NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING


# DOTA - 2024

# Defense of the Ancients

The Projects for DOTA2024
(Defense of the Ancients)
Singapore, July 2024.

# Table of Contents

# Detection and Prevention of Linux Rootkits

JIN Xuanyu[*]
University of Nottingham
Ningbo China
scyxj3@nottingham.edu.cn

CAO Yuhan[†]
Xi'an Jiaotong University
1526572820@qq.com

LU Zijun[‡]
The University of Queensland
zijun_lu@126.com

HUANG Ningyuan[§]
Southern University of
Science and Technology
12112205@mail.
sustech.edu.cn

ZHU Ziyi[¶]
Southern University of
Science and Technology
12112229@mail.
sustech.edu.cn

## ABSTRACT

Linux rootkits pose a significant threat to system security by providing attackers with unauthorized access and control while remaining hidden from detection. This paper presents three innovative techniques for detecting and preventing Linux rootkits using eBPF (extended Berkeley Packet Filter). First, we developed a program that hooks into syscalls to detect and intercept the installation of eBPF programs, which are often utilized by rootkits for malicious activities. Second, we utilized eBPF to monitor and verify the integrity of the syscall table, a common target for rootkit modifications. Third, we analyzed dumped kernel memory and compared it with debug symbols to identify suspicious entries indicative of rootkit presence. Our experimental evaluation demonstrates that these methods effectively detect and prevent rootkit activities with minimal performance overhead. The proposed techniques offer a robust and efficient solution for enhancing Linux system security against rootkits.

## General Terms

Computer Security

## Keywords

Rootkit, eBPF, Computer Security

## 1. INTRODUCTION

[*]This author proposes this topic and implements the Memory Dump Analyzer.

[†]This author is responsible for designing video.

[‡]This author is responsible for designing the poster.

[§]This author implements the eBPF Defender.

[¶]This author implements the eBPF Detector.

Rootkit is a term derived from Linux and Unix operating systems [26]. "Root" is normally referred to the account of the highest privilege, i.e., the admin account, whereas "kit" is referred to the software components that enable a malicious party to have unauthorised root access to the target machine [26]. Rootkits are one of the hardest types of malware to handle because they can hide their existence as well as their malicious activities from the victim hosts they attack [12, 18]. Detecting and removing such type of malware is still a challenge to many because it can manage to escape from capture by traditional antivirus software and detection tools of any operating system [12].

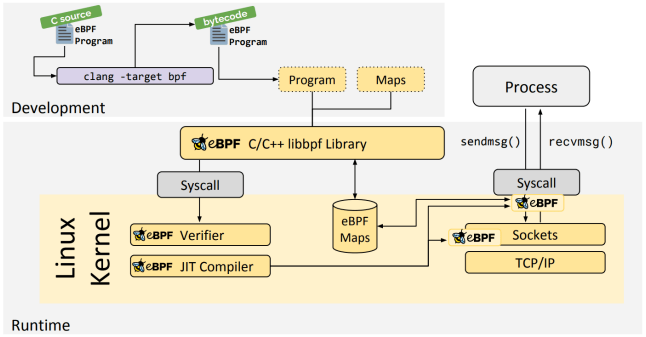
There are several techniques for implementing Linux rootkit, including LKM, eBPF, `LD_PRELOAD` rootkit, syscall table hook, inline hook, kprobe rootkit and VFS layer rootkit.

**LKM rootkit.** Loadable Kernel Module (LKM) technology is an extension mechanism for the operating system kernel that allows dynamic loading and unloading. It allows developers to add new codes, source files or system services to a running operating system kernel without recompiling the entire kernel or rebooting the system[2]. Obviously, LKM techniques can be used to insert rootkits into system memory. The LKM rootkit is capable of hiding and modifying kernel data structures and hijacking system calls. The LKM technique is also used in the rootkits described below, such as syscall table Hook and inline hook.

**eBPF Rootkit.** eBPF is a double-edged sword technology. On the one hand, it is a technology that improves functionality of the Linux operating system [19, 25, 8], but on the other hand, it invites potential malware exploitation [19]. Things become even more complicated when rootkits and this revolutionary kernel technology interweave. Linux Rootkits based on eBPF negatively affects both the network and the user space [3]. They can execute malicious code in the kernel, allowing changes made to system behaviour and other offensive operations such as escalating local privilege and replacing file content [3]. For example, they can make changes to the data from critical files such as /etc/-sudoers [3].

