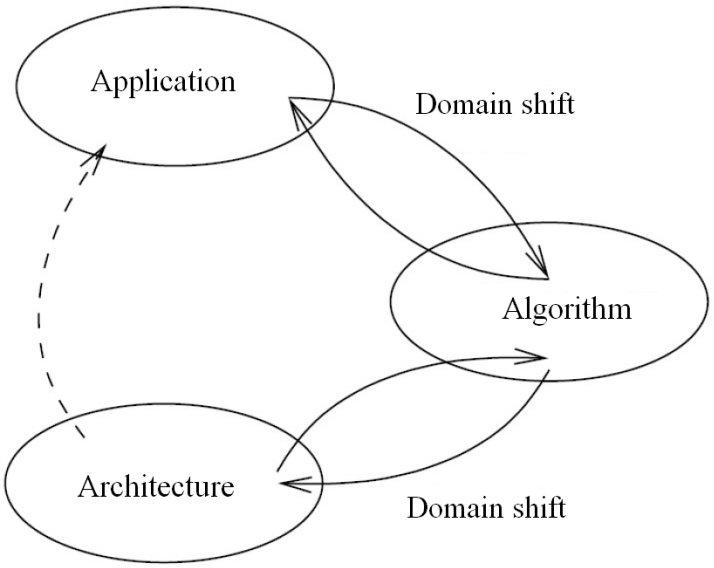

Figure 1.1: The A^3 development model divides development to three domains [5].

In original A^3 has each domain different meaning.
The Application domain is the highest level, where the problem is investigated in general. Several solutions are proposed and evaluated by this initial analysis. General requirements are also defined on this level.
The Algorithm domain serves for shifting proposed solutions into algorithms. Each algorithm can be defined as block diagram, flow graphs or can be expressed as programing language code (C is usualy chosen or Matlab for testing).
The last domain is about choosing the best fitting architecture. The code is investigated in detail (possibility to parallelisation, memory consumption, critical parts - bottle necks, data or command oriented parts, etc.). According to this investigation, the architecture, which can offer the best results for given requirements, is chosen. Implemented solutions are evaluated (some benchmark) and results are compared with requirements from the highest domain level (feedback between Application and Architecture levels).
The Original A^3 is more described in [5],[22] and [31].
Noteworthy is that several iterations are performed across different domains, hence an application can be expressed in more ways through different algorithms. An algorithm can be also mapped into more than one architecture.

Adapted A^3 model

The adapted A^3 model also contains these three domains, but the meaning of each of them is adapted. A diagram of this adapted model is in figure 1.2. The meaning of abstraction levels is as follow:

Application
This highest domain level serves for initial analysis and study of the problem, formulating requirements on application and define the main blocks of application.

Algorithm
This domain level is focused on algorithm, what it means to work with requirements from higher level and build a block diagram of algorithm. Make up code and test function of the algorithm.

Architecture
The algorithm from previous domain level is examined and fitted to the architecture. In comparision with original A^3 where is examined more possible architectures. In this adapted A^3 is platform given. Thereby iterations between algorithm and architecture level do not include different architectures, but in these iterations is changed the code and ways of matching the code to the platform.

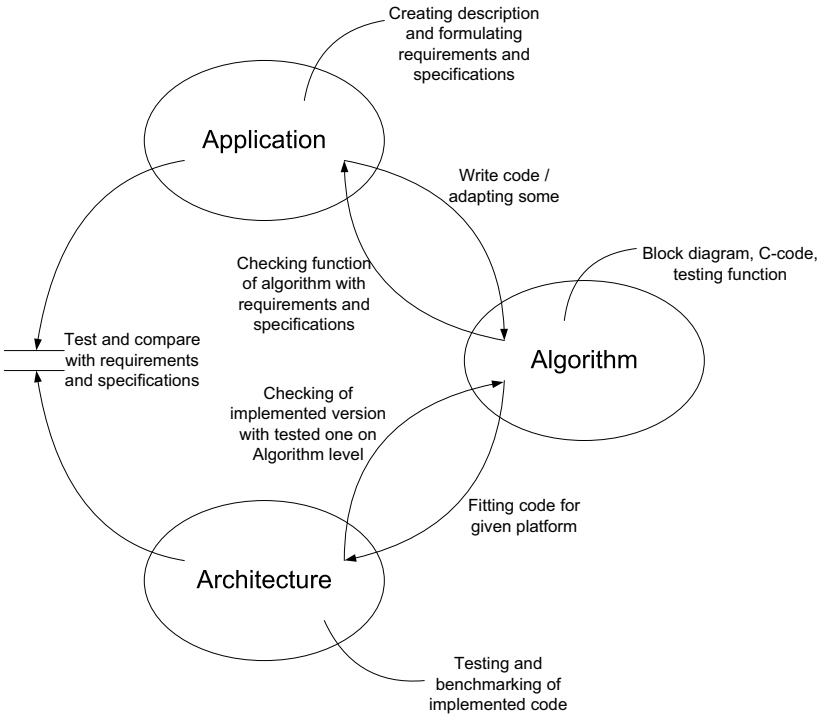

Figure 1.2: Generally the adapted A^3 development model with description of each domain level and shifting between domains.

1.3.1 Use of the Model

According to diagram in the figure 1.2 (adapted development model in general terms) each domain level is described separately. Below the description is a new diagram, which presents the development model with concrete terms actually used for this project. This diagram is shown in the figure 1.3.
This subsection of the project is updated during the writing and developing, therefore it is possible to find here remarks about parts and problems described later on.

Application

On this domain level is the given problem investigated generally. The security standards used in WiFi are explored with focus on the WPA and possible weaknesses used by cracking tools. Just mentioned tools are investigated too. Tools like Wireshark, airodump, aircrack-ng are tested. The requirements for application are formulated. The second and third chapter belongs to this domail level.

Algorithm

Based on requirements from previous part (application domain), a block diagram of program is made. As data input are used captured packets by airodump-ng tool. Parts of the communication containing 4-Way Handshakes were captured and stored like files. By network traffic analyzer tool Wireshark was distiled related data to authentication process.
The C code parts destiled from open source wpa_supplicant were conected together and was expanded by passphrase generator. For each generated passphrase this application process subsequently:

- Derivation of Master key from passphrase.
- Establishing Transient key from Master key.
- Calculating message integrity check.
- Comparison of calculated and captured message integrity check.

In case that captured message integrity check is equal to the calculated one, then used passphrase is equal to the generated one in this tempt of password cracker.
The performance analysis and profiling is performed to investigate the possibilities of parallel processing. Based on the code profiling can be decided tasks dividing and concept of parallelisation.
With this domain level is related the fourth chapter called Algorithm.

Architecture

As a first step to match PC (Personal Computer) code to PlayStation platform is adapting code for PowerPC. Further analysis of code provides information about parallelisation and about most consuming functions. Once the granularity of the algorithm is known, it is possible to implement code to Synergistic Processing element and start with optimisations of the code for better fit and performance utilisation of resources. The benchmarking is inseparable part of optimisation process to evaluate benefit of changes. The implementation contains two iteration steps. Firs is described in the fifth chapter. Two most consuming parts was replaced. In the first iteration it was SHA1 function and kernel functions to handling memory. The second iteration step replace SHA1 function again, but with rewritten code in SSE manner. This is described in the sixth chapter.

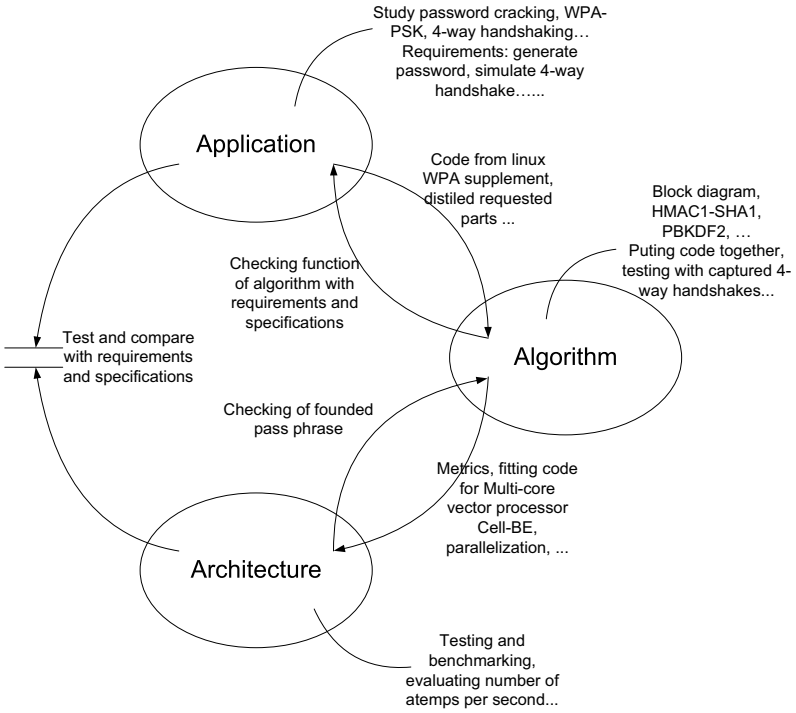

Figure 1.3: The adapted A^3 development model in real terms.

Chapter 2

WiFi Security Standards and WPA Cracking

This chapter offers deeper descriptions of important parts in WiFi security and provides fundamental knowledge of the following: i) WiFi security standards, how they work, where the weak points are, methods and approaches for hacking secured WiFi networks; ii) description of the techniques for password cracking and especially brute force password cracking; iii) an introduction to existing tools and algorithms for WPA cracking.

2.1 WiFi Security Standards

This section provides an overview of WiFi encryption standards. These standards are certificated[1] by the WiFi Alliance much like the devices, which incorporate those standards. The main reason is interoperability between all certificated devices. All devices with the WiFi trademark (logo in figure 2.1) have been certified[2] by the WiFi Alliance and meets all the requirements.

[1] certificated = to have certificate
[2] certified = to be proven

Figure 2.1: The WiFi Alliance trademark logo[2].

In these security standards stand security options like MAC (Media Acces Control) / MAC ID / MAC address filtering. In the AP (Acces Point) the list of MAC addresses is stored and only clients with a MAC address in the list are allowed to connect to the AP [25]. Unfortunately, spoofing any MAC address (i.e., making a fake MAC address) is fairly easy.
Another additional option is SSID (Service Set Identification) disabling. SSID is the identifier of the WLAN (Wireless LAN) which is broadcasted by the AP in the beacon frame[3] to enable easy selection of the WLAN by the user. Disabling the SSID makes the network invisible for the ordinary users and SSID has to be set in the client's network manager[25]. However, there are tools for sniffing the network communication, which can detect the WLAN with disabled SSID broadcasting [1].
Besides the two previously mentioned mechanisms, some encryption techniques can be applied to further secure wireless connections. In the following part of this section, the WEP, WPA and WPA2 standards are described.

2.1.1 WEP Encryption

WEP is an acronym for Wired Equivalent Privacy. It is basic encryption protocol from 1999 - included in the first IEEE 802.11 wireless standard. This protocol is based on the RC4 encryption algorithm with secret key. RC4 is a stream cipher, which generates a stream of pseudo-random bits [19]. WEP supports sizes of key from 40 to 104 bits, but many vendors refers to 40 and 104 bits encrypting as to 64 or 128 bits encrypting, because of the extra 24 bits which represent the IV (Initialisation Vector) that is required for control broadcasting [33].
The IEEE 802.11 standard has two types of authentication - Open system and Shared key. With Open system authentication, a client can join any network and receive any message that is not encrypted. With Shared key authentication, only clients with valid and correct authentication keys are able to connect to the network. The basic approach is based on a

[3]Part of the wireless network protocol

request/answer scheme. In an open system, a client sends an authentication request, the AP authenticates this client and the client can connect to the network. In a shared key system, the AP answers for the authentication request by a random challenge message to the client and the client is answer an encrypted session of the challenge message. Then the AP decrypts it and when it is matched with the sent one, authentication is successful and the client can connect to the network.

Problems with security, when the WEP encryption is used, are connected with RC4 stream cipher and with adding a field for integrity check to the packet. This integrity check is based on CRC-32 (Cyclic Redundancy Check). Each message is encrypted with a pseudo-random key. To ensure that two messages will not be encrypted with the same key, the 24 bits long IV is added to the shared key to create RC4 keys different for each packet. But there is still a certain probability that the same key will be used. When this happens, it is called IV collision. For 24 bit long IV there exists a probability that the same IV is repeated each 5000 packets [33].

In August 2001, S. Fluhrer, I. Martin and A. Shamir published the article "Weaknesses in the Key Scheduling Algorithm of RC4" [12] where they describe weaknesses of RC4 and in the end even a possible way of attacking the WEP. They explain a passive attack based on statistical analysis. The passive attack is undetectable eavesdropping of communication between the client and the AP. The time needed for eavesdropping depends on the amount of transmited data, but usually it is not more than one minute. This article was the beginning of the end of the WEP like secure transmition and shortly after the open source program "AirSnort", which enables to determine the WEP key from eavesdropped traffic on the network, was released [33].

