Password Cracking

Software and Hardware Comparison

Seminar Paper

Ausgewählte Kapitel der IT-Security

Submitted by:

[redacted]

Student Identification:

[redacted]

Submission Date:

08.01.2020

Abstract

The dilemma with *recovering* a password is the uncertainty about the amount of time to conclude. Choosing the wrong hardware or software can drastically affect the time required to complete the process and, thus, the feasibility of the project. This paper deals with the fundamental questions, which setup should be used when testing authentication mechanisms and passwords. For this purpose, the renowned programs *Hashcat* and *John the Ripper* have been deployed on various platforms, ranging from embedded devices and Smartphones over Laptops and Desktops to Servers, to gather benchmarks, which are subsequently processed and compared. In this case, the factors operating system, testing tool, and version as well as platform and processor play a determining role in deciding which combination is best suited for a specific situation.

Even though, the results show that generally, Graphics Processing Unit calculations are much faster than Central Processing Unit calculations. It can also be said that Hashcat, which specializes in Graphics Processing Units, is not very easy to get it up and running on any processor, whereas John the Ripper runs on almost every Central Processing Unit and Operating System. Even though some hardware has proven to be effective by the community, no general statement shall be made, and it should always be decided on a case-by-case basis, which setup suites best for a distinct purpose or any existing hardware.

Keywords: Password cracking, Password Attacks, Hashcat, John the Ripper, Benchmarking, Penetration Testing Tools

Abbreviations

2FA	**T**wo-**F**actor **A**uthentification
ADB	**A**ndroid **D**ebug **B**ridge
aka	**a**lso **k**nown **a**s
APT	**A**dvanced **P**ackage **T**ool
ARM	**A**dvanced **R**ISC **M**achine
AP	**A**ccess **P**oint
AVX	**A**dvanced **V**ector **E**xtensions
BF	**B**rute-**F**orce
CPU	**C**entral **P**rocessing **U**nit
CSS	**C**ascading **S**tyle **S**heet
DB	**D**ata**b**ase
DSP	**D**igital **S**ignal **P**rocessor
eg	**e**xempli **g**ratia (for the sake of an example)
ERD	**E**ntity **R**elation **D**iagram
FMS	**F**luhrer, **M**antin and **S**hamir
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
GPU	**G**raphical **P**rocessing **U**nit
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
h/s	**H**ash calculated per **S**econd
John	**J**ohn **t**he **R**ipper
JtR	**J**ohn **t**he **R**ipper
MFA	**M**ulti-**F**actor **A**uthentification
MFR	**M**anu**f**acture**r**
MIPS	**M**icroprocessor without **I**nterlocked **P**ipelined **S**tages
OMP	**O**pen**MP** Threads
OpenCL	**Open C**omputing **L**anguage
PHP	**H**ypertext **P**re**P**rocessor
PSK	**P**re-**S**hared **K**ey
PTW	**P**ychkine-**T**ews-**W**einmann
RISC	**R**educed **I**nstruction **S**et **C**omputer
SQL	**S**tandardized **Q**uery **L**anguage
SSE	**S**treaming **S**IMD **E**xtensions
WEP	**W**ired **E**quivalent **P**rivacy
WPA	**W**i-Fi **P**rotected **A**ccess

Contents

1. Introduction

Password cracking or the more general expression of *Key recovery* describes the process that is used to *find* a key to a known digital identity and is usually associated with high computing power requirements in order to succeed in a realistic amount of time. Since not everybody has a large budget to be able to afford the most powerful hardware, the hardware in their possession must be used most efficiently. Be it only in the context of scientific work or small projects to analyze the security of a system or mechanism, this work serves as a guide for the average end-user with limited budget who focus on owned hardware to avoid further purchases and maximize the otherwise ineffectively or unused computing power.

1.1. Problem Statement

It is generally claimed that Hashcat is more performant than John the Ripper (john) and that Graphical Processing Units (GPU) can calculate more hashes per second than Central Processing Units (CPU). These statements are critically examined here and analyzed on a case-by-case basis to determine whether they apply to all cases and in which they do not. In order to be able to make statements, case data is required. This case data includes the type of tool and version, information about the operating system (OS), the used hardware platform as a whole as well as on single processors, and the comparison between pre-compiled and self-compiled Hashcat and John binaries. These case data were collected in Chapter 5 and analysed in Chapter 6.

1.2. Outline

This document gives a brief overview of common password attack techniques and tools but does not provide information about optimal or detailed usage. Here mainly benchmarks were performed to analyze the performance of the tools John the Ripper and Hashcat on given hardware. More specific processors supported by Hashcat, such as Digital Signal Processors (DSP) or Field Programmable Gate Array (FPGA) were not investigated, but are supported. For information on the execution of specific attacks, refer to the developers' websites, which provide great documentation, examples, and additional help. For further guidance consider for instance the *HASH CRACK: Password Cracking Manual* [B1]. Besides, information about most penetration testing tools from various niches of the security and forensics fields can always be obtained from the Kali Linux knowledge-base [W2]. This site aims to list them all and provide a quick reference to these tools.

2. Background

This chapter provides the required knowledge in the field of *key recovery*. It gives the reader a brief introduction to password cracking in general. It also introduces common password attacks and cracking tools. The chapter then goes into more detail about the popular tools *John the Ripper* and *Hashcat*, which are examined in depth in this case study.

2.1. Password Cracking

Password cracking or *Password recovery* is used in *IT-Security* and *Cryptanalysis* to recover passwords or digital keys in general, which are the most critical part of a digital identity that is associated with a specific service. To acquire such digital keys, common *Password Attacks* from Section [2.2] are employed to infer the corresponding key. Other techniques, like social engineering or dumpster diving, are also possible in some scenarios. Keys are unspaced sets of characters that allow to authenticate against a given security mechanism in order to gain access to data, programs, or secured systems. However, these techniques could also be used to gain unauthorized access to a remote system.

Once a key has been recovered, it can subsequently be used to undermine a digital authentication process, which provides privacy protection by mitigating risks of unauthorized access to individuals' information. In other words, a key must be kept secret by a *claimant*, humanoid or not, while the *verifier* confirms the claimant's identity. Additionally, choosing a strong password is crucial to prevent cracking and render recovery impossible. Furthermore, a Multi-factor (MFA) or Two-factor authentication (2FA) is recommended, since a claimant has to provide more factors to the verifier than just one key, which are something the claimant knows (*knowledge*), has (*possession*) or is (*inherence*), whereby a password recovery becomes pointless without knowing of the factors *possession* or *inherence*. [P1]

It should be noted that keys can be available in various forms. On the one hand, clear-text passwords can be transmitted over the network or stored locally, which then only needs to be intercepted or found. On the other hand, passwords can also appear in hashed or obfuscated forms. The latter differs in that the obfuscated password is brought back to its original form with the appropriate reverse algorithm. In contrast, a hashed password benefits from the one-way properties of the algorithm used and can only be concluded by hashing guessed password repeatedly and comparing them to the target hash until the original password or any rare hash collision has been found. In the following, only hashed keys are treated.

In order to be able to conclude on the original keys, high computing power is needed. Calculations can be done either by using CPUs with a small number of cores that have been developed for the sequential execution of processes. Alternatively, by using GPUs, which have a very high number of cores working in parallel, making them theoretically ideal for password cracking. The time to crack a password depends on its entropy, measured by the bit strength of a password and as well as the password length forming the target keyspace.

2.2. Password Attacks

Password attacks can be performed in two ways. On the one hand, in the case of *online attacks*, designated system interfaces are used to try out passwords and, on the other hand, *offline attacks* in which hashed passwords were obtained via other channels and then processed locally at any time. Online attacks are much slower because the calculations are performed by the target, while the latency of the network link also plays an important role. Furthermore, such attacks could be detected and prevented by the attacked systems. Examples would be attacks on SSH or any web-based login form. On the other hand, offline attacks can no longer be detected after the hashes have been obtained and can be carried out highly effective since only hashes are calculated and compared here, but no further communication or processes between systems or the like are required.

As it is common in most areas of life, there is not only black and white but also the grey zone in between. For this reason, it should be noted that there are also non-electronic attacks on passwords, such as dumpster diving or social engineering. But also more drastic but highly effective ways like Coercion by the government or *Rubber-hose cryptanalysis* (torture) can be taken, in which the human factor as the weakest link in the chain is attacked. This work is limited to offline attacks, and no human nor animal has been harmed during this work. However, some tools for online attacks are listed for completeness in section 2.5.