**LD_PRELOAD Rootkit.** The `LD_PRELOAD` rootkit exploits the Linux `LD_PRELOAD` environment variable or the

**Figure 1: eBPF Workflow**




`/etc/ld.so.preload` file to preload a malicious shared library, overriding standard system libraries when applications load [24].

**Syscall Table Hook.** The Syscall Table hook modifies the Linux kernel's syscall table to redirect system calls to custom functions, and enable malicious actions like hiding processes and eavesdropping on network traffic.

**Inline Hook.** Inline hook technique is to insert instructions to the opcode of kernel functions. Examples of these instructions include `jmp`, `push` or `ret`[11]. These instructions intercept calls to initial target-functions and then redirect the hooked function to the attacker [11, 13]. During this process, the attacker can do some malicious operations to the hooked function before and/or after the function fulfill its own genuine missions [11, 13].

**Kprobe Rootkit.** The kprobe rootkit exploits the Linux kernel's dynamic tracing capabilities (kprobes) to hack and modify system behaviour by inserting probes before and after the execution of system calls or critical functions. This allows it to hide malicious activity, manipulate system call parameters and return values, and gain hidden control and attack the system [20].

**Ftrace Rootkit.** Similar to kprobe rootkit [20].

**VFS Layer Rootkit.** VFS (Virtual File System) rootkit attacks the victim host through exploiting the virtual file system layer. It hides network connections, processes and files after intercepting file-related system calls [20].

**Details About eBPF.** eBPF is a revolutionary technology that allows sandboxed programs to run in a privileged environment such as the operating system kernel. It is used to safely and efficiently extend kernel functionality without modifying kernel source code or loading kernel modules[9]. eBPF can be used for kernel debugging and performance analysis. eBPF programs can be attached to tracepoints, kprobes and perf events and its ability to access kernel data structures allows developers to implant and test new debugging code without recompiling the kernel.[10]

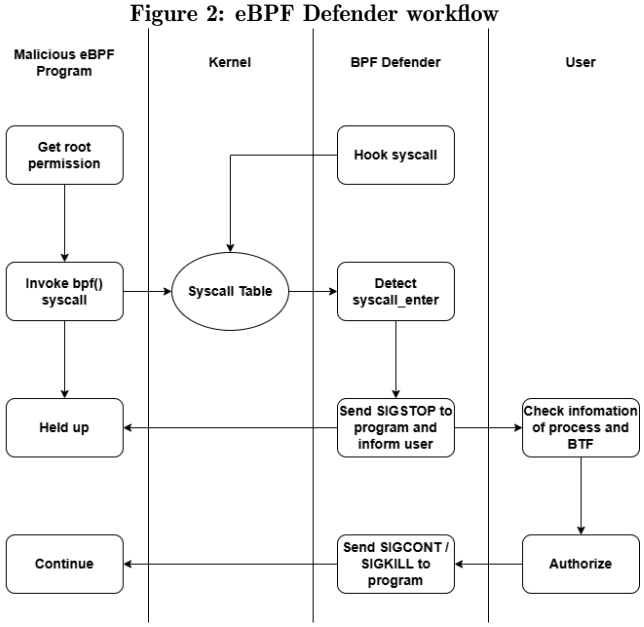
## 2. DEFENCE

To defend against and detect malware and keep your computer safe, we use eBPF in the following ways: hooking into system calls, tracing file system activity and process behaviour, analysing network traffic to detect unauthorised communications, and scanning kernel data structures such as the `sys_call_table` to defend against unauthorised modifications. In conclusion, eBPF can monitor system behaviour with minimal overhead.

This article presents three new tools to fight against Linux rootkits. A defender and a detector based on eBPF, and a memory dump analyzer.

## 3. IMPLEMENTATION

**Figure 2: eBPF Defender workflow**




### 3.1 eBPF Defender

We developed an eBPF program that hooks into key syscalls related to eBPF program installation. This eBPF program logs and prevents unauthorized installations.

#### *3.1.1 Rationale for Utilizing eBPF*

eBPF stands out for its ability to efficiently intercept syscalls, coupled with its elevated privileges that enable the monitoring and termination of suspect processes. As most of malicious software relays on the syscall, this method can efficiently block their behaviour.