An extension could be an active attack. It is not based only on monitoring but also on transmitting. One approach is a packet injection [6]. This is used when the attacker knows exact plaintext for one encrypted message. Then it is possible to construct the message and adjust correct packet encryption and integrity check, and this packet is accepted like a valid one. A modification of this type of attack is when the attacker has partial knowledge of the content - for example a command for Telnet, file servers and so on. The attacker should modify this command which should be accepted and executed [6]. Another approach is focused on predictable content of address fields in the packet. If the attacker achieves to send a packet to his machine over the Internet, then the message is decrypted by the AP and sent over the Internet. By this, the attacker obtains a plain text version of the encrypted message [6].

Sometimes the aim of the attack is to disable the network. Then the attacker should send fake messages and overload the attacked device - so that no-one can use the network. This

is called DoS attack (Denial-of-Service attack) [23].

2.1.2 WPA/WPA2 Encryption

As seen in chapter 1, WPA stands for WiFi Protected Access. WPA has been accepted in 2002 as a temporary solution by the WiFi Alliance, as a response to delayed development of the IEEE 802.11i standard (nowadays known as WPA2)[35]. WPA is an intermediate stage of security: it is back-compatible with WEP (it uses the same encryption core - RC4) and forward-compatible with IEEE 802.11i. For integrity check, WPA introduces MIC (Message Integrity Check) and for data security it introduces TKIP (Temporal Key Integrity Protocol) using dynamically changed key and extended IV to 48 bits [28]. MIC uses an algorithm called Michael. Michael represents the strongest possible option, that developers team of WPA should use for keeping compatibility with older network cards. Due to unavoidable weakness of the Michael algorithm, WPA contains a security mechanism which detects attempts to break TKIP and temporarily blocks communications with the attacker.
WPA2 added CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol), that is more sophisticated and also needs more powerfull HW (Hardware) than TKIP and MIC [35].
WPA/WPA2 is designed for working with 802.1X authentication servers which distribute different keys to different clients, but it is possible to use it with a pre-shared key - PSK, when all clients use the same access password [35]. This configuration with PSK is mostly used for SOHO (Small Office/Home Office) solutions, because fully using of RSN (Robust Secure Network) involves fully compatible RSN HW (usually means new HW), server/servers for distributing keys and professional/expert seting up.
PSK authentication was the only known weak point of WPA till the conference PacSec 2008, where Erik Tews and Martin Beck disclosed the weakness of TKIP. This Tews/Beck approach cannot recover the keys, but focus on an exploitable hole [11]. Thus, PSK authentication remains the only known way to crack WPA and obtain the keys.

2.2 Password Cracking

There are several approaches to password cracking and they have a common goal - to find the password. In the computer world, this means to recover password from data, from

code. But there are other approaches in the real word. They can be based even on human vulnerability like "social cracking", when someone tries to worm password out of a person who knows it (calling to secretary and pretending to be IT maintainance and ask for the password etc.). It could offer easier way to the password cracking.
But back in the computer world, we can see three main streams:

- Dictionary attack
- Brute force attack
- Direct attack

A dictionary password cracking is based on repeatedly trying guesses for the password. The dictionary attack involves a database of the most frequent passwords. And then use them one by one to guess the password until the acceptance of a password or the end of the dictionary. There is problem with this approach, because the dictionary have to contain the thousands, the hundreds of thousands passwords to achieve complexity. And the complex dictionary can achieve the size of the tens of gigabytes. Then processing of the dictionary leads to extremely high data bandwith.
Brute force password cracking is based on repeatedly trying too. But there is no dictionary, there is a password generator which generates passwords. Usually it generates all possible combinations, systematically one by one. But this approach involves incredible high numbers of tempts. In case of using all visible ASCI characters (ASCI codes from #32 to #126 = 95 possible characters on each position) for password with length of eight position exists more than 6.5 quadrillions possible combinations (6.5 peta).
Direct password cracking is based on knowledge about the system. This knowledge is involved in password cracking. Depend on the kind of knowledge the process can vary. For example, the encryption algorithm was breaked and some reverse technique was developed. Than password cracker directly distilate relevant data from captured ones and using reverse technique compute the password.
Real tools usually combines these approaches. For example, by using some knowledge in the system, some parts could be precalculated and stored in a dictionary.
It is possible to divide the concepts of password cracking by many aspects. It is different to try to break in a system where the security mechanism is unknown, or to try to open a file which was encrypted with an unknown password, but with known mechanism. Another situation is eavesdropping encrypted communication and cracking the password to understand the content of communication.

Therefore the password cracking can differ by the time: on one hand real time password cracking¨, and on the other hand, retroactive cracking.
Another point of view is connection. Online password cracking, when the system has to be connected, or offline, where the necessary information is collected when the system is connected and the cracking can be done offline.
Another way to divide password cracking can be using of real system vs. simulated system.
There are other possibilities, to divide the password cracking, but the above mentioned ones are related with this project.

2.3 WPA-PSK Cracking

In this project, an investigation of WPA cracking has been carried out, and it has been found out that WPA-PSK cracking is the only possible way to crack this security standard. According to the standard for PSK after authentication occurs key derivation. Key derivation consists of two handshakes. The first is 4-Way Handshake for PTK (Pairwise Transient Key) and GTK (Group Transient Key) derivation. And the second one is Group Key Handshake for GTK renewal[23].
The cracking concept is based on imperfection in 4-Way Hanshake, where the PTK and GTK keys are derivated from PMK (Pairwise Master Key). In the WPA-PSK system PMK is derived from PSK [23].
Figure 2.2 shows the hierarchy of keys. The way to derived KCK (Key Confirmation Key) from PSK is visible in this figure.Subsequently is calculated PMK from PSK, then the PTK is derived and first 128 bits of PTK represents the Key Confirmation key. KCK is so important for WPA-PSK password cracking, because it is used for computing MIC. The above mentioned steps are part of 4-Way Handshake.

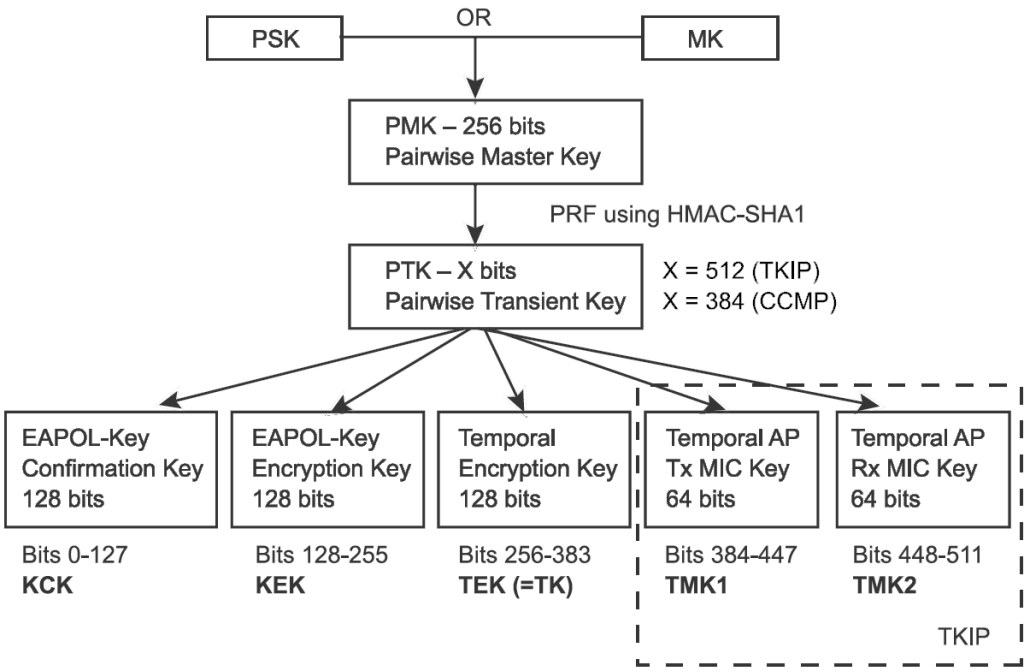

Figure 2.2: Pairwise Key Hierarchy [23]. On the top are inputs (PSK or MK). It is used for establishing of the PMK, consequently is established PTK. PTK consists of KCK, KEK, TK, TMK1 and TMK2.

Figure 2.3 represents the 4-Way Handshake. The four messages are exchanged between the client and the AP during the 4-Way Handshake. Before handshake the PMK is calculated from PSK and SSID. The PTK is derived from PMK, fixed string represented by "Pairwise key expansion", MAC of AP, MAC of client, and two random numbers: ANonce and SNonce[4]. These random numbers are generated by the authenticator and supplicant, respectively. The first message from the AP is non-crypted and contains ANonce. The client generates SNonce and is now capable to derive PTK and temporary keys. Then the client sends SNonce and MIC key calculated from the second message using KCK. Once the AP has received second message, it can then use the sent SNonce, because the message is non-crypted. With SNonce, the AP can calculate PTK and then can calculate MIC with calculated keys. If this MIC is identical with the MIC sent in the second message by the client, then it is proved that both (AP and client) have correctly calculated temporary keys and that the client has the correct PSK.

[4] Nonce stands for number used once; often a random number. A represents Authenticator (AP) and S represents Supplicant (client).

The other two messages of 4-Way Handshake are not so critical for this project, because the information in the first two messages are enough for password cracking. Even though it is enough, it is important to eavesdrop the whole 4-Way handshake to be sure that the handshake was successful and that the information in the first two messages is valid.

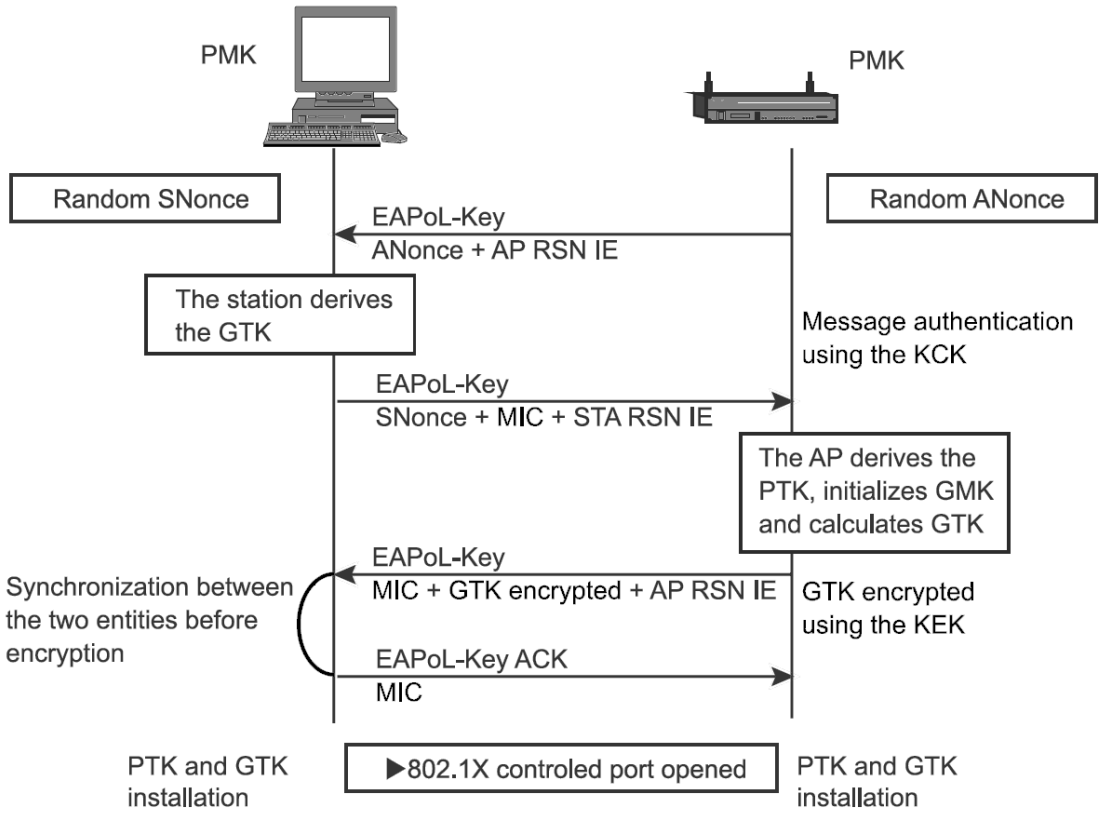

Figure 2.3: 4-Way Handshake [23]. All the necessary information is in two first messages: ANonce, SNonce, MIC, SSID, MACs. It is enough for WPA-PSK cracking.

2.4 List of Requirements

Now is the whole process of 4-Way handshake explored, therefore is possible to formulate list of requirements for next domain level - Algorithm:

- Eavesdrop whole 4-Way handshake.
- Isolate both nonces, message and MIC from captured handshake.
- Generate passwords (PSKs).