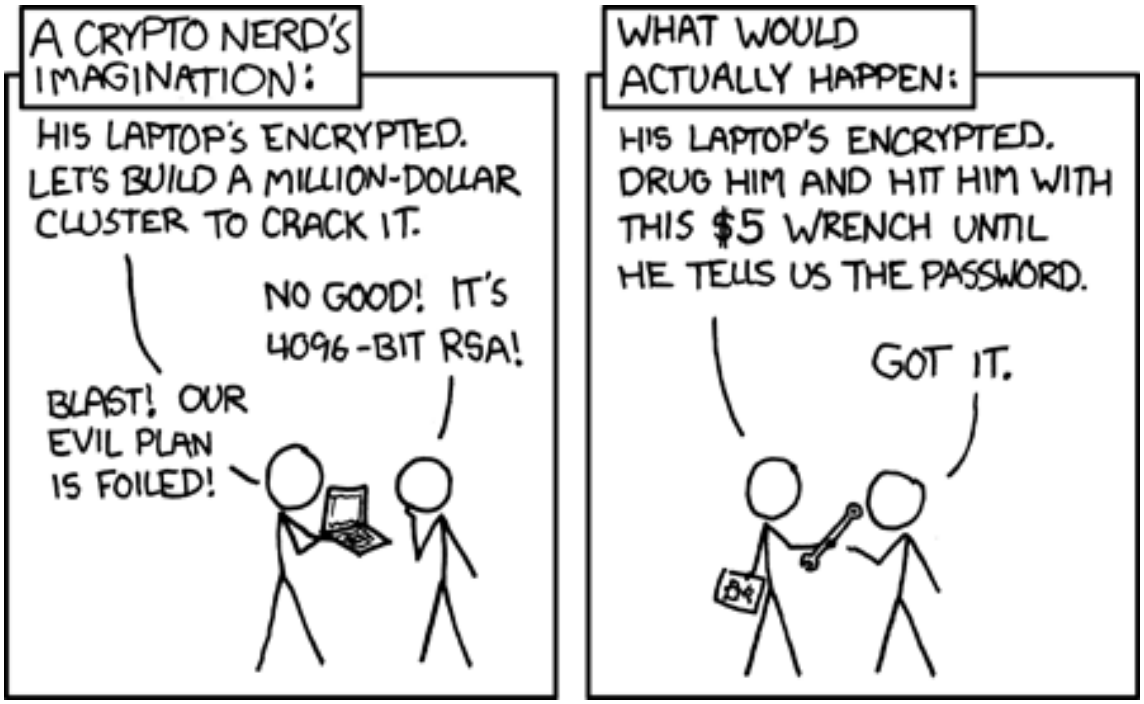

Figure 2.1.: Cracking skulls instead of passwords
`https://xkcd.com/538/` (CC BY-NC 2.5)

The following paragraphs present the most common modes used with password attacks.

Brute-force attacks try all possible passwords until the right one is found. Such password crackers generate and hash in the worst case every possible password in the keyspace, while none of these intermediate results are stored. Every hashed password is compared to the target hash on the fly until a match, and thus also a valid password has been found. This may be the original password or any rare hash collision. The success probability is given if the target password is in the known keyspace. However, for this attack, an extremely high computing power is needed to reach the target in a reasonable time. The Report [R1] from IBM Security deals with Footprinting, the process of gathering information about a target system, and brute force attacks on how to protect against these.

Dictionary attacks are a special kind of a brute force attack where the keyspace is restricted by using a dictionary (aka. wordlist). These wordlists usually contain common passwords grouped by specific criteria, which are then processed by crackers in sequence. The advantage is that one can quickly cover the most common passwords, so that these kinds of attacks are very effective, as many users use common weak passwords which are easy to remember. This can easily be fought by using passwords without underlying semantics. The best-known wordlist is the *rockyou.txt*, which contains 32 million passwords. Its content originates from a data leak of the company RockYou from the year 2009, who stored their passwords unencrypted.

Rainbow attacks are just a more specialized form of dictionary attack, where pre-computed hashes are stored in so-called rainbow tables. This serves as a lookup table to invert a certain hash function. This variant results in a time-memory trade-off, where the process is faster since only comparison operations have to be performed, but this is only effective if the rainbow-table is stored in RAM, furthermore rainbow-tables take up more space on the hard disk than wordlists. The use of a key derivation function that uses a salt makes this attack infeasible.

Hybrid attacks combine brute force and dictionary attack by appending or prefixing each entry of a wordlist with all permutations of the brute force keyspace. It is a great method if the format of a password ist known. An example would be the use of the word 'password' combined with all numbers from 0 to 9999. This is also very useful with usernames of companies using simple rules like an identifying string combined with an incrementing number. (e.g. C1710475117)

Rule-based attacks use methods to change passwords to cover all variants. It is a very flexible and efficient attack, but also very complicated to cover all desired cases. Ideal if the password requirements for a system are known, since passwords shall be easy to remember while being secure at the same time, many users use common passwords and modify them in a way that they match the password policy. (e.g. 'P@ssw0rD')

Syllable attacks are again a combination of brute force and dictionary attack in which all permutations of each word within a wordlist are checked. This is used when the password is not a real word and is often better than pure brute-forcing.

Other attacks, like Combinator, Fingerprint, Mask, Toggle-Case, Permutation, Table-Lookup, PRINCE may be similar to the ones noted above and shall be mentioned but are not covered in any detail here.

2.3. John the Ripper

John the Ripper acronyms for John or JtR is a popular open-source password cracker from the Openwall developers. Historically, the main use has been to find weak UNIX passwords. To this day, john has been evolved to be fast and rich in features. John has the ability to recognize password hash types and supports multiple cracking modes, but it is mostly used for dictionary-style or incremental brute-forcing attacks. It should be noted that john even provides the ability to define custom cracking modes using a built-in C compiler. The basic command-line is listed below.

```
1 john [OPTIONS] [PASSWORD-FILES]
```

The developers provide additional variants like a commercial *John the Ripper Pro* version and *Hash Suite*, which is recommended when using Android or Windows, which performed rather badly in my tests. Besides the above-mentioned programs, there is also a graphical version of John named *Johnny* [1], for those who are not friends of the console, that is available for Linux, macOS, and Windows.

Furthermore John offers some support for CUDA and OpenCL GPUs[2]. Whereas, CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs. OpenCL (Open Computing Language), on the other hand, is a low-level API for heterogeneous computing that runs on CUDA-powered GPUs. Using the OpenCL API, developers can launch compute kernels written using a limited subset of the C programming language on a GPU. [W7].

Implementations exist for many UNIX-based systems such as Linux, macOS, Android, and Solaris, as well as for Windows. But also for more exotic devices running under DOS, BeOS, or OpenVMS. The resources are provided as pre-compiled binaries or as source code. The latter can then be adapted as desired and compiled for a specific system. A basic version of John can be installed using package managers such as *apt*, *pkg*, or *snap*. However, it is recommended to choose the community enhanced-jumbo version, which offers support for a variety of additional password hash types as well as non-hashes like SSH private keys, RAR archives, or PDF files. The following listing provides the necessary steps to clone and build latest bleeding-edge Jumbo version of John on most UNIX-based system. Source files for all versions could also be downloaded from Openwall.net[3]

```
1 # Clone GIT repository
2 git clone git://github.com/magnumripper/JohnTheRipper -b bleeding-jumbo john
3 # Build
4 cd ./john/src
5 ./configure && make -s clean && make -sj4
```

Listing 2.1: John the Ripper - Clone and Build

It is also very easy to get John the Ripper to run on Android. Generic builds for ARM, x86 and MIPS processor architectures can be found on the Openwall wiki.[4] A rooted device is not needed but a place in the filesystem with write permission. This is in most cases: */data/local*,

[1] https://openwall.info/wiki/john/johnny
[2] https://openwall.info/wiki/john/GPU
[3] https://download.openwall.net/pub/projects/john/
[4] https://openwall.info/wiki/john/custom-builds#Compiled-for-Android

/data/tmp or */data/local/tmp/*. Note that those folders are cleared after a restart. Termux terminal emulation tool with its built-in package manager pkg and apt may be used to download and install John, but this worked rather badly in this scenario.

```
Host$ adb devices # Start ADB Deamon
# Get Android device CPU architecture
Host$ adb shell getprop ro.product.cpu.abi
# Copy files to a writable space on the Android device
# ARCH must be replaced by the appropriate CPU architecture
Host$ adb push user/. /data/local/tmp
Host$ adb push libs/ARCH/john /data/local/tmp
Host$ adb shell # Start an android shell
# Done. Change permissions and run.
Android$ cd /data/local/tmp
Android$ chmod +x ./* && ./john
```

Listing 2.2: John the Ripper - on Android

2.4. Hashcat

Hashcat is yet another very popular cross-platform cracking tool, whose developers claim to have developed the world's fastest and most advanced open-source password recovery utility. Since Hashcat version 3, the old CPU-based *hashcat-legacy* and GPU-based *oclHashcat* have been merged. The tool allows to use all OpenCL compatible processors of the same device in parallel and even work distributed over the network.

```
hashcat [options]...
    hash|hashfile|hccapxfile
    [dictionary|mask|directory]...
```

Hashcat supports five unique attack modes for more than 200 highly optimized hash algorithms. No official numbers have been found regarding the number of supported hash algorithms for john. According to the statistics displayed in Figure 5.1 of this work, 199 hash algorithms were successfully tested with Hashcat and 2319 with John. Many of the algorithms tested under john are combinations of algorithms. These combinations are identified by the foreword *dynamic_* and a subsequent number. Such combinations only appear in the jumbo version and have been added by the community.