#### *3.1.2 Risk Detection Methodology*

Our defender hooks on the `tracepoint/syscalls/sys_enter_bpf`. The initiation of a eBPF program installation triggers the `bpf()` syscall. Since eBPF does not allow blocking and waiting for input, we use system signals to stop the process. Once triggered, it evaluates the arguments and sends the `SIGSTOP` signal to the process if it is attempting to install an eBPF program. `SIGSTOP` is a signal that pauses a process and cannot be ignored or caught by the process, allowing the root user to review the process and decide whether to continue the installation. The defender then sends information about that process, such as PID and command, to user space for further processing.

#### 3.1.3 Authorization by Root User

Upon detecting dubious activity, the system reports the details of the implicated process and its intended eBPF hook. It uses the strings command to filter out suspicious strings and list them for the root user. Installation will only resume after the root user inputs the password and sends `SIGCONT` to the paused process.

## 3.2 eBPF Detector

Normally, the system call table won't be changed when a LKM is installed or other normal operation is done. However, changing the system call table and hook system call is an easy and popular way to implement a rootkit. As a result, we can check whether the system call address is changed. To achieve this, we can check whether the address is still in the ".text" segment, where the core data of kernel is placed. The idea comes from the article introducing `Tracee` [16].

The detector's program has two parts: user space program and kernel space program(eBPF). The eBPF program is attached to `sys_enter_ioctl` tracepoint, and the user space program invokes the ioctl system call with certain file descriptor and operation code to trigger eBPF program. The combined program checks the system call table to see if any system call's address is out of the boundary of ".text" segment, which indicates the system call has been hooked.

#### 3.2.1 Find the address of syscall table

In user space, the program read the base address of the system call table from the file `/proc/kallsyms`, which is a special file in the Linux `proc` filesystem that provides a list of all the symbols (functions and variables, including the system call table) in the kernel, along with their addresses. We can use the symbol `sys_call_table` to find and extract the address from the file.

#### 3.2.2 Pass the base address to eBPF

The user space program then pass syscall table's base address to eBPF program, which is in the kernel. To pass the information to kernel, the data structure "BPF map" is needed. The map is stored in kernel space, both user space program and kernel space program can use eBPF's API to access the data, thus share information with each other.

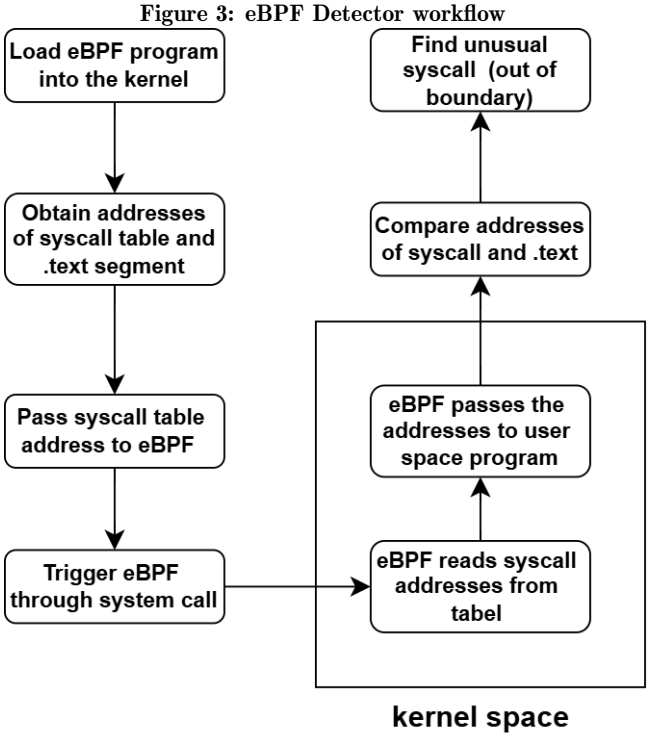
#### 3.2.3 Extract addresses of system calls

The eBPF program runs in the kernel, so it has the ability to access to kernel memory, where we can read the syscalls' addresses from syscall table, whose address we already have. Then we store the addresses to a BPF map, and let user space program to further process the data.

#### 3.2.4 Compare syscall addresses with boundaries

In this step, user space program reads the start and end address of .text segment, from the same `/proc/kallsyms` as before, using `_stext` and `_etext` symbols respectively. Then, each system call address is compared with range of ".text" segment, and warnings with useful information will be printed on the screen if any system call's address is out of the boundary, indicating that it may have been hooked by some rootkit.