- Simulate derive of MIC from generated PSKs, SSID, MACs and eavesdropped data.
- Compare computed MIC with eavesdropped one.

2.5 WPA Cracking Tools

From the time of releasing WPA there are teams of security experts and hackers, who try to break it and some developers have even released tools for cracking WPA. The tools are usualy connected with some platforms. Not only the PC is considered, there are also tries to involve other platforms. It is worth to mention the FPGA (Field-Programmable Gate Array), where FPGA is used to accelerate the most critical parts. For example, one product by the company Pico Computing Inc. is the PC card containing an FPGA (there are modifications with few interfaces like CompactFlash, CardBus, ExpressCard format) [15]. Another interesting approach involves the computing power of the GPU (Graphic Processing Unit). NVIDIA Corporation responded to the developers by releasing CUDA (Compute Unified Device Architecture). This architecture allows writing code, that is executed on GPU. Again the most critical part can be accelerated.
For this project two aproaches have been considered. Exploring tools available for the PC or porting and accelerating those tools on the Cell BE processor. Explore the tools for the PC was prefered for two reasons. Cell BE platform is installed in the Embeded Laboratory at AAU while tools for PC installed on laptop offers mobility (for capturing traffic etc.). The second reason was, that many tools are released and precompiled for PC. The tools are usually released for the Linux OS (Operating System), due in part to the open source.
The tool Aircrack-ng consists of many parts [1], with each has a specific purpose. Whole tool has been installed with the specially adapted driver (to allow monitoring mode and the packet injection). Therefore this tool is more complex and offers more ways of cracking several standards as compared to other tools. The main used parts are for monitoring, capturing frames and cracking. These parts were used in this project to capture 4-way handshakes. Therefore was established testing connections of AP and laptop using WPA-PSK. Another laptop with Linux and aircrack-ng tools was used for capturing 4-Way Hanshakes. In the listing 2.1 is a list of main commands for using Aircrack-ng tool. First line checks if any programs are using interface. Second one switches the interface to monitoring mode. On third line the command shows whole WiFi environment - all access points and clients, this is in figure 2.4. For capturing 4-Way Handshake is used the

next command on line four. Command on line five deauthenticates the station and forces the station to repeat the authentication and 4-Way Handshaking. The last command executes password cracking with captured file .cap and with dictionary file .lst, in figure 2.5 is shown password cracking process.
The list of used HW for this test:

- AP: D-Link DL-524
  - MAC: 00:1B:11:FD:E8:B2
- Client: laptop ASUS F5SR-AP028C, OS: Ubuntu 8.04 live
  - MAC: 00:1D:E0:19:41:23
  - WiFi card: miniPCI Intel centrino agn4965
- Monitoring/capturing: laptop Lenovo ThinkPad R61i, OS: Ubuntu 8.04 with aircrack-ng
  - WiFi card: miniPCI Intel centrino agn4965

```
1 airmon-ng check wlan0
2 airmon-ng start wlan0
3 airodump-ng mon0
4 airodump-ng -c 6 --bssid 00:1B:11:FD:E8:B2 -w psk mon0
5 aireplay-ng -0 1 -a 00:1B:11:FD:E8:B2 -c 00:1D:E0:19:41:23 mon0
6 aircrack-ng -w password.lst wpa2.*.cap
```

Listing 2.1: Commands used to control Aircrack-ng tool.

```
CH  4 ][ Elapsed: 3 mins ][ 2009-03-10 23:58

BSSID              PWR  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID
00:1B:2F:0C:08:3E  167      174        0    0  11  54. WPA  TKIP   PSK  NETGEAR4
00:1C:DF:7E:BF:FD  166      152        0    0   6  54  WPA2 CCMP   PSK  Belkin_N
00:1B:11:FD:E8:B2  164      152        0    0   6  54. WPA  CCMP   PSK  LaWifi
00:11:6B:12:7D:80  164       65        0    0  11  54  WEP  WEP         level2_adi
00:90:4C:91:00:01  163       27        0    0   6  54  WEP  WEP         moedelokale

BSSID              STATION            PWR   Rate  Lost  Packets  Probes
00:21:29:70:06:C3  00:17:AB:CE:04:F6  170   1- 1     0       17  The Shizzle net
(not associated)   00:15:AF:26:C5:60  181   0- 1     0      323  qper
(not associated)   00:13:CE:CE:B8:03  175   0- 1     0      142  freepr0n,AAU-1x
(not associated)   00:1D:E0:19:41:23  170   0- 1     0        9
(not associated)   00:1F:3C:28:92:4F  170   0- 1     0        6
(not associated)   00:1F:3A:BB:5D:10  168   0- 1     0        2
(not associated)   00:1B:EA:E6:33:C2  165   0- 1     0       31  102
(not associated)   00:1F:5C:28:A8:7D  163   0- 1     0        3  extremo
```

Figure 2.4: Monitoring of environment. Access Points in upper part and clients in lower.

```
alfieri@alfieri-laptop:~/Dokume
                                Aircrack-ng 1.0 rc1

                    [00:00:00] 12 keys tested (89.28 k/s)

                        Current passphrase: iloveyou

      Master Key     : 64 73 69 C8 7C 2F 4E CB 30 BF E8 F2 62 BB C6 09
                       5D 12 70 1A E5 34 BB D6 D2 BA 28 87 EE 9D E8 5F

      Transcient Key : D1 2C 86 80 08 BC BB AD 2E 75 F5 7F E1 40 31 A7
                       08 76 0F D5 EC 00 F0 86 30 E8 8A 72 23 69 08 F3
                       61 CD 1B 04 CF D1 87 8D D5 5B 67 8C 3D 6A 70 03
                       2F D8 C5 22 6D 1B 0B 8D 8D 37 28 AC 12 16 7A 66

      EAPOL HMAC     : E7 FB 82 40 7A F4 78 FF EA FB C2 3B 18 50 91 1F
```

Figure 2.5: The screenshot of executed password cracking with aircrack-ng application.

There were established and captured several different 4-way handshakes with different passwords. The MAC addresses are mentioned above and the SSID was ”CellBE” for all testing connections.
The captured 4-way handshakes are present on enclosed CD in folder \handshakes and the names of subdirectories are equal to used passwords.

The result of capturing are files containing the traffic between the AP and the Client. And in these captured files with traffic are 4-way handshakes. For analysing these files, another tool was used: Wireshark [9]. Wireshark is Network Protocol Analyzer.
The relevant informations was distilled from files by Wireshark. The example of distilled data is in table 2.1.

Passphrase: ”12345678”	
SSID	CellBE
PMK	dc646016b616bf772e34116421554f44f1c4f5b22d93efd80552225e703cb164
ANonce	4e9535c9e2a2dd7594a0b575a68ae0de1633e9ae58e1e617cc0b704b2dbd5425
SNonce	e24c20d6d934b598d119d13fcd80d3a3e24b0f6a73fee11813c5d762e316e21f
AP MAC	00:1B:11:FD:E8:B2
Client MAC	00:1D:E0:19:41:23
MIC	551cd915159d6ff62e1be92861a54ae8

Table 2.1: Table with example of distilled relevant information from captured file.

Chapter 3

PS3 Platform

The proposed platform for this project is Sony PlayStation 3 containing Cell BE on 3,2 GHz. The PS3 is a game console from Sony and has this HW [7]:

- CPU: Cell Broadband Engine[16]
- 256 MB XDR main RAM
- GigaBit Ethernet network card
- Special edition of a graphic card from NVIDIA with 256 MB of video RAM
- 40 GB Hard drive, Serial ATA
- Multi-format optical drive (reads CD,DVD and Blue-ray)
- 4x USB 2.0 ports
- Bluetooth 2.0

Proposed HW is Cell BE processor, which offers interesting possibilities due the innovative way of the internal chip architecture. The IBM company offers two-processors racks with two Cell BE processors and one gigabyte of memory for each processor. The game console PlayStation 3 represents accessible way for testing of projects based on the Cell BE processors. Therefore focus of next part is on the Cell BE architecture.

3.1 Cell BE Architecture

The Cell BE processor had been developed for six years by Sony, IBM and Toshiba. The Cell BE is a heterogenous chip multiprocessor, which consists of PPU (PowerPC Processing Unit) and eight specialised co-processors based on SIMD (Single Instruction Multiple Data) architecture called Synergistic Processing Unit (SPU), which is for data-intensive processing (cryptography, media, scientific applications etc.). Therefore the Cell BE processor provides two levels of parallelism. One level provided by SIMD instruction and second one on level of independent tasks, that may be executed in parallel on different SPUs [4].

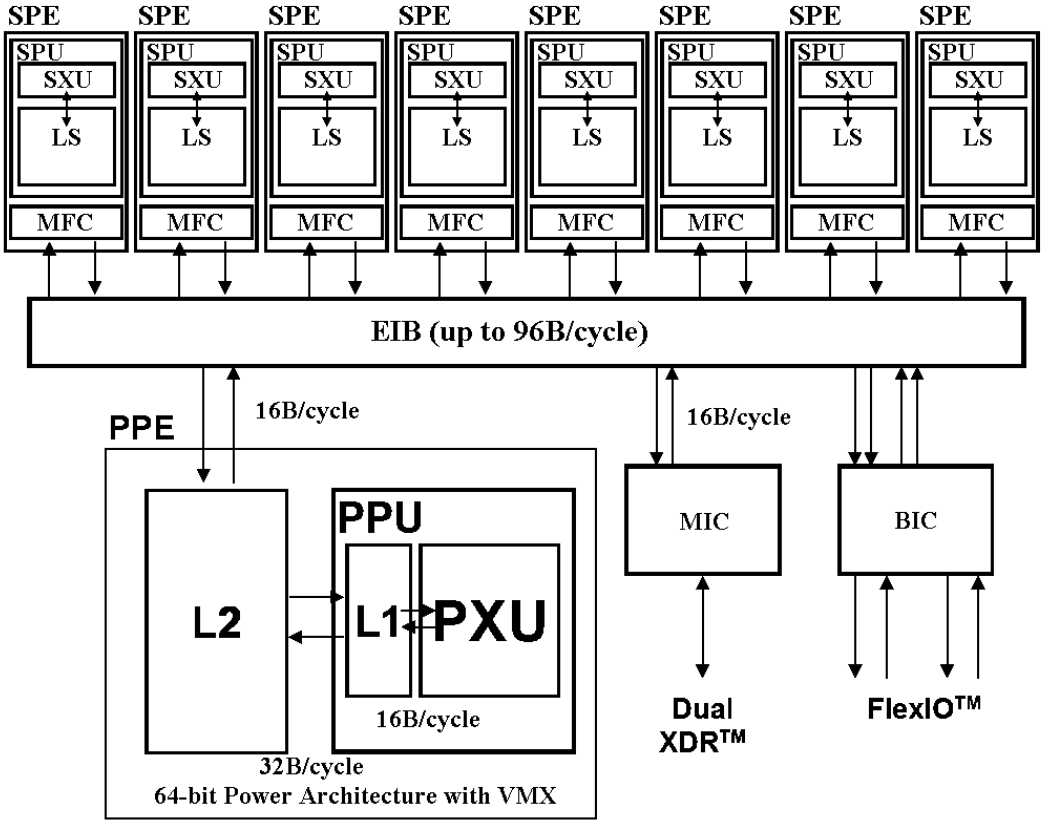

Figure 3.1: Cell BE block diagram introduce main parts of Cell BE. Diagram presents main bus - EIB, what connects all elements like SPEs, PPE and memory. Figure adopted from slides of [4].

The Cell BE processor is realized on 235 mm^2 chip, consists of 241 milions transistors, runs on clock frequency of 3,2 GHz (tentatively in lab runs > 4GHz). Concept idea differs from classical way of increasing performance. As classical concept is assumed increasing of the clock frequency, cache memory and numbers of homogenous cores, but concept of the Cell BE is processing unit with many specialised co-processors. This structure is introduced in the figure 3.1. Main processing unit, PPE (Power Processing Element) is based on core of PowerPC 970. It is 64b architecture with two multithreaded logical cores (one physical core) and includes VMX (Vector Multimedia Instructions) - instruction set

for processing of SIMD. PPE also contains 32kB of L1 cache and 512kB on L2 cache[1] and 2 way SMT (Simultaneous Multi Threading). Specialised co-processors, each SPE (Synergistic Processing Element) is RISC (Reduced Instruction Set Computer) with 128-bit SIMD instruction set, register file 128x128-bit, local store 256 kB and MFC (Memory Flow Controller). Each SPE has own program counter and executes the threads given by the controlling PPU. The RAM memory is accessible via EIB (Element Interconnect Bus)[4].

The Cell BE processor used in PS3 offers to the developer only six SPUs. One SPU is disabled to improve production yields and another is used to manage the hypervisor that allows Linux to run as a guest OS. It does important task, it keeps Linux from corrupting the Game OS [8]. The Cell BE architecture, as it is in PS3 is displayed in figure 3.2.