Hashcat currently supports CPUs, GPUs, and other hardware accelerators such as DSPs, FPGAs, and co-processors in Linux, Windows, and macOS, as well as distributed password decryption functions. Almost all of this information and much more can be found on the Hashcat man page, which is one of the best that I have seen so far. Hashcat can be downloaded via package manager like *apt* under Linux or *brew* on macOS. But here it is recommended to download the source files again and build it manually. Compared to john, however, fewer platforms are supported here. The processors available for each platform can be listed using *-I* or *–opencl-info*. These may need the corresponding drivers for Intel, AMD, or NVIDIA to be addressed correctly. This process will not be successful on every platform.

```
# Clone GIT repository
git clone https://github.com/hashcat/hashcat.git
# Build
cd ./hashcat
make && make install
```

Listing 2.3: Hashcat - Clone and Build

2.5. Miscellaneous Tools

There is no black and white, and all tools have their strengths as well as their weaknesses. Also, combinations between some are quite useful. This section consists mostly of descriptions of the individual developers as they can best introduce themselves and as this section is only intended to introduce further tools. Written-off paragraphs are marked with a footnote. Here the Kali Linux knowledge-base [W2] has been referenced, which in turn refer to the developers' websites and Github repositories, as well as giving short usage examples. The following examples have been selected by versatility and range of application to present other great tools besides Hashcat and John. *aircrack-ng* suite to analyze and attack wireless networks. *THC-Hydra*, an online login cracker and the alternative *ncrack* for our *nmap* lovers, Medusa is be the third in line. WFuzz, the web application cracker and last but not least RainbowCrack which uses rainbow tables to crack passwords.

2.5.1. aircrack-NG

Aircrack-ng is an 802.11 Wired Equivalent Privacy (WEP) and Wifi Protected Access pre-shared key (WPA-PSK) cracking program that can recover keys once enough data packets have been captured. It implements the standard Fluhrer, Mantin, and Shamir (FMS) attack along with some optimizations like KoreK attacks, as well as the all-new Pychkine-Tews-Weinmann (PTW) attack, thus making the attack much faster compared to other WEP cracking tools.[5]

However, Aircrack-ng is much more than just a cracking program. The developers offer a versatile application suite presented in Table [2.1] for analyzing and attacking wireless networks. *aerodump-ng* even makes it possible to capture packets in a wireless network, and then attack contained handshakes of a WPA/WPA2-PSK using Hashcat or John efficiently.

Application	Description
aircrack-ng	802.11 WEP and WPA/WPA2-PSK key cracking program.
airbase-ng	Aimed at attacking clients as opposed to the AP itself.
airdecap-ng	Decrypt WEP/WPA/WPA2 capture files.
airdecloak-ng	Remove WEP Cloaking™ from a packet capture file.
airdrop-ng	A rule based wireless deauthication tool.
aireplay-ng	Inject and replay wireless frames.
airgraph-ng	Graph wireless networks.
airmon-ng	Enable and disable monitor mode on wireless interfaces.
airodump-ng	Capture raw 802.11 frames.
airolib-ng	Precompute WPA/WPA2 passphrases.
airserv-ng	Wireless card TCP/IP server to use wit multiple applications.
airtun-ng	Virtual tunnel interface creator.
packetforge-ng	Create encrypted packets that can be used for injection.

Table 2.1.: Aircrack-ng Suite

[5]Source: `https://tools.kali.org/wireless-attacks/aircrack-ng`

2.5.2. THC Hydra

Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add. This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely.[6]

2.5.3. ncrack

Ncrack is a high-speed network authentication cracking tool. It was designed using a modular approach, a command-line syntax similar to Nmap, and a dynamic engine that can adapt its behavior based on network feedback. It allows for rapid, yet reliable large-scale auditing of multiple hosts. Ncrack's features include a very flexible interface granting the user full control of network operations, allowing for very sophisticated bruteforcing attacks, timing templates for ease of use, runtime interaction similar to Nmap's, and many more. [7]

At this point should be noted that Nmap itself can be used for simple online attacks, by using the *-script* parameter with the desired script like *telnet-brute.nse* and the corresponding values for *userdb* and *passwd* with the additional parameter *-script-args*.

2.5.4. WFuzz

Wfuzz (the web fuzzer) is a tool designed for bruteforcing Web Applications, it can be used for finding resources not linked (directories, servlets, scripts, etc.), bruteforce GET and POST parameters for checking different kind of injections (SQL, XSS, LDAP,etc.), bruteforce Forms parameters (User/Password), Fuzzing, etc. [8]

2.5.5. RainbowCrack

Hashcat nor john are capable of performing rainbow attacks. This is were RainbowCrack comes into the game. An alternative tool to perform rainbow attacks is Ophcrack, a free Windows (LM and NTLM) password cracker which also provides a GUI, and WOphcrack, a PHP based web frontend for Ophcrack.

RainbowCrack is a general propose implementation of Philippe Oechslin's faster time-memory trade-off technique. It crack hashes with rainbow tables. RainbowCrack uses a time-memory tradeoff algorithm to crack hashes. A time-memory tradeoff hash cracker needs a pre-computation stage. At the time, all plaintext/hash pairs within the selected hash algorithm, charset, plaintext length are computed, and results are stored in files called rainbow table. It is time-consuming to do this kind of computation. But once the one-time pre-computation is finished, hashes stored in the table can be cracked with much better performance than a brute force cracker. [9]

[6] Source: `https://tools.kali.org/password-attacks/hydra`
[7] Source: `https://tools.kali.org/password-attacks/ncrack`
[8] Source: `https://tools.kali.org/web-applications/wfuzz`
[9] Source: `https://tools.kali.org/password-attacks/rainbowcrack`

3. Benchmarking

This case study is based on the comparison between john and hashcat in terms of performance. In order to be able to carry out a comparison, benchmarks of various devices have been collected. For this purpose, the integrated benchmarking functions of hashcat and john are used. The results, together with some additional information about the device, are stored in a database. The individual steps are explained further below. It is very important that an idle system is used when executing the benchmarks to avoid biased results. In addition to the benchmarks, information such as operating system, processor, and version of the password cracker are included.

3.1. John the Ripper

John benchmarks all the enabled *ciphertext format crackers* with the *-test* flag. In order to test a specific ciphertext, the flag *–format=...* must be used. The following shell script generates all information to be processed by the parser.

```
1 #!/bin/bash
2 {{ uname -a
3 } && { lscpu | grep Model;
4 } && { cat /etc/os-release | grep PRETTY_NAME;
5 } && { john | grep version;
6 } && { john --test; };
7 } 1> $(hostname)_$(date +%Y_%m%d%_H%M_%S).log
```

Listing 3.1: John the Ripper - Generate Benchmarking Logfile

3.2. Hashcat

The flag *-b* is used by Hashcat to run benchmark of selected *hash-modes*. In order to benchmark all hash-modes, the additional flag *–benchmark-all* is required. The shell script below generates all necessary information processed by the parser.

```
1 #!/bin/bash
2 {{ uname -a
3 } && { cat /etc/os-release | grep PRETTY_NAME;
4 } && { hashcat -I;
5 } && { hashcat -b --benchmark-all; };
6 } 1> $(hostname)_$(date +%Y_%m%d%_H%M_%S).log;
```

Listing 3.2: Hashcat - Generate Benchmarking Logfile

4. Implementation

In order to be able to have a comparison between the employed tools, *JtR* (See)3.1) and *Hashcat* (See 3.2) as well as hardware and operating system, a method of data visualization has been developed, by parsing the results from the tools inherent benchmarking feature, piping the results to a database to finally plot easy to understand charts. The individual steps are explained further below. The Section 5.1 provides an overview of successfully tested devices up to this date. The implementation has been made available online [W1].