**Figure 3: eBPF Detector workflow**




## 3.3 Memory Dump Analyzer

This project dumps memory using the LiME (Linux Memory Extractor) kernel module and analyses the dump file to detect suspicious kernel hooks based on the Volatility 3, a widely used forensic tool framework [1].

The project includes a Volatility 3 plugin to detect hooks in function entries, a program that analyses the syscall table to find the system calls being hooked and their names. There is also a widget to detect the `LD_PRELOAD` libraries. These information is eventually summarised and presented to the user.

#### 3.3.1 Configure the Volatility 3 framework

Volatility need a symbol table called ISF (Intermediate Symbol Format) file to parse the memory dump, which can be generated from a kernel image with debugging information and System.map using dwarf2json [23, 28]. The banners in the file must match the banners in the memory image exactly, which includes more than just the version number [27].

For most Linux distributions, the kernel image with debugging information can be downloaded as a separate package [23, 28]. Note that the size of the image may be several gigabytes.

#### 3.3.2 Dump the memory

LiME (Linux Memory Extractor) can be used to acquire memory on Linux-based devices [1].

#### 3.3.3 Analyze the memory dump

**Table 1: eBPF defender and detector test result**

| Rootkit | eBPF Defender | eBPF Detector | Memory Dump Analyzer |
|---|---|---|---|
| ebpf-rootkit[7] | Success | Fail | Fail |
| Diamorphine[15] | Success | Success | Success |
| reveng_rtkit[21] | Success | Success | Success |
| KoviD[6] | Success | Fail | Not Tested |
| evilBPF[22] | Success | Fail | Not Tested |
| bds_lkm_ftrace[4] | Success | Not Tested | Success |

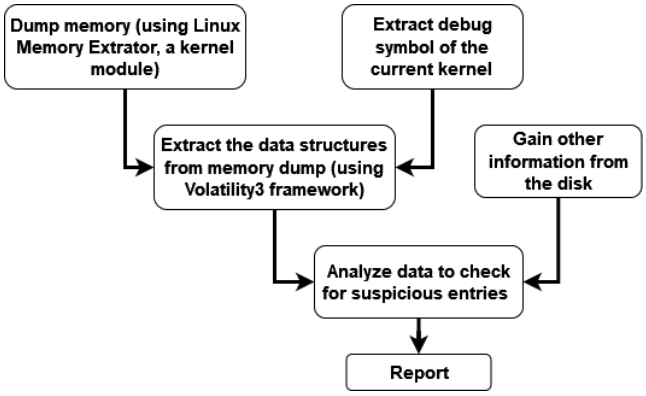



**Figure 4: Memory Dump Analyzer**

The memory dump can be analyzed by the Volatility 3 framework. The framework comes with a number of plugins such as `check_syscall`. It also allows users to write their own plugins to enable more inspection.

The `check_syscall` plugin can list information about functions in the `sys_call_table`, including their handler addresses and the corresponding symbols of these addresses. If a handler address has not been changed, the symbol corresponding to that address shall be able to be found in the symbol table, such as `__x64_sys_kill`. The framework can automatically handles memory offset to match addresses in memory dump to those in the symbol file. If a handler address cannot match any symbols, it can be assumed that the address has been modified, i.e. there is a system call table hook here.

On this basis, this project implements the filtering of the system calls that are hooked and find their original function names.

Many Linux system functions begin with a 5-byte long `nop`, such as 0x0F1F440000 on 64-bit systems. It is used by `kprobe` and `ftrace` to place an instruction, which can be `call`, `jmp` or `int3`, to hijack the control flow [20, 14].

This project uses `capstone` disassembler [5] to disassembly the first 5 byte of the system functions to detect `kprobe`, `ftrace` and other inline hooks.

#### 3.3.4 *Check* `LD_PRELOAD` *libraries*

Statically linked programs are not affected by `LD_PRELOAD` [24]. This project implements a program that are compiled statically to check for the existence of the `LD_PRELOAD` environment variable and `/etc/ld.so.preload` file. This is just a small tool, not about analyzing memory dump.

## 4. EVALUATION

We conducted extensive experiments to evaluate the accuracy of our detection methods. Our setup involved various rootkit samples.

## 5. REFLECTION

### 5.1 eBPF Defender

The current eBPF Defender does not efficiently analyze suspicious eBPF installation programs. It only scans the binary file and extracts strings that resemble hook points, a method that can be easily evaded through string obfuscation. A more robust solution involves obtaining the BPF Type Format (BTF) information from the BPF binary and extracting meta-information from it. This allows users to make more informed decisions about whether to install the eBPF program.