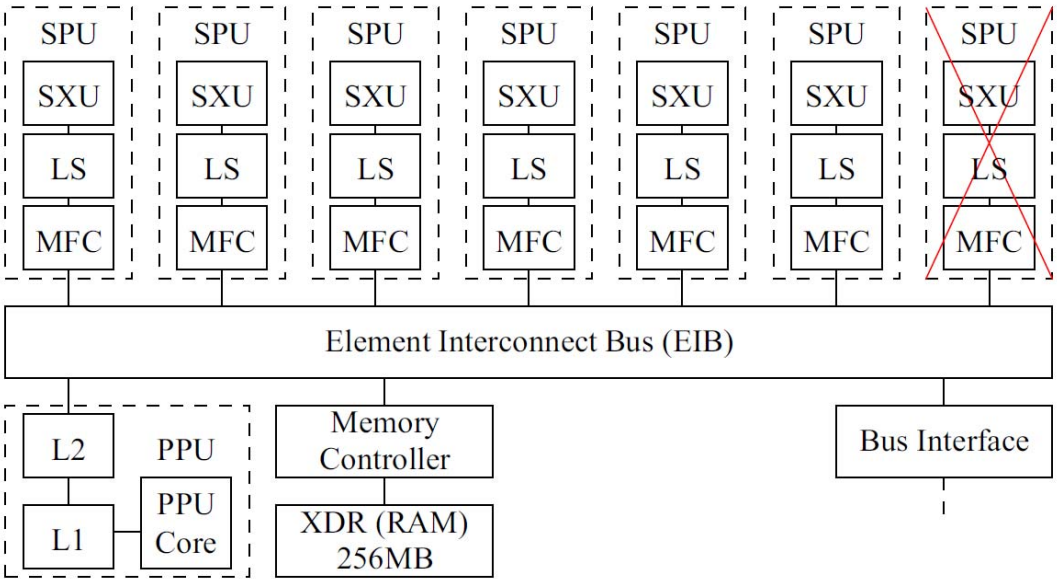

Figure 3.2: The PS3 Cell BE architecture. Element Interconnection Bus connects only seven SPUs, where one is used to manage hypervisor. Then EIB connects PPU and 256 MB RAM thru Memory Controller. Figure adapted from [31, page 32].

According to figure 3.2 the PPU contains two levels of caches, 64kB of L1 cache is parity protected and the 512 kB of L2 is protected with error-correction code. The OS runs on PPU and using of SPUs are controlled by the PPU. Clocked at 3,2 GHz, the PPU can theoretically achieve 2 x 3,2 = 6,4 Gflop/s of IEEE compliant double precision floating-point performance or even achieve 4 x 2 x 3,2 = 25,6 Gflop/s of non-IEE compliant single precision floating-point performance. It is assumed, that in this project the PPU will just

[1] L1,L2 - level one and level two cache

controll the SPUs and take care about maintenance of system [7].
The SPUs indeed presents the real performance for process huge amount of data. The SPUs are designed to run own programs from local storage. Each SPU with large 128-entry 128-bit vector register file can process simultaneously: 2 double precision values, 4 single precision values, 8 16-bit integers or 16 8-bit chars. The SXU is divided into even and odd pipeline (figure 3.3)and it can complete up to two instruction per cycle, what leads to the peak of 2 x 4 x 3,2 = 25,6 Gflop/s for each SPU or 6 x 25,6 = 153,6 Gflop/s for all six available SPUs, when single precision is used and one operation on each pipeline [7]. The SPU has two pipelines in SXU (Synergistic Execution Unit). The SXU contains six execution units, as it is shown in figure 3.3. The execution units are divided into two groups related to two pipelines. In each SPU are these execution units [18]:

- SPU Odd Fixed-Point Unit (SFS) which executes byte granularity shift, rotate mask and shuffle operations on quadwords.
- SPU Even Fixed-Point Unit (SFX) which executes arithmetic instructions, logical instructions, word shifts and rotates, floating-point compares, and floating-point reciprocal and reciprocal square-root estimates.
- SPU Floating-Point Unit (SFP) which executes single-precision and double-precision floating-point instructions, integer multiplies and conversions, and byte operations. (HW support only for 16-bit multipliers, therefore 32-bit ones have to be implemented in SW using 16-bit multiplies.)
- SPU Load and Store Unit (SLS) which executes load and store instructions. It also handles DMA requests to the LS.
- SPU Control Unit (SCN) which fetches and issues instructions to the two pipelines, executes branch instructions, arbitrates access to the LS and register file, and performs other control functions.
- SPU Channel and DMA Unit (SSC) which enables communication, data transfer, and control into and out of the SPU.

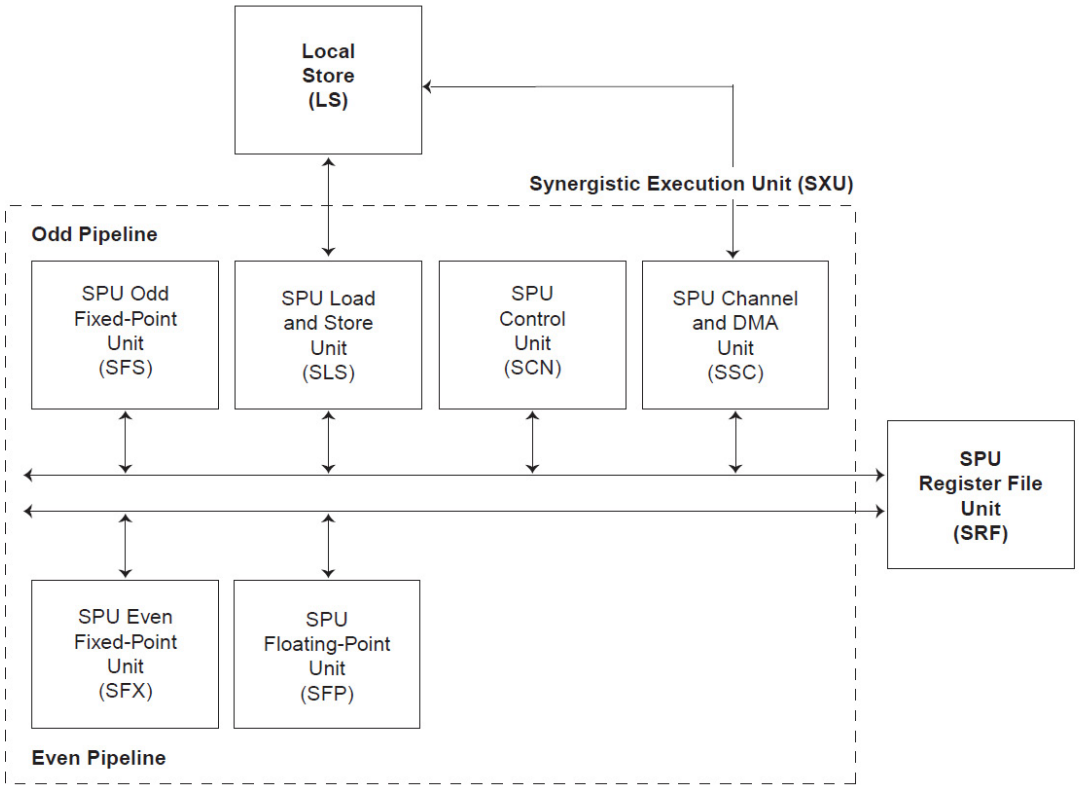

Figure 3.3: SPU Functional Units, odd and even pipeline and related execution units. Figure adopted from [18].

3.2 Implementation Method

The basic programming concepts are, that the PPE is just a PowerPC running Linux. That PPE manages SPE processes as POSIX (Portable Operating System Interface for Unix) threads.
According to heterogenous processor with two types of cores, the tool chain contains two compilers. One for PPU and one for SPU. ELF (Executable and Linking Format) object-files are used for SPU and PPU. And compiler tools embed SPE executables into PPE executables - one file provides instructions for all execution units[4].
The biggest issues are SIMD instructions and 6 SPU cores. Both aspects leads to parallelisation. Therefore the exploring of the algorithm is needed. There are few concepts of tasks dividing and two of them are presented in figure 3.4 [26].

- PPE-centric: Main application runs on PPE and tasks are executed on SPEs. PPE waits for results from SPEs.

- Multistage Pipeline: The data stream goes from PPE to first SPE, where is executed part of code and then the data stream is send to next SPE, where is executed another part. Last SPE send data stream back to PPE. Appropriate for sequential processing.
- Parallel Stages: When application contains huge amout of data, which can be divided, then PPU divides the data between SPUs and execution runs simultaneously.
- Service Model: Individual SPEs works like separate services, which are called by PPE when it is necessary.

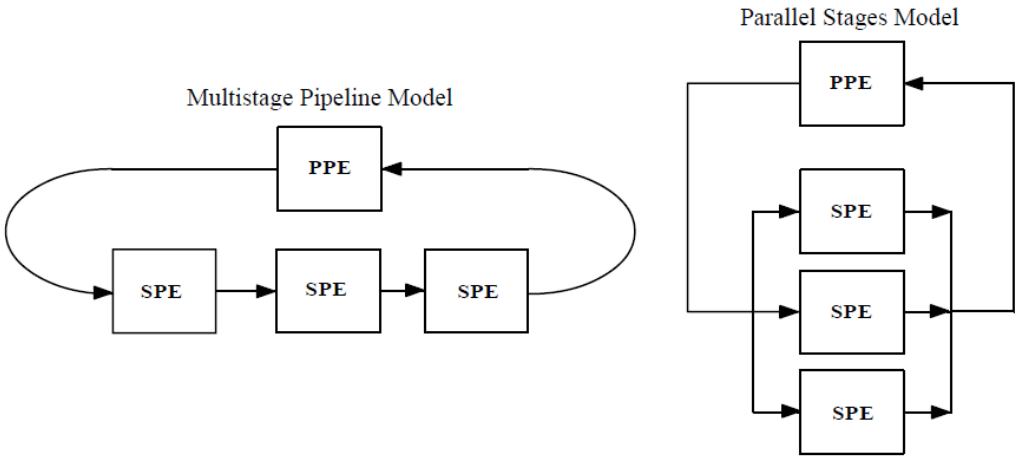

Figure 3.4: The tasks dividing examples, figure adopted from [26]

Chapter 4

Algorithm

This chapter belongs to Algorithm domain according to used development model. By the development model in this chapter is works with the requirements from Application domain and block diagram with working code are developed.

4.1 Block Diagram and C Code

According to the requirements, a block diagram of application is created. The requirements to capture handshake is not part of algorithm and it is replaced by already captured ones, which are used as input for developing application.
The requirements says to generate passwords and for the each one simulate 4-Way handshake and calculate Message Integrity Check (MIC). Then compare captured MIC with the calculated one. The process of simulating 4-Way handshake is divided to the three parts. Dividing is related with Key Hierarchy in figure 2.2. It means following parts: generating PMK from passphrase, derivation of KCK from PMK and calculating MIC. The designed block diagram is in figure 4.1
As entry point for writing the C code was used open source project wpa_supplicant. The code is complex and offers the full support for WPA and WPA2 to OS. The wpa_supplicant project web page was completelly downloaded and is located on enclosed CD in folder \wpa_supplicant_web. The whole supplicant contains hundreds of files with C code. In the figure 4.2 is a diagram with modules of supplicant. The source code was investigated and relevant parts of code were isolated, due to complexity of code it was more exacting.

The whole wpa_supplicant code is divided to many modules. It was evaluated as advantage and modularity was kept.
The C code is on enclosed CD in folder \C_code\PC.

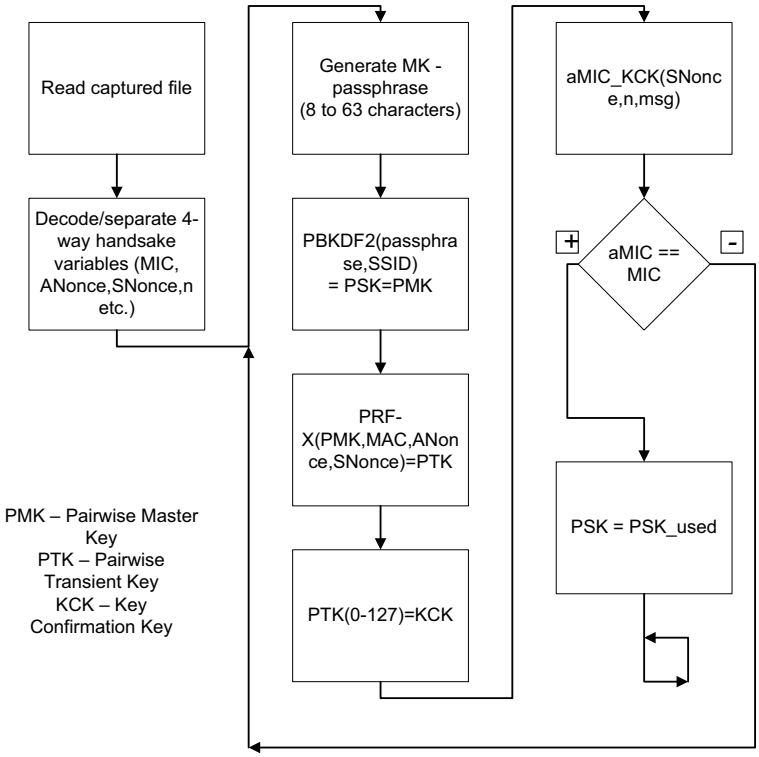

Figure 4.1: The diagram how to crack WPA [23]. It consists of loading captured data; generator of passphrases; simulating of the processes in WPA standard; and comparison of simulated result with captured one. In case of equivalency the guessed passphrase is the correct one.