4.1. Parser

The following shows an example of one record displayed by john. OMP stands for the available OpenMP threads. STATUS Indicates whether a benchmark was successful (DONE) or faulty (ERROR). Although a faulty benchmark will also return a result, it will be removed by the parser in the next step. TYPE depends on the HASHMODE and is one of the following: *Raw*, *Short(A) and Long(B)*, *ManySalts(A) and OnlyOneSalt(B)*. Where only *Raw* is displayed in one line and for the other two lines. VIRTUAL as REAL represented result of the benchmarks in h/s. Both values drift apart when the system is under load, and the real value becomes smaller. The virtual value is, therefore, the value to be expected at no load. In any case, the virtual value represents the performance of a thread. John first performs all the tests for one processor and then repeats them for each additional processor.

```
1 Benchmarking: HASHMODE, OPTIONS... (OMP) STATUS
2 TYPE_A:     REAL, VIRTUAL
3 TYPE_B:     REAL, VIRTUAL
```

Listing 4.1: John the Ripper - Example Benchmarking Record

Hashcat, on the other hand, tests all processors in parallel and also gives a total value (SUM). The individual processors are called DEVICE. For each DEVICE, there is one RESULT in h/s and some additional information. At hashcat, all HASHMODEs are provided with an ID.

```
1 Hashmode: ID - HASHMODE OPTIONS
2 Speed.#DEVICE_A.........:  RESULT (TIME) @ ACCELERATION LOOPS THREADS VECTOR
3 Speed.#DEVICE_B.........:  RESULT (TIME) @ ACCELERATION LOOPS THREADS VECTOR
4 Speed.#*.........:  SUM
```

Listing 4.2: Hashcat - Example Benchmarking Record

The self-written parser works on these patterns to insert values into the database. Both tools use the result of the script from Chapter 3. These contain additional information about the system, which is handled separately by the parser. In addition, the parser takes care of any deviation of the pattern or creeping error messages. This has been a problem for john because it doesn't offer such a clear pattern as Hashcat.

4.1.1. Database

MySQL server 5.7 has been used for the database. The Entity Relation Diagram (ERD) below displayed in Figure 4.1 provides a simplified view of the entire database. In this simple relation, available hashmodes are stored together with employed devices. *Mode* and *Option* form a unique key for a *Hash*, as do the *Device* combinations of *Model*, *OS*, *CPU*, *Tool* and its *Version*. Furthermore, several benchmarks can be saved per device combination and hash mode. The attribute *tool* functions as a distriminator for Hashcat or John. Due to the different HASHMODE TYPEs mentioned in 4.1 ist the actual Benchmarking realtion for John much diffrent then those for Hashcat by storing more detailed information.

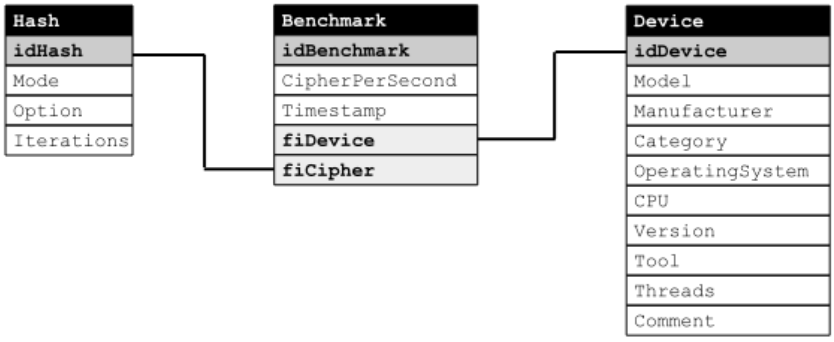

Figure 4.1.: Benchmarking - Simplified Database View

4.2. Plotter

For the creation of any graphics on the website as well as here in the appendix B, *python* with the *matplotlib* Library have been used. The data is loaded from the database using the *MySQLdb* Library. The website provides an interface for executing custom queries on the database allowing users to plot custom graphs and can be found under `https://93.177.65.122/query.php`.

4.3. Website

The website serves as a wrapper to provide all previous steps of data processing. The front-end consists of *HTML5*, *CSS3* and *ES6* using the *JQuery* framework. The backend consists of *PHP7*, *Python3* and *Mysql*. *phpMyAdmin* has also been installed for convenient data management. The website is provided by *Apache2*, and is protected from web crawlers by simple *.htacces* and *.htpasswd* configuration and runs on a private *vServer*. Interested parties can visit the website with the username 'anyone' and the password 'campus09'.

5. Results

This chapter serves as a wrapper for the Table 5.1, which provides an overview of the resulting data of this case study. These information provides the basis for the subsequent Analysis 6 and Conclusion 7. Except for a few devices, this table includes all devices successfully tested in the context of this seminar paper. Here, the combination of Processor, OS, and Version composes the identifying key resulting in a total of 25 different devices and 42 unique combinations benchmarked. The records are numbered in order to reference single combinations best. The number of hash modes available varies between Hashcat and John. That is why only *descrypt* and *md5crypt* are mentioned here because no mode is supported by all combinations. All benchmarks are in hashes calculated per second (h/s). It is essential to understand that these results do not allow ranking of the devices as the computing performance depends on the hash mode and behaves not proportionally between devices. So that some combinations may perform great at certain hash modes.

For a more versatile comparison visit the benchmarking website [W1].
Graphical representations of this case data can be found in Appendix B.

Comments

It should be noted that the records in this table have been minimized for space reasons. Some information, like software or OS subversions, as well as cross-product type descriptions of processors and devices, have been deliberately omitted, without losing the identifying role. The original data set is more complex and comprehensive.

The devices *TheBigK* and *A1707* play an important role in the following since the device *A1707*, on the one hand, is the most versatile device tested in this comparison and *TheBigK* is the most performant and advanced. The former is a *MacBook Pro (15-inch, 2017)* and is in the middle field of the performance scale. The device as a platform offers three processors. A CPU, as well as two different GPUs. The tests were performed on four different operating systems. Linux, Windows, and macOS were covered. Also, several different builds and versions of john were tested under macOS. Without forecasting, it should be said that the results for these combinations are far apart. The latter is a self-assembled workstation. This platform contains a *Intel i9-9900K* and a *NVIDIA 2080ti* processor, both of which are among the strongest of their kind in the end-user area. Other interesting data sets relate to the use of ARM-based processors. On one side the embedded devices *Raspberry Pi 2b+*, *3b+* and *ODroid UX4* were tested and on the other side the Android flagships: *Samsung S9+*, *Samsung Tab S3* and *Xiaomi Mi Mix 3*.

5.1. Statistics

The following Figure 5.1 provides an overview of the mass of case data obtained. A device may have multiple processors. For each device, benchmarks were performed for one or more combinations of individual processors, OS, and cracking tool version. The combination influences the number of hashes that can be calculated. For many hash modes, there are additional options for various applications. So the statistics table can be read as follows: 9864 benchmarks were calculated for john, which were collected from 43 combinations of 24 different devices. The benchmarks consist of 511 hash modes, which were performed with five different options on average. It should be noted that the number of available or supported hash modes varies greatly depending on the combination. This leads to a minimum of 5, a maximum of 568, and an average of 230 tested hash mode options per combination. Hashcat, on the other hand, supports mostly always the same hash mode options if the employed processor is supported.