Additionally, eBPF Defender has the potential to enhance its defensive capabilities. It can provide authorization for virtually all syscalls and prevent the installation of other types of malicious software, such as Loadable Kernel Modules (LKMs).

There is another a significant drawback of eBPF Defender, that it can only run on higher version of Linux kernel, so for rootkits that can run on lower version of Linux kernel, eBPF Defender is not applicable.

### 5.2 eBPF Detector

The current version of detector only does one thing. It goes through the syscall table and checks the system call boundary. It won't work if the rootkit uses eBPF to implement its functions, where it's not necessary to hook on system call by changing the syscall table. Also, if the rootkit hooks on other kernel functions, instead of system calls, checking the syscall table won't work.

The detector's behavior is also limited to the kernel version to some extent, the amount of system calls to check is written as 448, which is the highest used system call number on Linux 5.15.x kernel. Other versions of kernel, like 4.x, have different amount of system call, this will lead to the detector giving wrong warnings or miss some system call checking.

Like the defender, it also needs to run on a high version of Linux kernel, its compatibility can be further improved.

### 5.3 Memory Dump Analyzer

1. Checking only the first 5 bytes of a function may not detect hook instructions inserted in later parts of the function.

2. Cannot detect eBPF tracepoint.

3. Functions checked do not include vfs functions, so it is not possible to detect VFS layer rootkits, and the same applies to rootkits that hook other functions that are not in the `sys_call_table`.

4. Cannot detect rootkits that patch files on disk, such as patch dynamic linker's `LD_PRELOAD` rootkit like vleny [17].

# 6. CONCLUSIONS

In this paper, we presented three new tools for detecting and preventing Linux rootkits: the eBPF Defender, which hooks into syscalls to prevent unauthorized eBPF program installations; the eBPF Detector, which verifies the integrity of the syscall table; and the Memory Dump Analyzer, which identifies suspicious kernel hooks by analyzing dumped kernel memory. Our experimental results demonstrate that these methods effectively detect and prevent rootkit activities, showcasing the potential of eBPF to enhance Linux system security and the strength in analyzing memory dump. However, limitations such as the requirement for high kernel versions and the dependence on accurate symbol tables highlight areas for future improvement. By addressing these limitations, our techniques can provide even more robust defenses against rootkit-based malware.

# 7. GITHUB LINK

Welcome to visit our GitHub repository!
`https://github.com/LogicDX342/NUS-DOTA-Project`

# 8. REFERENCES


[1] 504ensicsLabs. GitHub - 504ensicsLabs/LiME — github.com. `https://github.com/504ensicsLabs/LiME`, 2024. [Accessed 17-07-2024].

[2] Akash7. Linux Loadable Kernel Module. `https://www.geeksforgeeks.org/linux-loadable-kernel-module/`, 2023. [Accessed 18-07-2024].

[3] M. S. Bajo. An analysis of offensive capabilities of ebpf and implementation of a rootkit. `https://raw.githubusercontent.com/h3xduck/TripleCross/master/docs/ebpf_offensive_rootkit_tfg.pdf`, 2022. [Accessed 19-07-2024].

[4] V. bluedragonsecurity's full-sized avatar BlueDragonSecurity. GitHub - bluedragonsecurity/bds_lkm_ftrace: Ftrace Based Linux Loadable Kernel Module Rootkit for Linux Kernel 5.x and 6.x on x86_64, hides files, hides process, hides bind shell & reverse shell port, privilege escalation, cleans up logs and bash history during installation — github.com. `https://github.com/bluedragonsecurity/bds_lkm_ftrace`, 2023. [Accessed 19-07-2024].

[5] Capstone Engine. GitHub - capstone-engine/capstone: Capstone disassembly/disassembler framework for ARM, ARM64 (ARMv8), Alpha, BPF, Ethereum VM, HPPA, LoongArch, M68K, M680X, Mips, MOS65XX, PPC, RISC-V(rv32G/rv64G), SH, Sparc, SystemZ, TMS320C64X, TriCore, Webassembly, XCore and X86. — github.com. `https://github.com/capstone-engine/capstone`, 2024. [Accessed 18-07-2024].

[6] carloslack. GitHub - carloslack/KoviD: Linux kernel rootkit — github.com. `https://github.com/carloslack/KoviD`, 2024. [Accessed 18-07-2024].