4.1.1 Passphrase Generator

This part of code was written from the scratch and it is represented by only one function *next_pswd*. The inputs to the function are pointer to passphrase, length of passphrase and the minimal and the maximal value of considered characters. Some limited range of possible values is given by the visible characters, because any user can use only visible characters to entry passphrase. Therefore the range from 0 to 255 is not considered, but only codes between 32 and 126 (for example, all numbers have codes between 48 and 57). This function *next_pswd* returns pointer to the new passphrase.

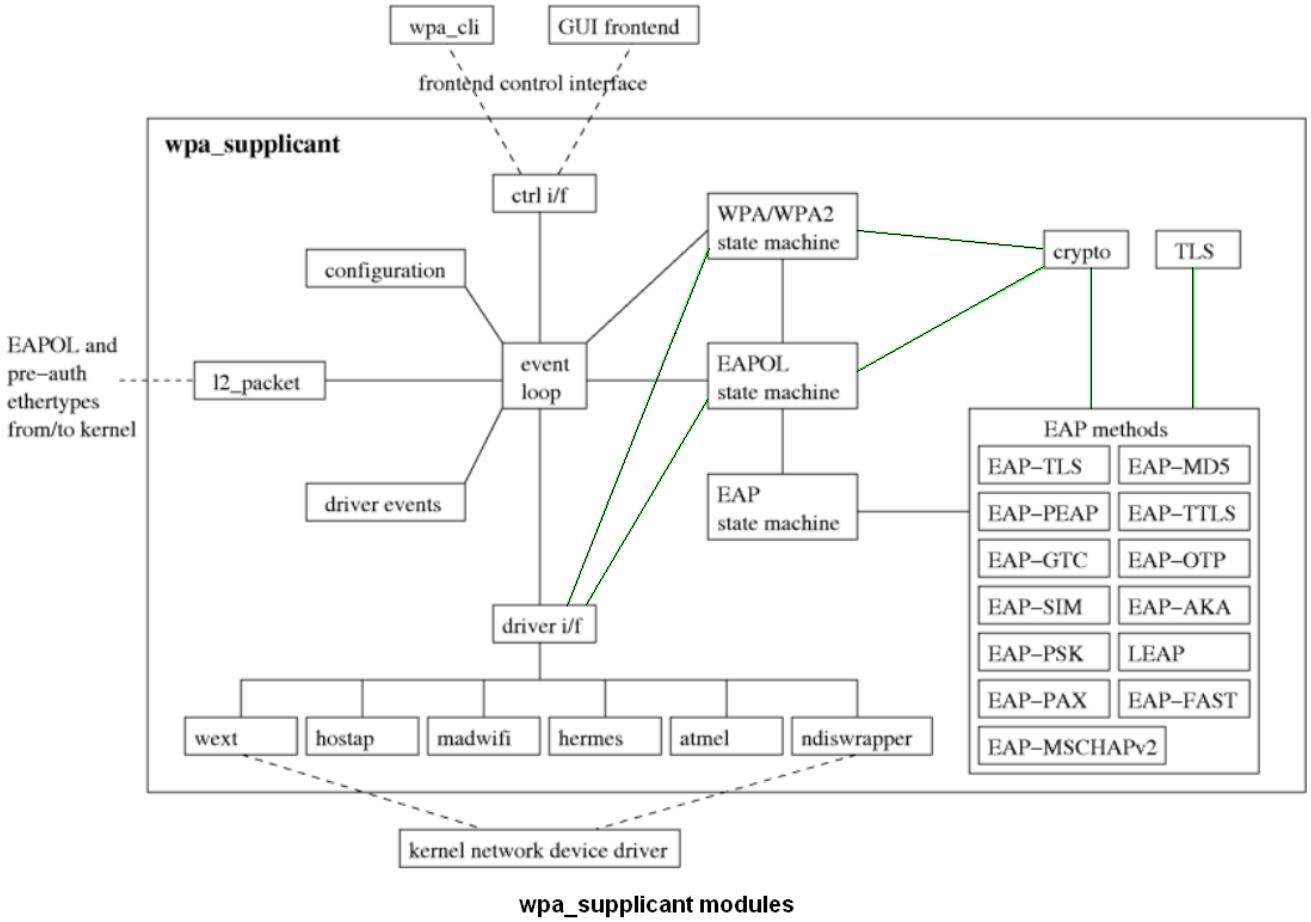

Figure 4.2: wpa_supplicant modules.

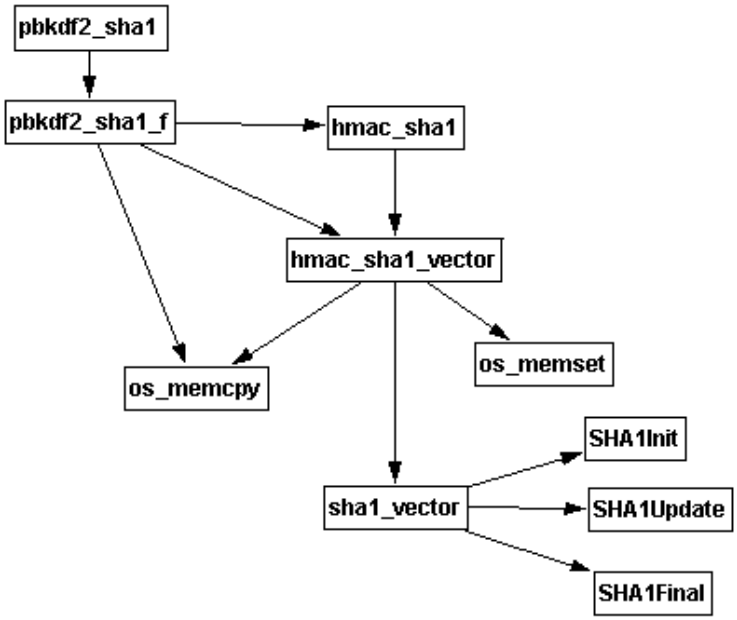

Figure 4.3: Call graph of PBKDF2 function for derivating PMK.

4.1.2 PBKDF2

The shortcut PBKDF2 stands for Password-Based Key Derivation Function. It is a key derivation function which derivates Pairwise Master Key. In distiled code has name *pbkdf2_sha1*. The Call graph of this function is in figure 4.3. Inputs are passphrase and ssid. This function performs 8192x times a cryptographic function *hmac_sha1_vector*.

4.1.3 PTK

Pairwise transient key derivation from PMK, ANonce, SNonce, MACs and constant string "Pairwise key expansion". The call graph for this function is in figure 4.4. The function has name *wpa_pmk_to_ptk*.

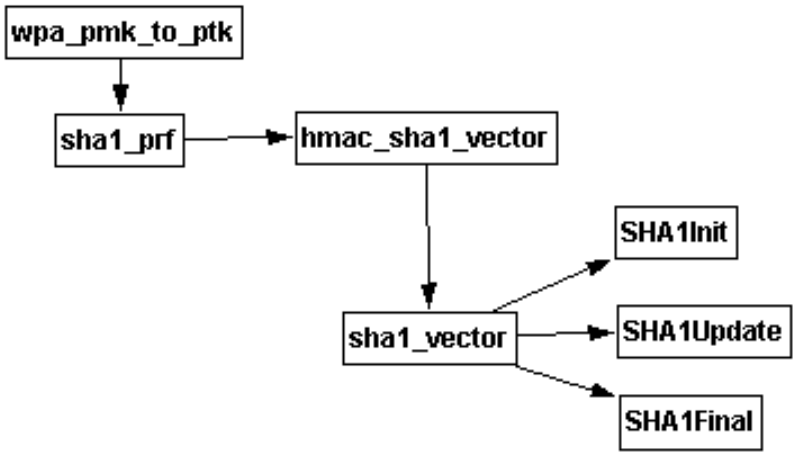

Figure 4.4: Call graph of the function derivating PMK.

4.1.4 MIC

This function calculate the Message Integrity Check. KCK is used to calculate MIC over the message in second frame of handshake. Only this function in the project depends on WPA standard. In case of using older WPA the MD5 hash[1] algorithm is used, in case of WPA2 the SHA1 hashing algorithm is used. This is visible in call graph of this function in figure 4.5.

4.2 Profiling and Performance

The complete C code was partly written and partly adapted from wpa_supplicant open source. The code was tested with correct passphrase "12345678" to proof the functionality. The C code was explored and executed with profiler to obtain information about the bottle neck.
In case of WPA2, when the MIC is calculated with SHA1 hash function, then all blocks uses and calls HMAC-SHA1 function. The HMAC-SHA1 is represented by function *hmac_sha1_vector*. This function is called 8192 times just by the PBDKF2. It is the highest amount and this function is the most consumption part of code. The others parts call the function *hmac_sha1_vector* only few times. The function derive PTK from PMK call HMAC-SHA1 only four times and the function calculating MIC call it only once.

[1]hash function converts a large amount of data into a small datum

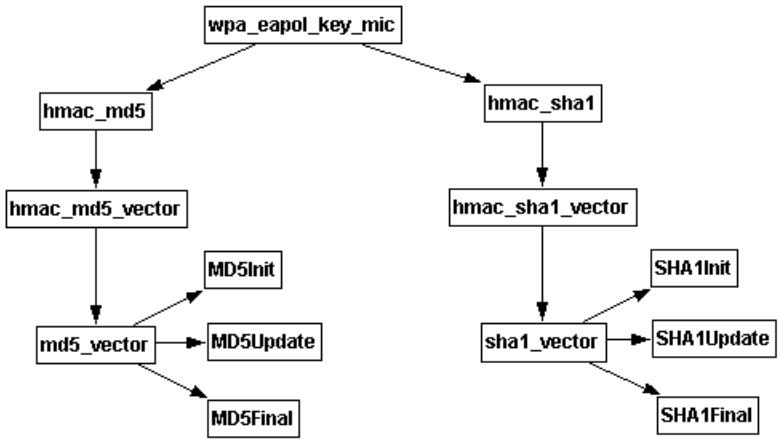

Figure 4.5: Call graph of the function calculating the MIC from KCK and message. There are two branches, one is used in case of older WPA (MD5 hash function) and second in case of the newer standard WPA2(SHA1 hash function).