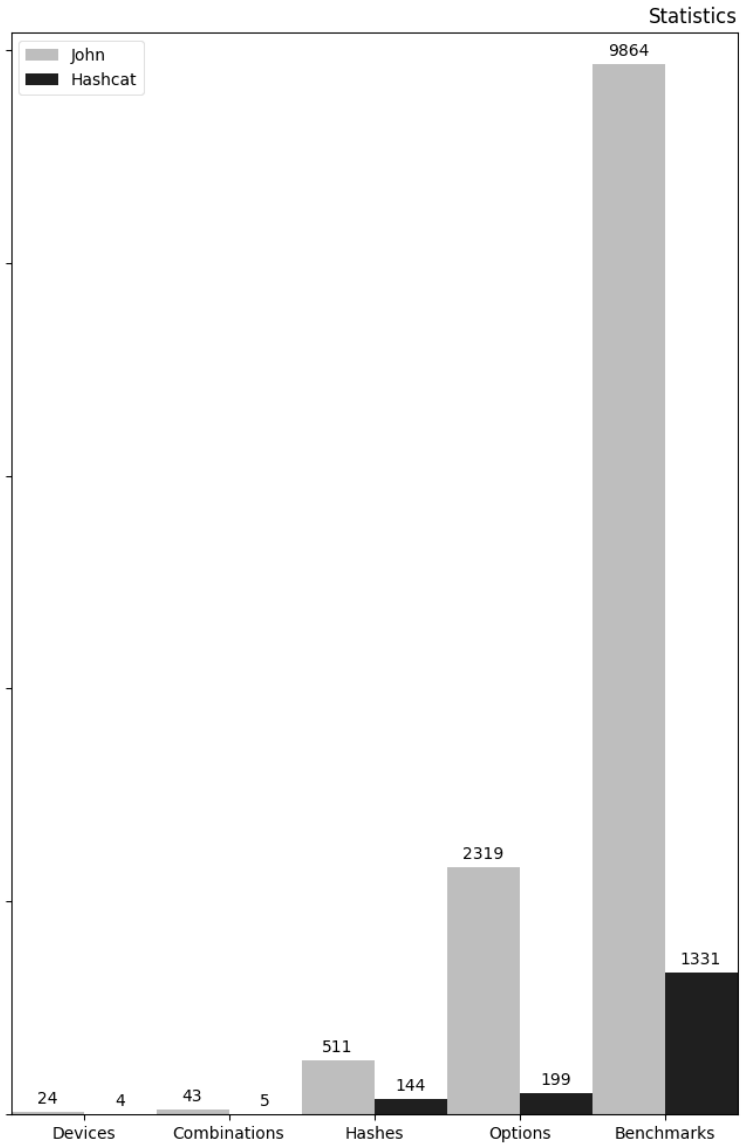

Figure 5.1.: Benchmarking Statistics

ID		Combination			Results	
Ⓝ	Device	Processor	OS	Version	descrypt	md5crypt
John the Ripper - Central Processing Units						
①	TheBigK	i9-9900K	Windows 10	1.8.13-j.	102094000	-
②	A1707	i7-7700HQ	MacOS 10.14	1.9-j.	11460000	79920
③	A1707	i7-7700HQ	MacOS 10.14	1.8-AVX	34427000	-
④	A1707	i7-7700HQ	MacOS 10.14	1.8-SSE	17032000	-
⑤	A1707	i7-7700HQ	Debian 10	1.8	5403000	-
⑥	A1707	i7-7700HQ	Windows 2019	1.8.13-j.	33288000	-
⑦	A1707	i7-7700HQ	Windows 2016	1.8.13-j.	28494000	-
⑧	15cx	i7-8750H	Windows 10	1.8.13-j.	34841000	-
⑨	MB X Pro	i7-8550U	Windows 10	1.8.13-j.	33255000	-
⑩	Swift 5	i7-7500U	Mint 19.1	1.8	5564000	-
⑪	Swift 5	i7-7500U	Mint 19.1	1.9-j.	22880000	190848
⑫	Custom	i7-6700	Windows 10	1.8-j.	39317000	-
⑬	A1391	i7-4770HQ	Debian 10	1.8	1222000	-
⑭	A1391	i7-4770HQ	MacOS 10.14	1.9-j.	2163000	16240
⑮	M83	i7-4790	Ubuntu 18.04	1.8	6057000	-
⑯	M83	i7-4790	Ubuntu 18.04	1.9-j.	46268000	403200
⑰	K75VJ	i7-3630QM	Kali 2019.1	1.8.13-j.	4678000	-
⑱	K75VJ	i7-3630QM	Debian 10	1.8	20545000	-
⑲	E5470	i5-6300U	Ubuntu 16.04	1.8	4626000	-
⑳	Custom	i5-4670S	Windows 10	1.8-j.	35219000	-
㉑	Custom	i5-3570k	Windows 10	1.9-j	22137000	197204
㉒	Surface 3	x7-Z8700	Windows 10	1.9-j	5574000	48573
㉓	E13	Pentium N3540	Kali 4.18	1.8.6-j.	1601000	-
㉔	ES13	Celeron N3050	Kali 4.18	1.8.6-j.	1601000	-
㉕	DL360 G4	Xeon 2005	Debian 10	1.9-j.	2113000	18912
㉖	vServer	QEMU	Ubuntu 18.04	1.8	2412000	-
㉗	vServer	QEMU	Ubuntu 18.04	1.9-j.	2560000	21792
㉘	Pi 2b+	BCM2837	Kali 2019.1	1.8.13-j.	606208	-
㉙	Pi 3b+	BCM2837B0	Kali 2019.1	1.8.13-j.	716800	-
㉚	UX4	Exynos 5422	Ubuntu 18.04	1.9-j.	2465000	27881
㉛	S9	Exynos M3	Android 9	1.8	2913000	-
㉜	Tab S3	Snapd. 820	Android 9	1.8	1607000	-
㉝	Mi Mix 3	Snapd. 845	Android 9	1.8	1873000	-
John the Ripper - Graphics Processing Units						
㉞	15cx	GTX 1050Ti	Windows 10	1.8.13-j.	79324000	-
㉟	MB X Pro	MX150	Windows 10	1.8.13-j.	-	-
㊱	A1707	Raedon 555	MacOS 10.14	1.9-j.	-	-
㊲	A1391	Iris Pro 630	MacOS 10.14	1.9-j.	-	-
Hashcat - Graphics Processing Units						
㊳	Custom	GTX 770	Windows 10	v5.1.0	13266500	855300
㊴	A1707	Radeon Pro 555	macOS 10.14	v5.1.0	-	1087000
㊵	A1707	HD Graphics 630	macOS 10.14	v5.1.0	-	221000
Hashcat - Central Processing Units						
㊶	K75VJ	i7-3630QM	Kali 2019.1	v5.1.0	1842900	1714
㊷	ES13	Celeron N3050	Kali 4.18.0	v5.1.0	0	3893

Table 5.1.: Tested Combinations. Results in h/s.

6. Analysis

The results from Chapter 5 contradict many of the assumptions I had made beforehand, especially regarding the use of different software versions and operating systems resulting in huge differences in the results. Whereby the hardware behaves as expected, and the computing power of the processors is reflected. Note that not all observations can be derived from the example hash types from Table 5.1, but are based on the entire [W1] data set. The analysis is limited to the device *A1707*, as it is the most versatile device tested, and the *descrypt* hash type. However, the observations are also applicable to other devices. And, the corresponding entries are highlighted in the table 5.1 and are referenced below by circled numbers Ⓝ.

Starting with the results of John the Ripper using the same CPU, knowing that ③ works best and is described with 100% performance. The results of other combinations are given as a fraction of ③'s performance. When comparing operating systems, it is now fascinating to note that the performance of ⑤ under Debian was only 16%, but the results of ⑦ on Windows 2016 (83%) and ⑥ on Windows 2019 (96%). Note the small difference of 13% better performance for the current Windows Server version. It should be noted that Windows as a host operating system generally performs very well in many cases, while Linux performs rather poorly. Although the best result was achieved with this device under macOS, there are massive differences in performance depending on the version of john used, where ② only achieved 33%, which is the second-worst result of this test series. Although, as in ⑩|⑪, ㉖|㉗ and ⑮|⑯, using the current self-compiled jumbo version of john resulted in a performance improvement over the otherwise used version 1.8, which is installed by default via package manager. However, the decisive factor here was the use of the appropriate CPU command set when compiling. Streaming SIMD Extensions 2 (SSE2) was released in 2000 with the Pentium 4, while Advanced Vector Extensions (AVX) was optimized for newer processors, which is also reflected in the results and represents a performance difference of 50% on the same processor. ② was compiled here for generic processors.

After comparing the OS and SW, the focus is now on the use of GPUs. As one can see from the test data, the support of GPUs under John is poor and depends on hardware and software employed. The device considered here has two different graphics cards, but only one of them could be used in one of 6 possible combinations of OS and SW. Hashcat, on the other hand, supports both graphics cards, but not the CPU. Unfortunately, hashcat was not tested extensively in this study. However, Hashcat does by far not run on every hardware, but on average, it delivers better results. Furthermore, the simultaneous use of processors is a big advantage.

One last observation is still to be shared. ⑰ and ⑱ were both performed on the same hardware. Both use pre-compiled versions of john. ⑰ uses the version shipped with Kali Linux and ⑱ was installed using apt on Debian 10. From the observations seen so far, ⑱ should perform rather poorly, although ⑰ is expected to perform rather well under Kali Linux. However, the results show a 77% gap in favor of ⑱.

7. Conclusion

In summary, the topic of key recovery was introduced and various application areas, attack modes and tools were briefly presented. The offline-based tools john and hashcat were examined more closely and benchmarks were established. These were then analysed as case data for a comparison between the two tools to define which factors can influence their performance.