[7] cnwangjihe. GitHub - cnwangjihe/ebpf-rootkit: A simple rootkit written in ebpf. — github.com. `https://github.com/cnwangjihe/ebpf-rootkit`, 2023. [Accessed 18-07-2024].

[8] eBPF. ebpf documentation. `https://ebpf.io/what-is-ebpf/#ebpf-safety`, 2024. [Accessed 16-07-2024].

[9] eBPF.io. eBPF Documentation. `https://ebpf.io/what-is-ebpf/`, 2024. [Accessed 18-07-2024].

[10] M. Fleming. A thorough introduction to ebpf. `https://lwn.net/Articles/740157/`, 2017. [Accessed 19-07-2024].

[11] He1m4n6a. Anti-intrusion strategy summary - rootkit detection. `https://he1m4n6a.github.io/2019/07/21/%E5%8F%8D%E5%85%A5%E4%BE%B5%E7%AD%96%E7%95%A5%E6%80%BB%E7%BB%93-rootkit%E6%A3%80%E6%B5%8B/`, 2019. [Accessed 18-07-2024].

[12] X. hu, M. Huang, Y. Xue, L. Jiang, Y. Liu, and G. Xie. Drootkit: Kernel-level rootkit detection and recovery based on ebpf. *Journal of Circuits, Systems and Computers*, 33(04):2450073, 2023.

[13] M. Hutchins. Inline hooking for programmers (part 1: Introduction). `https://malwaretech.com/2015/01/`

inline-hooking-for-programmers-part-1.html, 2015. [Accessed 19-07-2024].

[14] M. H. Jim Keniston, Prasanna S Panchamukhi. Kernel Probes (Kprobes); The Linux Kernel documentation — docs.kernel.org. https://docs.kernel.org/trace/kprobes.html. [Accessed 18-07-2024].

[15] m0nad. GitHub - m0nad/Diamorphine: LKM rootkit for Linux Kernels 2.6.x/3.x/4.x/5.x/6.x (x86/x86_64 and ARM64) — github.com. https://github.com/m0nad/Diamorphine, 2023. [Accessed 18-07-2024].

[16] I. Maouda. Hunting Rootkits with eBPF: Detecting Linux Syscall Hooking Using Tracee. https://www.aquasec.com/blog/linux-syscall-hooking-using-tracee/, 2022. [Accessed 18-07-2024].

[17] mempodippy. GitHub - mempodippy/vlany: Linux LD_PRELOAD rootkit (x86 and x86_64 architectures) — github.com. https://github.com/mempodippy/vlany, 2019. [Accessed 19-07-2024].

[18] E. Metula. *Managed code rootkits: hooking into runtime environments*. Syngress, Burlington, MA, 1st edition, 2010.

[19] L. Miś. Linux ebpf as a double-edged sword. https://www.defensive-security.com/blog/linux-ebpf-as-a-double-edged-sword, 2023. [Accessed 16-07-2024].

[20] R. O'Neill. *Learning Linux Binary Analysis*, chapter Chapter 9: Linux /proc/kcore Analysis. Packt Publishing, Birmingham, England, Apr. 2023.

[21] reveng007. GitHub - reveng007/reveng_rtkit: Linux Loadable Kernel Module (LKM) based rootkit (ring-0), capable of hiding itself, processes/implants, rmmod proof, has ability to bypass infamous rkhunter antirootkit. — github.com. https://github.com/reveng007/reveng_rtkit, 2023. [Accessed 19-07-2024].

[22] rphang. GitHub - rphang/evilBPF: Weaponizing the Linux Kernel (Hide Files/PID, SSH backdoors, SSL Sniffer, ...) by poking around eBPF/XDP — github.com. https://github.com/rphang/evilBPF, 2024. [Accessed 19-07-2024].

[23] A. Taghikhani. Build a Custom Linux Profile for Volatility3. https://medium.com/@alirezataghikhani1998/build-a-custom-linux-profile-for-volatility3-640afdaf16 2023. [Accessed 19-07-2024].

[24] A. Thakur. Memory Malware Part 0x2 — Writing Userland Rootkits via LD_PRELOAD — compilepeace.medium.com. https://compilepeace.medium.com/memory-malware-part-0x2-writing-userland-rootkits-via-l 2020. [Accessed 18-07-2024].

[25] Tigera. ebpf explained: Use cases, concepts, and architecture. https://www.tigera.io/learn/guides/ebpf/, 2024. [Accessed 16-07-2024].