The result of profiling is in listing 4.1. On the third line is represented whole program by function *main*. Of course it runs 100% of time, but column "self" means total amount of time spent in this function and it is zero, because this function just call other functions and do not perform any calculation. The next column named "children" represents total amount of time propagated into this function by called functions (children). For example function *pbkdf2_sha1* with children consumed almost whole time (1.50 of 1.51 seconds) and it was called 11 times by function *main* of total 11 calls, this is meaning of the next column named "called". The main part of consumed time by this function takes called function (child) *pbkdf2_sha1_f*, which runs 1.44 seconds (line 14,column children).
After exploring whole profiler output, the bottlenecks were located, that HMAC-SHA1 consumed 93.7% of the run time of whole code. In the code is HMAC-SHA1 represented by function *hmac_sha1_vector*. Another interesting result is, that functions *os_memcpy* and *os_memset* takes 0.46 and 0.20 seconds respectively. They represent OS functions handling data in memory (copy and set). The counter of bytes handled just by these two functions was added to code with result, that during one passphrase tempt is handled with total amount of 8.8 MB. They belongs to OS (kernel functions) and they are assumed like fundamental functions. Because they transfer data byte by byte and SPU can handle data block of 16B, it should be adapted for SIMD process.

```
1 index % time  self  children   called     name
2                                              <spontaneous>
3 [1]     100.0    0.00     1.51                main [1]
```

```
                0.00    1.50      11/11          pbkdf2_sha1 [2]
                0.00    0.01      11/11          wpa_pmk_to_ptk [13]
                0.00    0.00       2/35          os_strlen [12]
                0.00    0.00      11/11          wpa_eapol_key_mic [15]
                0.00    0.00      11/1262415     os_memset [11]
                0.00    0.00      23/45          os_memcmp [19]
                0.00    0.00      11/11          next_pswd [20]
-----------------------------------------------
                0.00    1.50      11/11          main [1]
[2]     99.6    0.00    1.50      11           pbkdf2_sha1 [2]
                0.06    1.44      22/22          pbkdf2_sha1_f [3]
                0.00    0.00      22/8024929     os_memcpy [10]
-----------------------------------------------
                0.06    1.44      22/22          pbkdf2_sha1 [2]
[3]     99.6    0.06    1.44      22           pbkdf2_sha1_f [3]
                0.01    1.41   90090/90101       hmac_sha1 [4]
                0.01    0.00      22/35          os_strlen [12]
                0.01    0.00   90112/8024929     os_memcpy [10]
                0.00    0.00      22/90167       hmac_sha1_vector [5]
-----------------------------------------------
                0.00    0.00      11/90101       wpa_eapol_key_mic [15]
                0.01    1.41   90090/90101       pbkdf2_sha1_f [3]
[4]     94.6    0.01    1.41   90101           hmac_sha1 [4]
                0.05    1.36   90101/90167       hmac_sha1_vector [5]
-----------------------------------------------
                0.00    0.00      22/90167       pbkdf2_sha1_f [3]
                0.00    0.00      44/90167       sha1_prf [14]
                0.05    1.36   90101/90167       hmac_sha1 [4]
[5]     93.7    0.05    1.36   90167           hmac_sha1_vector [5]
                0.01    1.32  180334/180334      sha1_vector [6]
                0.03    0.00  180334/1262415     os_memset [11]
                0.01    0.00  180334/8024929     os_memcpy [10]
-----------------------------------------------
                0.01    1.32  180334/180334      hmac_sha1_vector [5]
[6]     87.8    0.01    1.32  180334           sha1_vector [6]
                0.07    1.19  180334/180334      SHA1Final [7]
                0.01    0.05  360778/7032927     SHA1Update [8]
                0.00    0.00  180334/180334      SHA1Init [18]
-----------------------------------------------
                0.07    1.19  180334/180334      sha1_vector [6]
[7]     83.6    0.07    1.19  180334           SHA1Final [7]
                0.14    0.94 6672149/7032927     SHA1Update [8]
                0.11    0.00  721336/1262415     os_memset [11]
```

```
47 ------------------------------------------------
48                 0.01     0.05  360778/7032927    sha1_vector [6]
49                 0.14     0.94 6672149/7032927    SHA1Final [7]
50 [8]      75.3   0.15     0.99 7032927          SHA1Update [8]
51                 0.49     0.08  360734/360734     SHA1Transform [9]
52                 0.42     0.00 7393661/8024929    os_memcpy [10]
53 ------------------------------------------------
54                 0.49     0.08  360734/360734     SHA1Update [8]
55 [9]      37.6   0.49     0.08  360734          SHA1Transform [9]
56                 0.06     0.00  360734/1262415    os_memset [11]
57                 0.02     0.00  360734/8024929    os_memcpy [10]
58 ------------------------------------------------
59                 0.00     0.00      11/8024929    sha1_prf [14]
60                 0.00     0.00      11/8024929    wpa_eapol_key_mic [15]
61                 0.00     0.00      22/8024929    pbkdf2_sha1 [2]
62                 0.00     0.00      44/8024929    wpa_pmk_to_ptk [13]
63                 0.01     0.00   90112/8024929    pbkdf2_sha1_f [3]
64                 0.01     0.00  180334/8024929    hmac_sha1_vector [5]
65                 0.02     0.00  360734/8024929    SHA1Transform [9]
66                 0.42     0.00 7393661/8024929    SHA1Update [8]
67 [10]     30.1   0.46     0.00 8024929          os_memcpy [10]
68 ------------------------------------------------
69                 0.00     0.00      11/1262415    main [1]
70                 0.03     0.00  180334/1262415    hmac_sha1_vector [5]
71                 0.06     0.00  360734/1262415    SHA1Transform [9]
72                 0.11     0.00  721336/1262415    SHA1Final [7]
73 [11]     13.2   0.20     0.00 1262415          os_memset [11]
74 ------------------------------------------------
75                 0.00     0.00       2/35         main [1]
76                 0.00     0.00      11/35         sha1_prf [14]
77                 0.01     0.00      22/35         pbkdf2_sha1_f [3]
78 [12]      1.0   0.01     0.00      35          os_strlen [12]
79 ------------------------------------------------
80                 0.00     0.01      11/11         main [1]
81 [13]      0.4   0.00     0.01      11          wpa_pmk_to_ptk [13]
82                 0.00     0.01      11/11         sha1_prf [14]
83                 0.00     0.00      44/8024929    os_memcpy [10]
84                 0.00     0.00      22/45         os_memcmp [19]
85 ------------------------------------------------
86                 0.00     0.01      11/11         wpa_pmk_to_ptk [13]
87 [14]      0.4   0.00     0.01      11          sha1_prf [14]
88                 0.00     0.00      11/35         os_strlen [12]
89                 0.00     0.00      44/90167      hmac_sha1_vector [5]
```

```
90                  0.00    0.00       11/8024929     os_memcpy [10]
91  -----------------------------------------------
92                  0.00    0.00       11/11          main [1]
93  [15]     0.0    0.00    0.00       11          wpa_eapol_key_mic [15]
94                  0.00    0.00       11/90101       hmac_sha1 [4]
95                  0.00    0.00       11/8024929     os_memcpy [10]
96  -----------------------------------------------
97                  0.00    0.00  180334/180334     sha1_vector [6]
98  [18]     0.0    0.00    0.00  180334          SHA1Init [18]
99  -----------------------------------------------
100                 0.00    0.00       22/45          wpa_pmk_to_ptk [13]
101                 0.00    0.00       23/45          main [1]
102 [19]     0.0    0.00    0.00       45          os_memcmp [19]
103 -----------------------------------------------
104                 0.00    0.00       11/11          main [1]
105 [20]     0.0    0.00    0.00       11          next_pswd [20]
106 -----------------------------------------------
```

Listing 4.1: The output of gprof profiler for WPA password cracker. It is visible, that function *hmac_sha1_vector* is bottle neck.

4.3 Parallelisation

Because the whole code after compiling has tens of kB and dynamic arrays are used rarely, than it is possible to implement the whole code to one SPU to Local Store (LS). And knowledge, that there is one function which consumes almost whole run time. It leads to accept the concept of tasks dividing called Parallel stages, which is in figure 3.4. Further exploring oh the HMAC-SHA1 function has been done. The HMAC-SHA1 transform looks like:
SHA1(K XOR opad, SHA1(K XOR ipad, text)), where K is an n byte key, ipad is the byte 0x36 repeated 64 times, opad is the byte 0x5c repeated 64 times and text is the data being protected [34]. The SHA1 is cryptography hash algorithms. SHA1 cut the input data to the blocks of 512b (64B). The last block is filled, aligned including adding the information about length(for this is reserved last 64b). The output of SHA1 has 160b (20B).

Chapter 5

Implementation of Algorithm

In this chapter is described the process of implementation to Cell BE. The first step is to fit the code to PPU and test the functionality. Afterwards the code is modified to run on SPU. Some investigation and suggestions to increase the number of password tempts has been done. Especially the *hmac_sha1_vector* function was adapted to speed up the whole execution. The next step is execute the WPA password cracker on all available SPUs simultaneously.

5.1 Implementation on PPU

The easiest first step is just to try compile the code for PC on PPU. The code for CPU was compiled without problems, but the results was incorrect. It took some time to realize, that the problem is big-endian. There are two main approaches how to store data in registers and memory. They are varying in position of the MSB(Most Important Bit) and LSB(Least Important Bit) in memory. And the systems according to system of storing are called big-endian and little-endian machines. In case of big-endian system, the most significant value in the sequence is stored at the lowest storage address (i.e., first). In a little-endian system, the least significant value in the sequence is stored first [27].
Most modern Computers, including PCs are little-endian systems. The PowerPC system is bi-endian, what means, that system can be switched between both modes. But the whole concept of open WPA supplicant is to be as much as flexible to all architectures. Therefore in main module is possible to switch the between parts of codes written for big-endian and little-endian. The preprocessor directives are used for this switching.

When the endians problem was solved, the code functionality was tested. The output was correct for both versions of the hash functions (MD5 and SHA1) and because it is not planed to use WPA password cracker on PPU because PPU has to run OS and control SPUs, another modifications of code for PPU was not realized.

5.2 Implementation on SPU

The PPU code was compiled correctly, and because the SPUs are little-endian, even the output was correct. It was first mile stone, WPA password cracker on SPU. Because the concept of task dividing called Parallel stage was chosen (in figure 3.4) and whole code will be executed on each available SPU, the development and adapting of the code can be performed on one SPU. It is huge advantage, because implementation on one SPU will avoid to problem of synchronisation and data passing between SPUs and PPU.
The first discovered part of the code, which should be changed was in function *wpa_pmk_to_ptk*. The pointers to both Nonces and MAC addresses are passed to the function and then they are sorted by values and joined to one variable **data**. Because the MAC addresses and Nonces are constant, therefore is enough to sort them and join them only once on start-up. The last parameter of this function is size of the key. Whole PTK (pairwise transient key) has 64B, but for calculating of the MIC(Message Integrity Check) just the first 16B of PTK is enough. Therefore we can set the last parameter to 16. Then the rest of PTK will be random, but first 16B representing KCK will be correct. It is because the calculating is like stream and first 16B are all, what is required for calculating MIC.
But the most important function is HMAC-SHA1. This function is defined as SHA1(K XOR opad, SHA1(K XOR ipad, text)), where K is an n byte key, ipad is the byte 0x36 repeated 64 times, opad is the byte 0x5c repeated 64 times and text is the data being protected . The SHA1 is cryptography hash algorithms. SHA1 cut the input data to the blocks of 512b (64B). The last block is filled, aligned including adding the information about length(for this is reserved last 64b). The output of SHA1 has 160b (20B). Implementation is via three SHA1 functions - Init, Update and Final - in figure 4.3. The init part just fill the variables with initialisation constants. The most important part is Update. This part is executed many times. The Diagram of one execution is in figure 5.1. After some time of investigating the code, was discovered recently released library from IBM called libspucrypto, which contains support for many crypto functions like AES, MD5 and SHA. Whole library is on he enclosed CD in the folder \C_codeCellBE\libcrypto. Unfortunately library is released only as binary file, therefore is impossible to see the source code.

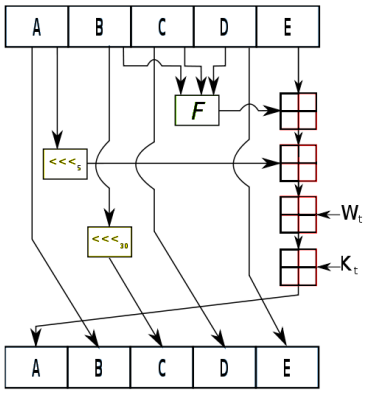

Figure 5.1: One iteration within the SHA1, figure adopted from [36].

According to the file sha.h was chosen following functions : *SHA1_Init*, *SHA1_Update_fast* and *SHA1_Final*. After implementation of the SHA1 with libspucrypto was changed only function *sha1_vector*. This function is represented in listing 5.1. Important variable here is pointer to length with name **len**. It represent length of the input data buffer. The function *SHA1_Update_fast* process data aligned to 64B or multiple of 64. The rest, data block smaller than 64B have to pass via *SHA1_Update*. When the libspucrypto SHA1 had been implemented, it was expected reasonable increasing of psk tempts per second. But this implementation optimised only part of SHA1, just function *SHA1_Update_fast*.

```
1  void sha1_vector(size_t num_elem, const u8 *addr[], const size_t *len,
2          u8 *mac)
3  {
4      int lr;
5      SHA_CTX ctx;
6      size_t i, offset;
7      lr = SHA1_Init(&ctx);
8
9      for (i = 0; i < num_elem; i++){
10         if (len[i] >= 64){
11             offset = len[i] - (len[i] % 64) ;
12             lr = SHA1_Update_fast(&ctx, (unsigned char*) addr[i], offset);
13             if (lr !=1)
14                 lr = SHA1_Update(&ctx, (unsigned char*) addr[i], offset);
15             lr = SHA1_Update(&ctx, addr[i]+offset, len[i]-offset);
16         }
17         else
18             lr = SHA1_Update(&ctx, addr[i], len[i]);
19     }
20
```

```
21      lr = SHA1_Final(mac, &ctx);
22
23  }
```

Listing 5.1: The modified function *sha1_vector*, now calls SHA Init, Update and Final from library libspucryfto from IBM.

The next discovered bottleneck are kernel functions to handle memory - copy and set. They offers possibility to rewrite for vector processing, because general implementation is independent on the platform and handle byte by byte. The first proposal was to divide data to the 16B blocks and handle them via SIMD function and the rest what is not aligned to 16B process byte by byte. After rewriting of copy function was founded the project Newlib [20]. "*Newlib is a C library intended for use on embedded systems. It is a conglomeration of several library parts, all under free software licenses that make them easily usable on embedded products*[20]."
The adapted code is on enclosed CD in folder \C_code\CellBE\pswd_spu.