7.1. Conclusion

In conclusion, john and hashcat both have very different applications and are not necessarily directly comparable. John offers a variety of additional hash types in its Community enhanced jumbo version and is suitable for determining and cracking weak passwords. Furthermore, the tool can run on almost any hardware. Hashcat, on the other hand, should be used when there is strong underlying hardware and a strong password to crack. Advantageous here is the support of graphics cards, their parallel usability and if necessary the distribution over the network. But there are still downsides because less hashes and hardware is supported.

Furthermore, both tools have strengths and weaknesses in some hashes, like john is very strong in cracking LM-hashes. As has been shown, it is not enough to have the strongest hardware, it is more important to adjust the software to the hardware in order to use it optimally. Incorrect implementation of the software can lead to extreme differences in performance. It should be decided on a case by case basis, what the optimal platform for cracking a target hash is, by performing benchmarks for the target hash. When using john the jumbo version should always be used. Moreover, for john and hashcat the binaries should be self-compiled and pre-compiled versions should be avoided.

7.2. Acknowlegment

Special thanks go to all those involved who supported me in this work. Be it by sending benchmarks or by providing hardware.

A. Appendix

List of Tables and Figures

Listings

Bibliography

[W1] [redacted]: John the Ripper and Hashcat benchmarkings, htaccess: anyone campus09
`https://93.177.65.122` 10, 12, 15

[W2] **Kali Linux**: The quieter you become, the more you are able to hear
`https://www.kali.org`
`https://tools.kali.org`
Visited: 25.11.2019 1, 7

[W3] **Openwall**: John the Ripper password cracker,
`https://www.openwall.com/john/`
`https://openwall.info/wiki/john/benchmarks`
`https://openwall.info/wiki/john/johnny`
`https://openwall.info/wiki/john/GPU`
`git://github.com/magnumripper/JohnTheRipper`
Visited: 22.11.2019

[W4] **Hashcat**: Advanced Password Recovery,
`https://hashcat.net/hashcat/`
`https://hashcat.net/wiki/`
`https://hashcat.net/wiki/doku.php?id=hashcat`
`https://github.com/hashcat/hashcat`
Visited: 23.11.2019

[W5] **Aircrack-ng**: complete suite of tools to assess WiFi network security,
`https://www.aircrack-ng.org/documentation.html`
`https://github.com/aircrack-ng/aircrack-ng`

[W6] **Varonis**: How to Use John the Ripper,
`https://www.varonis.com/blog/john-the-ripper/`
Updated: 13.08.2019, Visited: 25.11.2019

[W7] **NVIDIA**: High Performance Computing,
`https://developer.nvidia.com/opencl`
`https://developer.nvidia.com/cuda-zone`
Visited: 25.11.2019 5

[W8] **ophcrack**: a free Windows password cracker,
`https://ophcrack.sourceforge.io/`
Visited: 25.11.2019

[R1] **IBM X-Force Research**: Beware of older cyber attacks,
`https://www.ibm.com/downloads/cas/OAN7VKK4`
IBM Security, 2016 4

[B1] Hash Crack: *Password Cracking Manual.*
NETMUX, 2019, ISBN: 9781793458612. 1

[P1] Special Publication 800-63: *Digital Identity Guidelines.*
NIST, 2017, DOI: NIST.SP.800-63-3 2