[26] veracode. Rootkit: What is a Rootkit and How to Detect It | Veracode — veracode.com. https://www.veracode.com/security/rootkit, 2024. [Accessed 18-07-2024].

[27] Volatility Foundation. Creating New Symbol Tables. Volatility 3 2.7.0 documentation. https://volatility3.readthedocs.io/en/stable/symbol-tables.html#mac-or-linux-symbol-tables, 2022. [Accessed 16-07-2024].

[28] Volatility Foundation. GitHub - volatilityfoundation/dwarf2json: convert ELF/DWARF symbol and type information into vol3's intermediate JSON — github.com. https://github.com/volatilityfoundation/dwarf2json, 2024. [Accessed 17-07-2024].

# DPKI Based Secure Communication Scheme For Self-driving Vehicle Networking*

Cao Yuan
Harbin Institute of Technology, Shenzhen
tsaoyuan@foxmail.com

Cao Li
Southern University of Science and Technology
12212752@mail.sustech.edu.cn

Hu Haojun
Harbin Institute of Technology, Shenzhen
hhj29132002@gmail.com

Hou Beijie
Shanghai Jiao Tong University
houbeijie@sjtu.edu.cn

Yang Jinqi
Xi'an Jiaotong University
jqyangxjtu@foxmail.com

## ABSTRACT

This paper provides a design of the authentication and communication mechanism for futuristic self-driving vehicles, which based on DPKI and blockchain techniques.

To avoid long winded certificates and hierarchical certification bodies in traditional PKI framework[1], we implement a decentralized PKI (DPKI) in vehicular authentication and communication networks to enhance security and ensure reliable, rapid identity verification, aligning with the dynamic needs of future smart transportation systems[2].

In the DPKI framework we have designed, individual information is decentralized and recorded as Distributed Identifiers (DIDs) on a blockchain. Department of Motor Vehicles(DMV) is the only institution with the right to write information about vehicles into the corresponding block of the blockchain so as to facilitate the subsequent query of identity authentication.

In order to resolve communication challenges in vehicular networks which similar to the "two generals problem" and fully utilize the road infrastructure in future smart cities, we endow Road Side Units (RSUs) with multiple functions, including identity authentication, convoy assignment, and providing robust communication security in dynamic vehicular environments. After RSU distributes group session key to the convoy, vehicles in the convoy can use this key and team members' public keys for encrypted communication.

## Keywords

Self-driving vehicles, Authentication, DKPI, Blockchain

*Paper from project of DOTA course, NUS SOC Summer Workshop 2024

## 1. INTRODUCTION

In recent years, with the rapid development of intelligent driving technology, self-driving vehicles have gradually entered the public eye.

Nowadays the successful deployment of the "Robot" self-driving vehicles called "Apollo Go" in Wuhan, China has become a significant milestone in the application of vehicular networks technology. Vehicular networks (V2X) facilitate the interconnection and communication between vehicles (V2V), vehicles and infrastructure (V2I), vehicles and networks (V2N), and vehicles and pedestrians (V2P), providing strong support for autonomous driving and intelligent transportation systems.

In actual traffic environments, the importance of information synchronization is particularly prominent. Take the phenomenon of phantom traffic jams, for instance. Phantom traffic jams refer to sudden traffic congestion that occurs without any apparent obstacle, caused mainly by abrupt deceleration or braking of vehicles, leading to a chain reaction of congestion. The occurrence of such phenomena is largely due to the lack of real-time information sharing and synchronization between vehicles. If each vehicle could promptly obtain the driving information of the vehicle in front, the probability of phantom traffic jams would be significantly reduced.

Another typical example is the green light start issue. When a traffic light turns green, usually the first vehicle starts, followed by the subsequent vehicles, resulting in lower overall traffic efficiency. If information synchronization among vehicles could be achieved through vehicular networks, all waiting vehicles could receive the green light signal simultaneously and start at the same time, thus greatly improving intersection traffic efficiency.

Therefore, the significance of vehicular networks technology lies not only in enhancing the intelligence of individual vehicles but also in optimizing the entire traffic system through information synchronization. This technology can not only reduce traffic congestion and improve road traffic efficiency but also effectively lower the accident rate, enhance driving safety, and provide a solid foundation for building intelligent

urban transportation systems.

## 2. BACKGROUND

In this section, we are going to introduce the key technologies used in our design, including blockchain, DPKI, and vehicle-to-vehicle symmetric session key establishment.