5.3 Implementation on Multiple SPU

To expand WPA password cracking on all available SPUs, there have to be some control process on the PPU, thread controller. The model of tasks dividing was already chosen - Parallel Stages Model (block diagram in figure 3.4. To avoid situation, when more SPUs crunching the same passphrases, the controller will determine what passphrases each SPU will test. The SPUs tasks are independent on each other, therefore asynchronous executing is not only possible, but wanted. It allows to avoid to the situation, when an SPU crunch data faster then other and has to wait to the new synchronous start.
The designed thread controller have to:

1. On the beginning sends initial data as SSID, MACs, MIC, etc. and first set of passphrases to procces.
2. Wait until any available SPU, then sends new passphrases to process.
3. Wait to successfully tested passphare.

The IBM second implementation of Runtime Management Library uses for multithreading the POSIX threads. Therefore the possibilities of threads was explored and created multi

thread framework, what is C code handling threads and is possible to expand it by concrete SPU and PPU code. On the CD it is located in folder \C_code\multi_pthreads_frame.
According to the articles about POSIX threads (pthreads) there are a set of functions to handle them [30]. There are two basic functions to create thread *pthread_create* and to destroy thread *pthread_join*, which check if the thread is still running and close it. When this thread is still active, then waits and close it. This function *pthread_join* is blocking, because waits until thread terminate. But according to the requirements the thread controller have to execute new SPU thread immediately after some terminate. And there is no function, which just check the status and do not block. Therefore multi thread frame was designed. In case, that six threads is required (as number of available SPUs), then is created six control threads. Each control thread creates SPU thread. And now, when the function *pthread_join* is called before the SPU terminate the control thread waits till SPU terminate.
Each SPU thread have to send set of passphrases to process. To avoid processing one passphrase multiple times, the main PPU thread use the same generator of passphrases as SPUs and sends just the first passphrase to SPU, the number of passphrases to process is constant. The variable **global_passphrase** is saved in the main memory. When control thread calls SPU, then just pass the variable **global_passphrase** to SPU and increment it with generator so many times as constant number of passphrases, then save the new **global_passphrase** back. To avoid simultaneous access to variable **global_passphrase** by two control threads, there is used mutex. The mutex is additional variable, which provides access to **global_passphrase** to only one control thread in the time.
To pass the first passphrase to the SPU is used DMA. The DMA transfers are effective and fast. The DMA is only one possible way to access main memory, because the SPU can not access main memory directly. There is library spu_mfcio.h, which provides functions to handle DMA. Each SPU contains MFC (Memory Flow Controller) and can provides up to 16 simultaneous transfers. In this application is transferred just passphare, therefore all the great features of DMA will not be used. The base functions are the *mfc_get* and *mfc_put*. The *mfc_get* function read data from main memory and store them in LS. The *mfc_put* write data to main memory. The first one is used on the beginning to pass constant data (MACs, SSID, Nonces, etc.) to SPU. Lately is used just for delivering new passphrase to SPU. The *mfc_get* function is used, when the correct passphrase is found. Then is the information given back to PPU thread by DMA.
The multi thread controller was implemented in PPU code and DMA handling was added to both codes - for PPU and SPU. The core of SPU code remains without changes.

5.4 Benchmarks

During the whole process of implementation was measured, how fast the code is. In this kind of applications, which crunches all possible passwords in hope to find the correct one is only one criteria to measure - the number of tested passwords per time. There will be used number of tested passphrases per second - therefore artificial unit *psk/sec*.
Measure on the PC was realized with addition of the time functions. The time was stored on the beginning of password cracking and on the end. The same approach was used for PPU, but there are no time function for SPU. Therefore was used counter to count number of machine cycles. The value of cycles has to be divided by the frequency of core. All SPU runs at the same frequency as PPU - 3.2 GHz. Then was the values divided by number of tested passphrases.
Measurement was performed on PC (Core2 Duo@1.5 GHz; 2048 MB RAM; OS: MS WinXP SP2) just after implementation and code was not adapted at all. The next measurements were performed on PPU and SPU. On the SPU was measured before adapting and after implementation of libspucrypto SHA1 and kernel mem functions. And final measurement was with using all six available SPUs The whole numbers are in table 5.2. The graph demonstration follows in figure 5.3.

Measured speed of WPA password cracker:	
Platform	psk/sec
PC (Core2@1,5 GHz)	6,88
PPU	9,6
SPU not optimised	9,11
SPU optimised	44,19
6 SPUs optimised	263,14

Figure 5.2: The measured speeds of password cracking table.

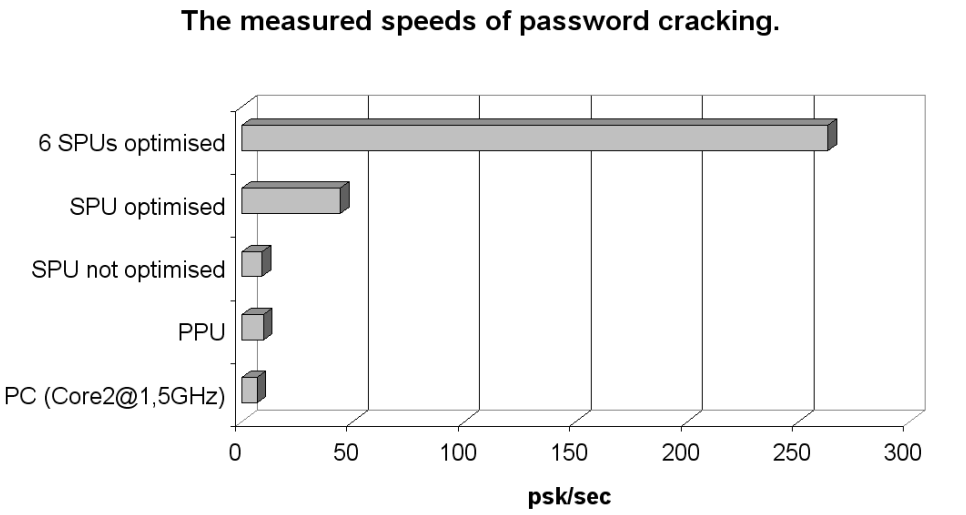

Figure 5.3: The measured speeds of password cracking graph.

Chapter 6

SSE2 Implementation

This chapter describe the implementation based on SSE (Streaming SIMD Extensions). The same computer from last chapter can perform up to 90 *psk/sec* when is used SSE. The cracking tool aircrack-ng contains the implementation using the SSE and the open source code of implementation SHA1 is available [24]. It was rewritten for SPU and tested.

6.1 SSE2

The SSE2 stands for Streaming SIMD Extensions version 2. Some processors extends the instruction set for processing multimedia faster. It brings new instructions and registers. The first was MMX (MultiMedia eXtensions). It was released in 1997. It brings new eight 64 bit registers and SIMD instructions, which can process data in these registers. Then in 1999 came SSE with new eight 128 bit registers. And finally in 2001 was released first CPU with SSE2 extension. SSE2 extend the MMX and allows to perform MMX instruction on 128 bit wide registers.
Programming of the SSE2 use Assembly language - ASM. It is the lowest level of programming [24].

6.2 Implementation of SSE2 code

The open source code of SHA1 for SSE2 is the part of aircrack-ng tool. The source code is available on the website: http://svn.dd-wrt.com:8000/ddwrt/browser/src/router/aircrack-ng/src/sha1sse2.h?rev=11606 . The code has different structure than C language. But SSE2 contains 128b registers and uses the SIMD instruction too. On the following listing 6.1 is example of ASM and corresponding C code for SPU. Between lines number 4 and 9 is declaration of variable and the rest is macro calculating some F0. On line 21 is variable declaration and from line 23 is definition of macro F0, but rewritten for the SPU. The whole code is on CD in folder \C_code\CellBE \sh1-sse2 .
The SSE code contains SHA1 with already known parts - Init, Update and Final. The new implementation was ported to SPU code and replaced SHA1. The rest of SPU code remains.

```
1  // Assembly code
2
3  .data
4  .align(16)
5  const_init_a:
6  .long 0x67452301
7  .long 0x67452301
8  .long 0x67452301
9  .long 0x67452301
10
11 #define F0(x,y,z)          \
12         movdqa  x, tmp2; \
13         movdqa  x, tmp1; \
14         pand   y, tmp2;     \
15         pandn z, tmp1;     \
16         por    tmp2, tmp1;
17
18
19 // SPU Intrinsics code
20
21 const vector unsigned int init_a = {0x67452301, 0x67452301, 0x67452301, 0
       x67452301};
22
23 #define F0(x,y,z)                                    \
24         tmp2 = x;                                    \
25         tmp1 = x;                                    \
26         tmp2 = spu_and(tmp2,y);                      \
```

```
27        tmp1 = spu_and(tmp1 = spu_nor(tmp1,tmp1),z); \
28        tmp1 = spu_or(tmp1, tmp2);
```

Listing 6.1: The part of source code with SSE2 and corresponding rewritten part for SPU Intrinsic.

6.3 Benchmark

The new code was benchmarked on one SPU and on all six ones. The table is also expanded by the value measured on PC with SSE2 support. The speeds increased tremendously.

Measured speed of WPA password cracker:	
Platform	psk/sec
PC (Core2@1,5 GHz)	6,88
PC (@1,5 GHz) + SSE2	88
PPU	9,6
SPU not optimised	9,11
SPU optimised	44,19
6 SPUs optimised	263,14
SPUs optimised SSE	121,54
6SPUs optimised SSE	701,89

Figure 6.1: The measured speeds of password cracking table with SSE2 Implementation.

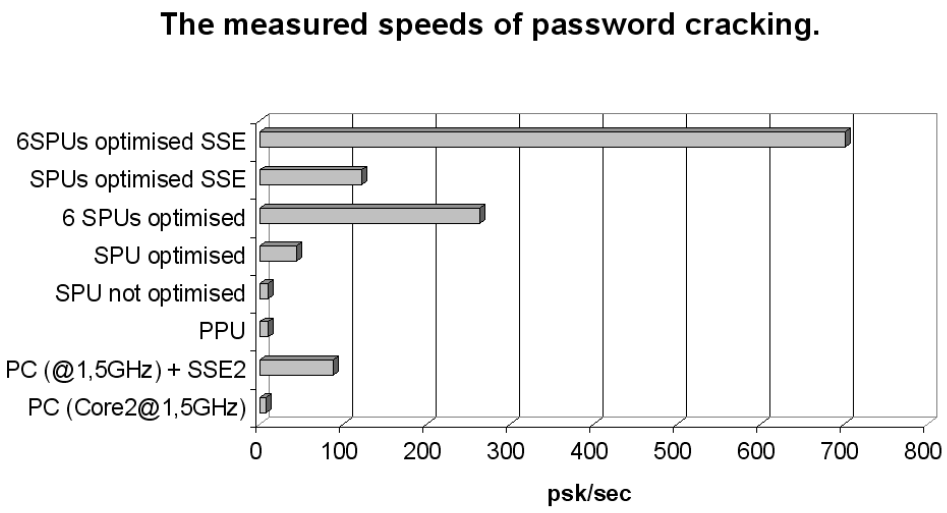

Figure 6.2: The measured speeds of password cracking graph with SSE2 Implementation.

Chapter 7

Conclusion and Future Work

In this chapter the individual parts and individual implementations are evaluated. The problem in this project was to design a WPA password cracker on the Cell BE architecture. The WPA security standard offers many variations, but only one known weak point. The weakness point was investigated together with standards in WiFi. Another area to investigate was password cracking with possible approaches and methods of implementation.

7.1 Preparing for Implementation

The preparing for implementation on the Cell BE contained: investigated problem, standards, possible solutions and designed the functional PC code. The investigation led to decision of using brute force password cracking method with simulating of the important part of WPA weak point - 4 Way handshake authentication. Then possibilities of implementation 4-Way handshake was explored. As the best entry point was assumed open source code WPA supplement. Then begun the process of implementing of 4-way handshake on PC. The code had been properly investigated and the usable parts had been separated. The problem with entry data - test vector - had to be solved, therefore the testing connection was established and with using of appropriate software tools were captured handshakes. Because of that was installed OS Linux with aircrack-ng and airodump-ng.
The modularity of open source code WPA supplement offered possibility to divide whole application to several parts. The firs part represents passphrase generator. It is small part,

which subsequently generates all possible passphrase. The next block derive the master key from generated passphrase. This part of whole code consumes the most resources. To derivate key is used Password-Based Key Derivation Function (PBKDF2) which uses HMAC-SHA1 hash algorithm. The derived key is Pairwise Transient Key (PTK). As an input of the following block is used only part of the PTK. The first 128bits represents Key Confirmation Key. This separated part of PTK is used to calculating of Message Integrity Check (MIC). The MIC was captured together with handshakes and during transfer it is non-crypted, thus the calculated MIC is compared with captured one. In case, that they are equal then the generated passphrase is equal with used one.
When the each block worked perfectly, the implementation to another platform could begin. As the Cell Be platform was used PlayStation3. The platform and techniques of implementation was explored.