B. Graphs

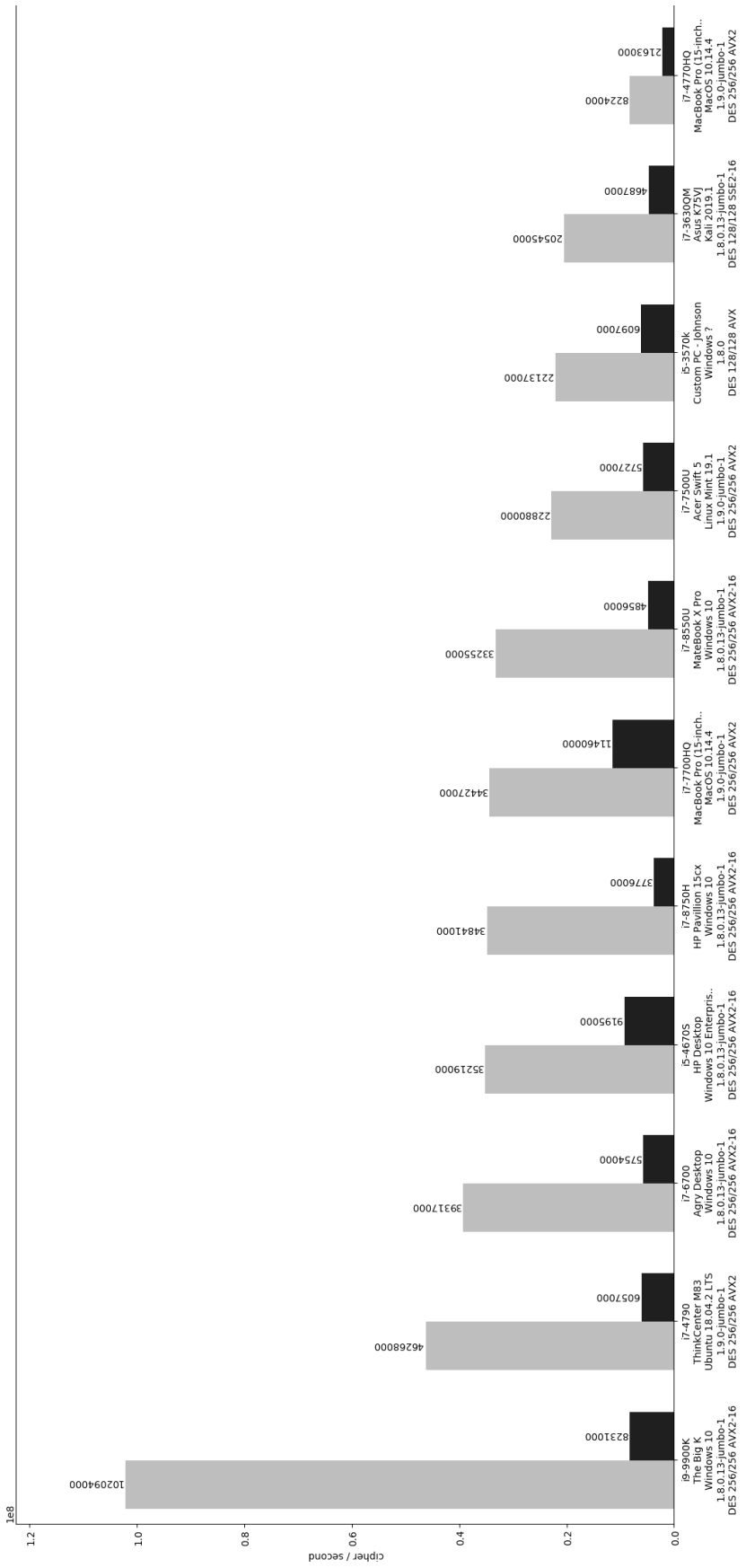

Figure B.1.: John the Ripper - descrypt - Part A

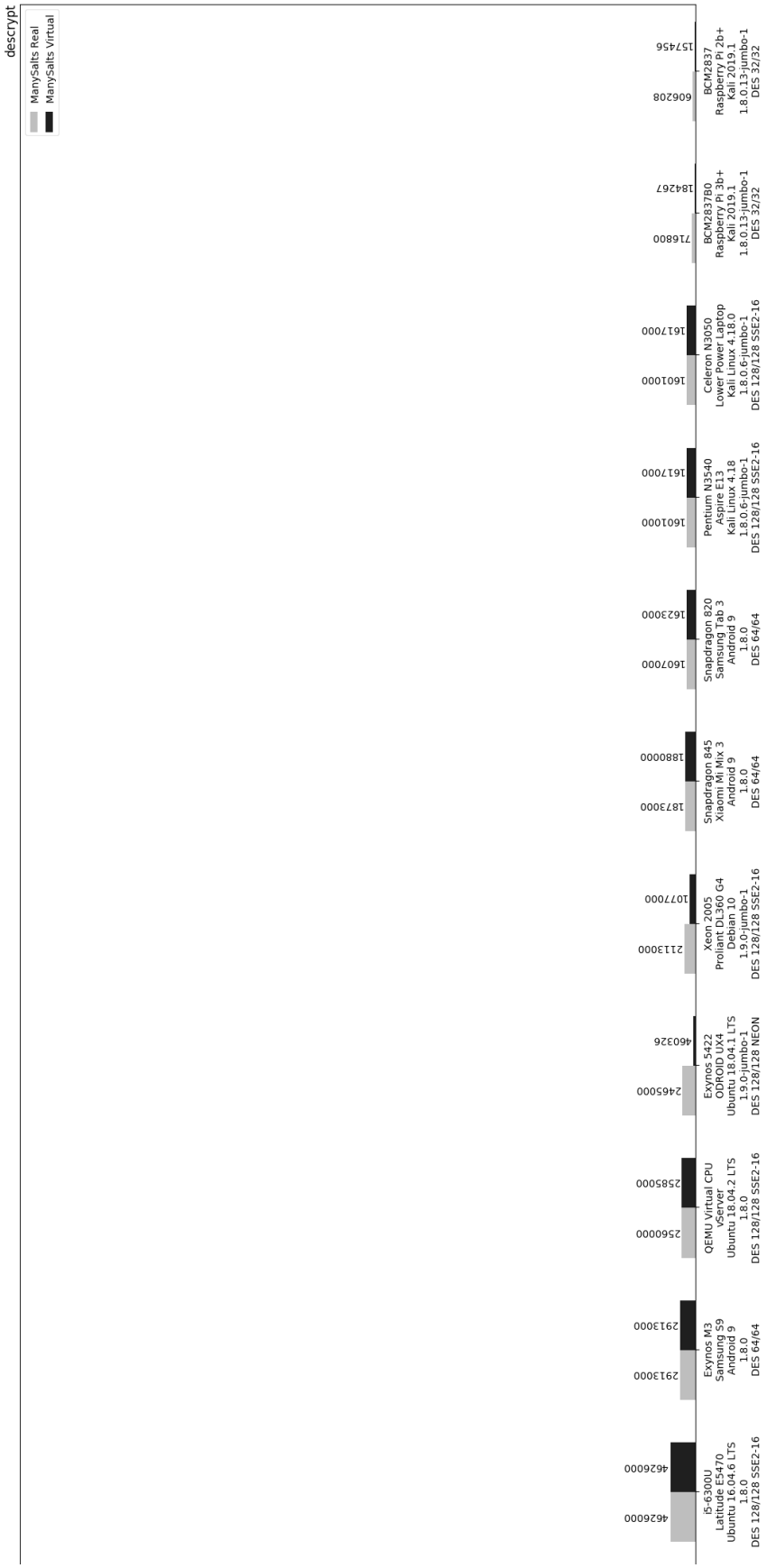

Figure B.2.: John the Ripper - descrypt - Part B

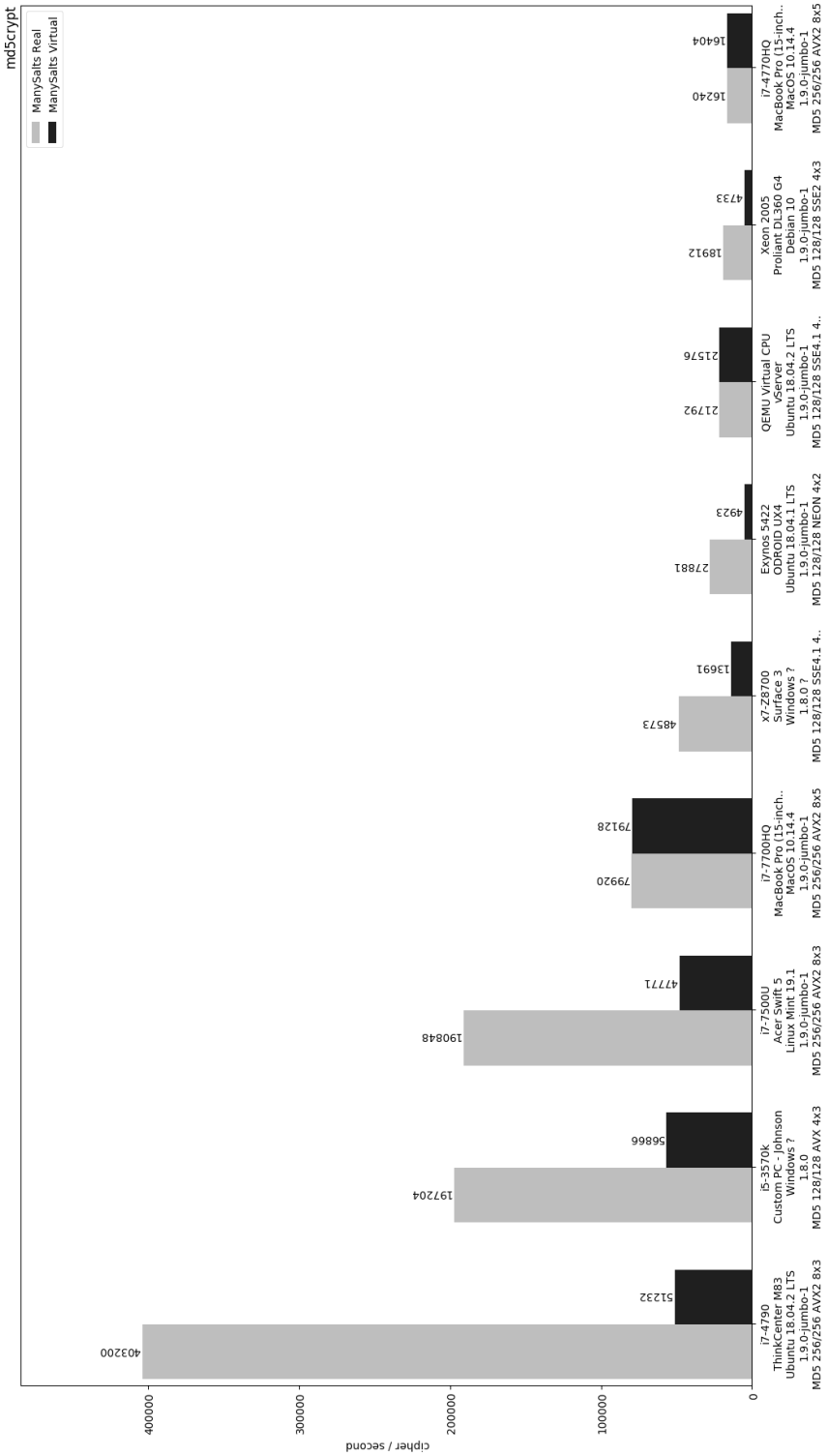

Figure B.3.: John the Ripper - md5crypt

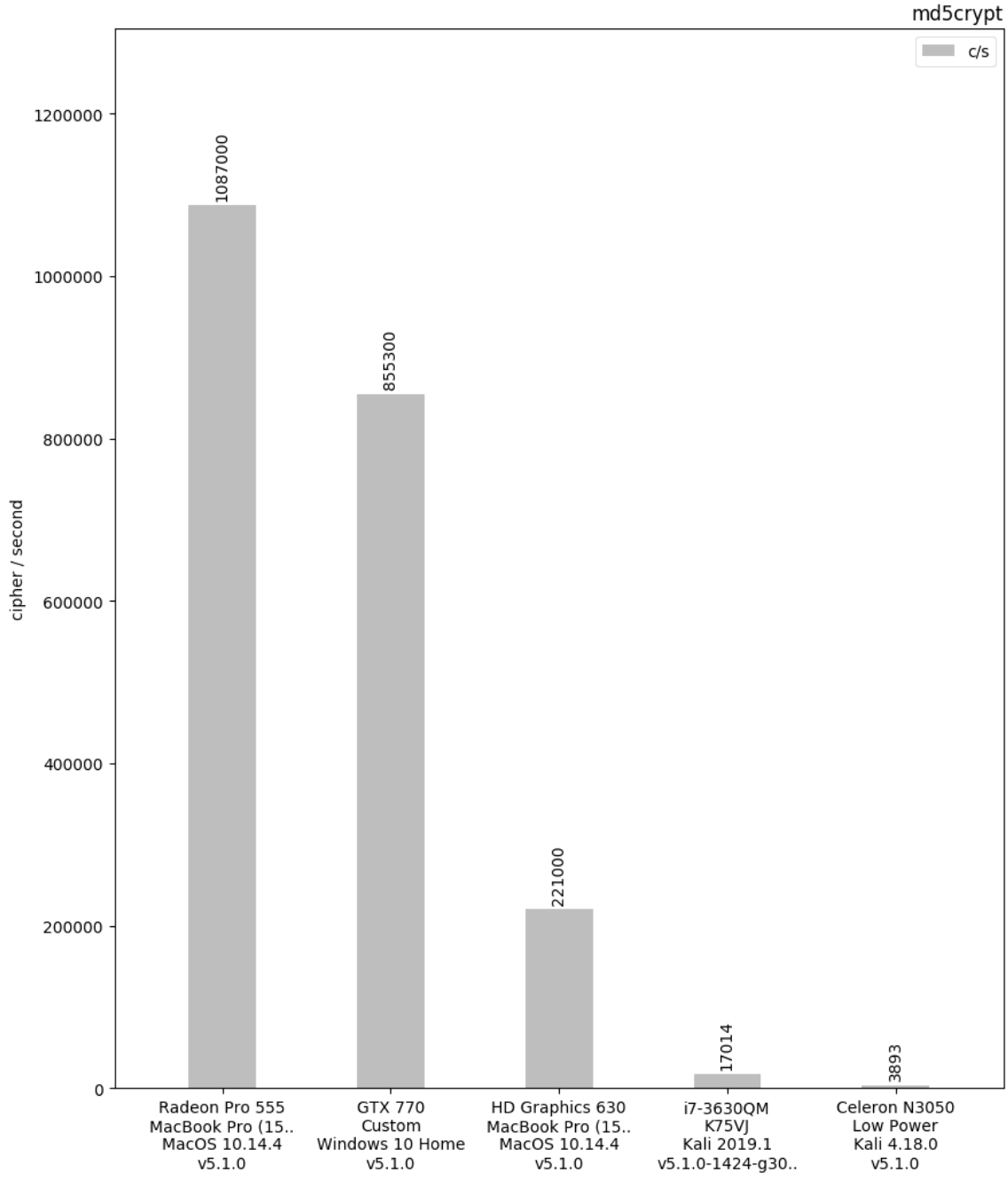

Figure B.4.: Hashcat - md5crypt

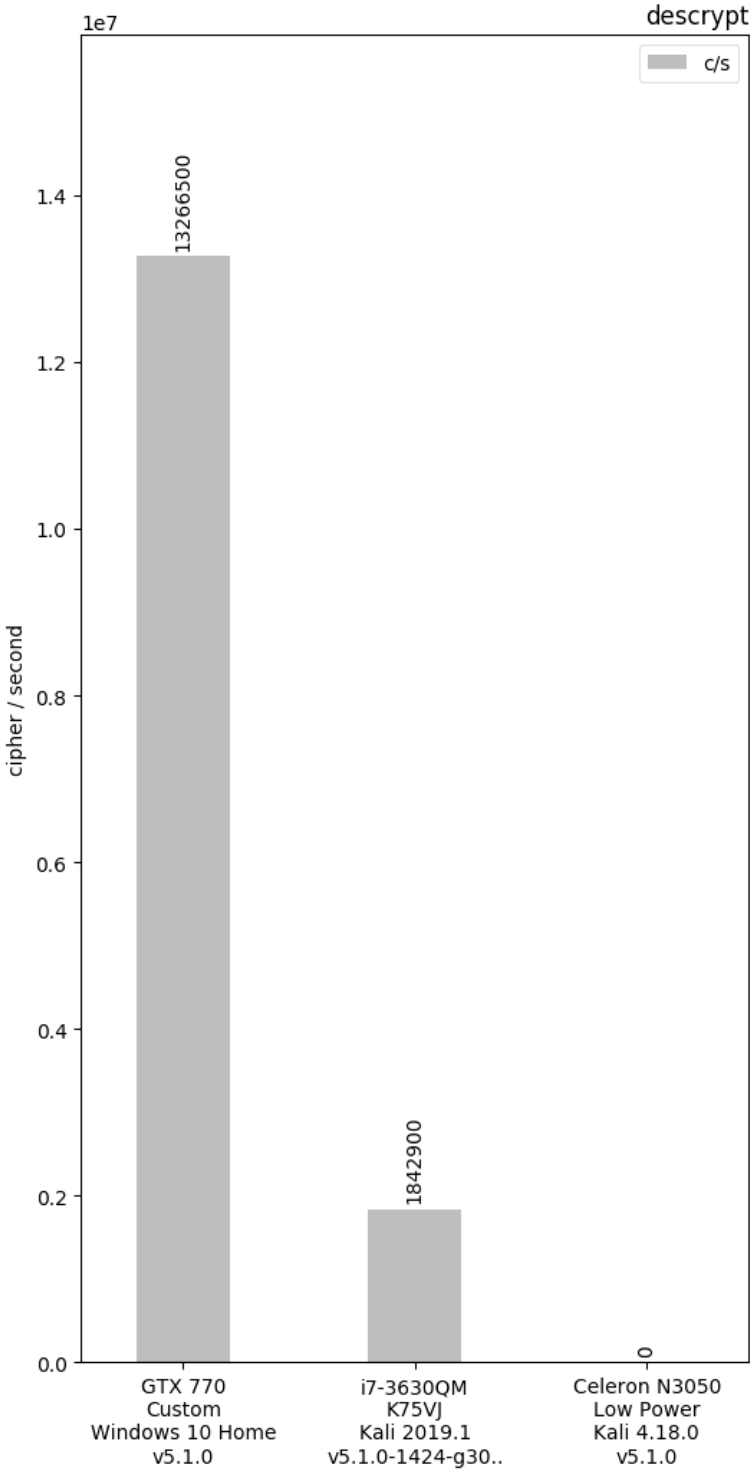

Figure B.5.: Hashcat - descrypt

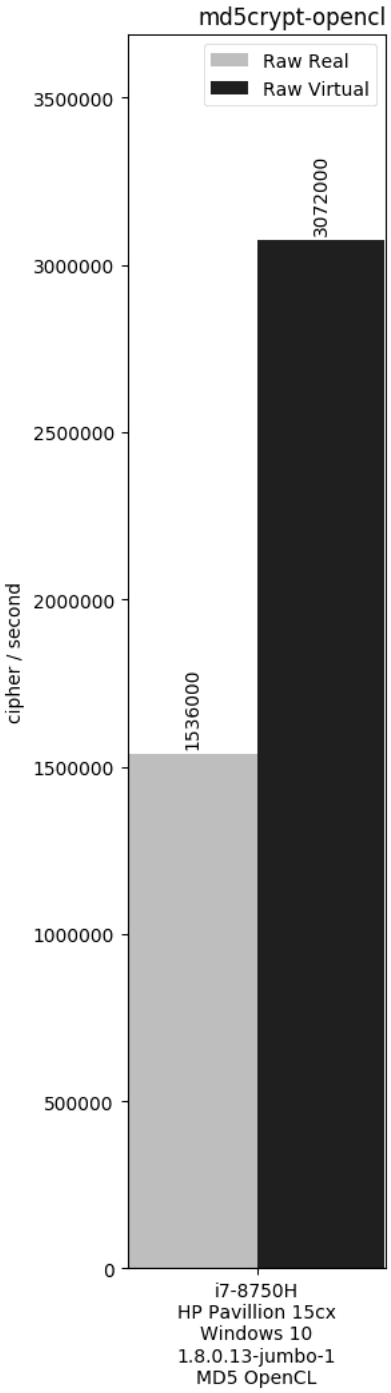

Figure B.6.: John the Ripper - md5cryp openCL

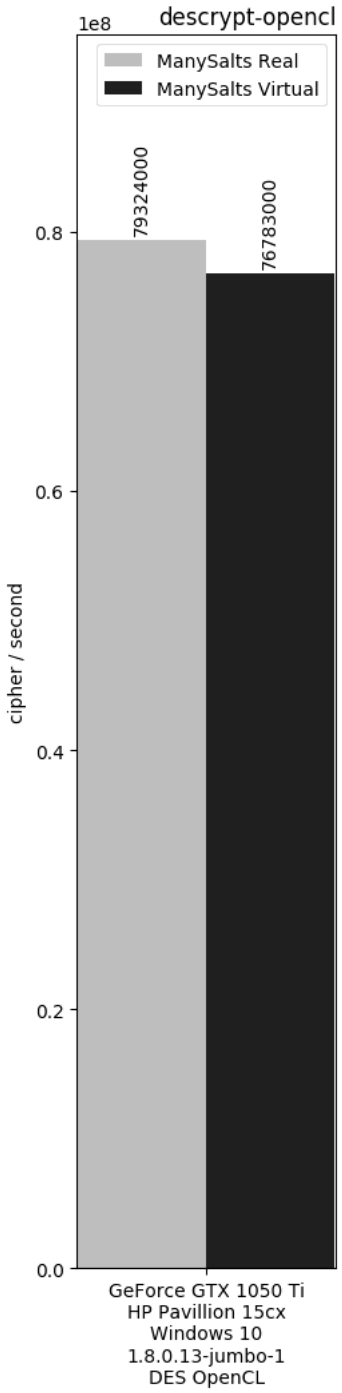

Figure B.7.: John the Ripper - descrypt openCL