7.2 Implementation on Cell BE

Implementation on the PPU was the first step. It is recommended and quite reasonable approach of developing. The performance analysis of the code and localisation of the most consuming parts is unavoidable step for design the concept of tasks dividing and degree of parallel processing.
The transfer of code from PC to PPu was seamless, except the inconvenience with little/big endianness. The PPU is primary for OS and controlling task, which handling the using of SPUs. Using of Parallel Stage Model for tasks dividing was already decided by profiling. Therefore the code was not modified for PPU.
The first step of implementation on SPU was just try to compile the code for SPU. The program functionality was correct and further investigation of code could begin. There was made some minor changes on the code, but the most important modifications was happened in two main iterations - according to the development model. In the first iteration was modified SHA1 function with original IBM library spulibcrypto by replacing parts Init, Update and Final. And using of optimised kernel functions memcpy and memset. This first iteration led to speed up from 9.11 to 44.19 *psk/sec*, what is almost 5 times more. It is good to realise, that in terms of password cracking the improvement ”5 times” is tremendous, because it is a huge difference to cracking some system one hour or five hours.
To increase the speed of cracking the rest of SPUs was involved. This part of implemen-

tation had not any effect on the core of the WPA password cracker running on the SPU. This involves using of pthreads for parallel processing and DMA channels for exchanging data between PPU and SPUs. The increasing of the speed is quite linear to number of used SPUs. According to table 5.2 the speed was increased from 44.19 to 263.14 *psk/sec*, it means multiplying factor 5.95 and it really close to six, the actual increasing of resources. The second iteration of Domain shifting from Algorithm to Architecture Domain contains implementation of the SHA1 function in SSE2 manner. As the entry point of this implementation was used opensource SHA1-SSE2 code. The assembly language SSE2 code was rewritten with SPU intrinsic functions and ported to SPU code, where replace SHA1. This second implementation caused another speed increase from 44.19 to 121.54 *psk/sec* and it means multiplying factor 2.75. By involving the other SPUs is reached the 701.89 *psk/sec*. Again the dependence between increased number of SPUs and speed up is quite linear. In this case the multiplying factor is 5,78, again close to six.
Total speed up between non optimised code on SPU and cluster of six SPUs with optimised code in SSE2 manner is amazing. The speed increased from 9.11 to 701.89 *psk/sec*, it is almost 77 times faster.

7.3 Evaluation of Cell BE suitability

The finally implemented version reached 77 times faster processing of passwords than initially implemented code. In comparison with desktop CPU the final implementation reached 100 times faster processing and in comparison with SSE optimised PC code reached 8 times faster processing. Definitely the Cell BE offers better performance for data crunching tasks, than desktop CPUs. The Cell BE processors are even available as PCIe boards [32], what offers wide accessibility of platform. But he Cell BE processors are usually clustered. Many vendors offers some kind of Cell BE cluster. From light weight as ”8 Node PS3 Cluster” containing 8 80GB PlayStations connected via ethernet and runs Linux [10], through medium weight as ”14 QS22 BladeCenter Chassis” containing 14 IBM QS22 Cell blades (CPU PowerXCell 8i - new generation of Cell BE) with 8GB RAM per blade [10], up to heavy weight represented by Supercomputer Roadrunner in Los Alamos with 6562 AMD Opterons and 12240 Cell chips [17].
For example, if the optimised code is implemented on Roadrunner, just on Cells without involving Opterons, then it should crunch up to 12 millions *psk/sec* (just multiplied 121 *psk/sec* by number of SPUs - each Cell in Roadrunner has 8 SPUs).

But even 12 millions *psk/sec* is not enough. The passhprase can be up to 63 bytes long and use all visible characters (between code 32 and 126, thus 95 characters). Just the 4 characters long password with 95 possible characters per position has over 81 millions of combinations. Dependency between the length of password and possible combinations is exponential function with base 95 (can vary with range of used characters) and exponent equal to the length of password. Therefore the raw brute force is used rare.
The WPA passphrase cracking usually use some passwords dictionary. Then the speed of 700 *psk/sec* is pretty high. (Using of the dictionary will not significantly slow down password cracking, because the EIB (Element Interconnect Bus) offers bandwidth 25.6 GB/s and 700 passphrases with average length of 32B have total size less than 22kB.)
The above mention could be the answer to the project problem definition 1.2. How efficient is the Cell BE processor for WPA cracking? The efficiency of the Cell BE is really high, because it is 8 times faster, than processing on SSE2. Moreover the Cell BE is frequently used in clusters. The answer is definitely positive, because the Cell BE as platform offers much better performance than the ordinary desktop processors.

7.4 Further Development

The Cell BE is frequently used in clusters, many clusters are even build from bunch of PS3s. These clusters usually runs on Linux and some software interface connects all processors together. Therefore the ability to employ other PS3s should be implemented. It contains to add communication over IP stacks and promote one PS3 to a master. The master will control the tasks dividing, similar approach to dividing tasks on SPUs could be used for PS3s.
Nowadays the SDK do not support usage of graphic card in PS3. The used graphic card is some special edition of NVIDIA with 256 MB of video RAM. If this card supports the CUDA (Compute Unified Device Architecture), then should be considered the usage of GPU for WPA password cracking.
The dictionary based password cracking could be added. The PS3 has Blue-ray drive, which should be perfect to accommodate the large dictionary.

Bibliography

[1] Aircrack-ng, Team [2008], 'Aircrack-ng introduction'. Visited on 2008-11-01.
URL: *http://aircrack-ng.org/doku.php*

[2] Alliance, WiFi [2007], 'About the alliance'. Visited on 2008-10-13.
URL: *http://www.wi-fi.org/about_overview.php*

[3] Ashink [2007-12-11], 'Sony playstation used to crack password', *hacking truths* . Visited on 2008-10-13.
URL: *http://www.hungry-hackers.com/2007/12/sony-playstation-used-to-crack-password.html*

[4] Bader, David A. [2007], 'Web page of cell programming workshop'. Visited on 2008-12-03.
URL: *http://sti.cc.gatech.edu/programming.html*

[5] Bjerrum, Bo, Jes Toft Kristensen and Klaus Dahl Kristiansen [2007], 'Noise reduction for hands-free car phone'.
URL: *http://kom.aau.dk/group/07gr840/turnin/*

[6] Borisov, Nikita, Ian Goldberg and David Wagner [2001], Security of the wep algorithm. Visited on 2008-11-14.
URL: *http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html*

[7] Buttari, Alfredo, Piotr Luszczek, Jakub Kurzak, Jack Dongarra and George Bosilca [2007], *SCOP3 - A Rough Guide To Scientific Computing on the Playstation 3*, Innovative Computing Laboratory, University of Tennessee Knoxville. Visited on 2009-02-18.
URL: *http://www.netlib.org/utk/people/JackDongarra/PAPERS/scop3.pdf*

[8] Chabala, Greg [2007], 'How to install fedora core 6 on your playstation 3'. Visited on 2009-02-19.
URL: *http://gregchabala.com/computer/playstation3/howto-linux-on-ps3.php*

[9] Combs, Gerald [2008], 'About wireshark'. Visited on 2009-03-09.
URL: *http://www.wireshark.org/about.html*

[10] Fixstars [2008], '8 node ps3 cluster'. Visited on 2008-12-02.
URL: *http://us.fixstars.com/store/index.php?submit=software&submitimg[hardware][solutions]=1*

[11] Fleishman, Glenn [2008-11-06], 'Battered, but not broken: understanding the wpa crack', *ars technica* . Visited on 2008-12-18.
URL: *http://arstechnica.com/security/news/2008/11/wpa-cracked.ars*

[12] Fluhrer, Scott, Itsik Mantin and Adi Shamir [2001], Weaknesses in the key scheduling algorithm, *in* 'RC4, Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography', pp. 1–24.

[13] Gans, Joshua S., Stephen P. King and Julian Wright [2005], Wireless communications. Visited on 2008-11-01.
URL: *http://profile.nus.edu.sg/fass/ecsjkdw/WirelessCommunications_Final.pdf*

[14] George, Randy [2008-05-21], 'Lockheed breaks wpa-encrypted wireless network with 8 clustered sony playstations'. Visited on 2008-12-02.
URL: *http://www.networkcomputing.com/blog/dailyblog/archives/2008/05/lockheed_breaks.html#more*

[15] Hulton, David [2007], 'Hacking the airwaves with fpgas'. Visited on 2009-02-17.
URL: *http://openciphers.sourceforge.net/slides/shmoocon-2007.pdf*

[16] IBM [2008], 'Cell broadband engine technology'. Visited on 2009-01-21.
URL: *http://www-03.ibm.com/technology/cell/*

[17] IBM [2009], 'Los alamos national lab's roadrunner'. Visited on 2009-06-10.
URL: *http://www.ibm.com/ibm/ideasfromibm/us/roadrunner/20080609/index.shtml*

[18] IBM, Systems and Technology Group [2007], 'Cell broadband engine programming handbook'. Version 1.1.
URL: *http://www.ibm.com/chips/techlib/techlib.nsf/techdocs/*

[19] Itsik, Martin [2002], 'Rc4'. Visited on 2008-11-25.
URL: *http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html*

[20] Johnston, Jeff [2009], 'The newlib project'. Visited on 2009-02-01.
URL: *http://sourceware.org/newlib/*

[21] Kramer, Matt [2008-04-16], 'Lockheed martin opens wireless cyber security lab'. Visited on 2008-12-02.
URL: *http://www.reuters.com/article/pressRelease/idUS172500+16-Apr-2008+PRN20080416*

[22] Lal, Shradha and Shabanpreet Kaur Warar [2007], Sdr implementation of a multi-carrier transmitter with link adaptation, Technical report, Aalborg University, The Faculty of Engineering, Science and Medicine.

[23] Lehembre, Guillaume [2006], 'Bezpecnost wi-fi - wep, wpa a wpa2', *hakin9.org* . Visited on 2008-11-10.
URL: *www.hsc.fr/ressources/articles/hakin9_wifi/hakin9_wifi_CZ.pdf*

[24] Marcos [2009], 'Aircrack-ng wpa optimalizace pro sse2 procesory'. Visited on 2009-04-19.
URL: *http://airdump.cz/aircrackng-sse2-procesor/*

[25] Martin, Chris [2008-10-23], 'Wi-fi security - how to secure your wi-fi network'. Visited on 2008-10-28.
URL: *http://www.aboutonlinetips.com/wi-fi-security-how-to-secure-your-wi-fi-network/*

[26] Michal, Blazek [2007], Implementace skrytych markovovskych modelu na procesoru cell, Technical report, Czech Technical University in Prague, The Faculty of Electrical Engineering.

[27] Peterka, Jiri [1992], 'Little endian vs. big endian'. Visited on 2009-04-15.
URL: *http://www.earchiv.cz/a92/a237c120.php3*

[28] Puzmanova, Rita [2007], 'Bezpecnost wifi zalezi jen na vas', *Lupa* . Visited on 2008-11-17.
URL: *http://www.lupa.cz/clanky/bezpecnost-wifi-zalezi-jen-na-vas/*

[29] Reed, Brad [2008-05-19], 'Inside lockheed martin's wireless security lab'. Visited on 2008-12-02.

URL: *http://www.networkworld.com/news/2008/051908-lockheed-martin-wireless-security-lab.html*

[30] Robbins, Daniel [2000], 'Posix threads explained'. Visited on 2009-02-5.
URL: *http://www.ibm.com/developerworks/linux/library/l-posix1.html*

[31] Simonsen, Peter Augustus and Jes Toft Kristensen [2007], Ds-cdma procedures with cell broadband engine, Technical report, Aalborg University, The Faculty of Engineering, Science and Medicine.

[32] Systems, Mercury Computer [2006], 'Maly superpocitac s procesorem cell be do pcie x16 slotu'. Visited on 2009-03-18.
URL: *http://www.cdr.cz/a/18155*

[33] Vasicek, Ondrej [2007-05-09], Wep - zabezpeceni site wifi, Technical report, Czech Technical University in Prague, The Faculty of Electrical Engineering. Visited on 2008-10-15.
URL: *http://radio.feld.cvut.cz/personal/mikulak/MK/MK07_semestralky/wi-fi_zabezpeceni_wep.pdf*

[34] Vondruska, Pavel [2002], 'Crypto-world'. Visited on 2009-03-15.
URL: *http://crypto-world.info/casop4/crypto78_02.pdf*

[35] Vyroubal, Michal [2008], 'Wi-fi protected access'. Visited on 2008-11-25.
URL: *http://wi-fi.unas.cz/wpa.php*

[36] Wikipedia [2002], 'Sha hash functions'. Visited on 2009-05-11.
URL: *http://en.wikipedia.org/wiki/SHA_hash_functions*