### 2.1 Blockchain Technology

Blockchain technology originated from Bitcoin as the underlying technology for which trust between multiple participants is established in a decentralized manner. The essence of blockchain is a decentralized distributed ledger database maintained by a peer-to-peer network (P2P). With the help of distributed storage technology, P2P network, consensus algorithm, chained data structure, digital signature, encryption algorithm and other cryptographic techniques to achieve the purpose of decentralization, non-tampering, anonymity, traceability and so on. As an innovative technology, blockchain has been rapidly developed and applied in various industries, with results such as Ether in the public chain, central bank digital currencies, Linux Foundation-supported Hyperledger, and so on.

The system model of blockchain technology can be roughly divided into six layers from top to bottom: application layer, contract layer, incentive layer, consensus layer, network layer, and data layer, as shown in Figure 1:

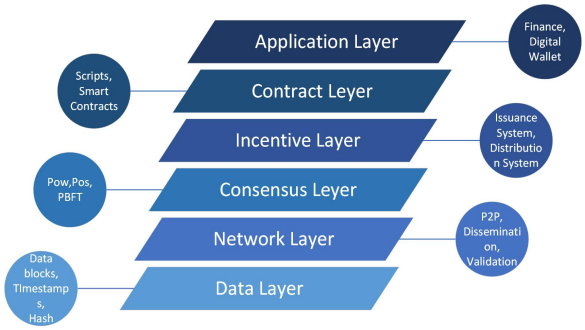



**Figure 1: Blockchain Architecture**

1. Application Layer: The application layer of the blockchain encapsulates various application scenarios and cases to provide users with various services and applications such as Finance and Digital Wallet.

2. Contract Leyer: The contract layer is responsible for encapsulating various programmable scripts, algorithms, and smart contracts to enable deterministic and automated execution of instructions. For example, a smart contract is a piece of code that is stored, verified, and executed on the blockchain so that it can be automatically executed without a third party if a defined constraint is met. By replacing humans with programmed algorithms to arbitrate and enforce contracts, this will save us huge costs of trust.

3. Incentive Layer: The incentive layer consists of a system for issuing and distributing economic incentives, which aims to provide certain incentives for network nodes to participate in the security verification of the blockchain, such as mining in Bitcoin.

4. Consensus Leyer: The use of consensus algorithms and consensus mechanisms to allow highly decentralized network nodes to reach a consensus in the blockchain network and decide which node can add a new block to the main chain is one of the core technologies of the blockchain.

5. Network Layer: The network layer mainly realizes the mechanism of distributed network through P2P technology, the network layer includes P2P networking mechanism, data dissemination mechanism and data verification mechanism. Due to P2P characteristics, when data is transmitted between nodes, even if some nodes or networks are destroyed, it will not affect the transmission of other parts.

6. Data Layer: The data layer mainly describes the physical form of the blockchain, which is the chain structure on the blockchain starting from the genesis block, and contains the block data, chain structure of the blockchain, as well as the random numbers, timestamps, public keys, private key data on the block, etc. It is the lowest data structure in the whole blockchain technology.

### 2.2 Decentralized PKI (DPKI)

DPKI represents an advancement and refinement of the traditional PKI framework. Current centralized PKIs suffer from an over-reliance on trusted central authorities (CAs), which undermines the autonomy of users in managing their identities. This centralized model introduces critical vulnerabilities, notably the susceptibility to single points of failure through erroneous certificate issuance by third parties, and susceptibility to Man-in-the-Middle (MITM) attacks, where deceptive CAs can falsely gain trust. Furthermore, the emergence of quantum computing has compromised the security of widely used cryptographic algorithms such as RSA and ECC, making them increasingly unreliable.

Given these concerns, the implementation of a decentralized PKI (DPKI) is crucial in vehicular communication networks to enhance security and ensure reliable, rapid identity verification, aligning with the dynamic needs of future smart transportation systems.

In the DPKI framework we have designed, individual information is decentralized and recorded as Distributed Identifiers (DIDs) on a blockchain. This setup allows users to independently manage and modify the content displayed within their own DIDs.

Third-party entities, such as roadway infrastructure or vehicle management organizations, can authenticate specific segments of a user's DID and log this information into their respective blocks. Under this system, the disclosure of all information is controlled by the individual, thus ensuring the privacy and security of vehicular data. Additionally, since third-party entities are responsible only for authenticating segments of data, the risk of a single point of failure is mitigated.

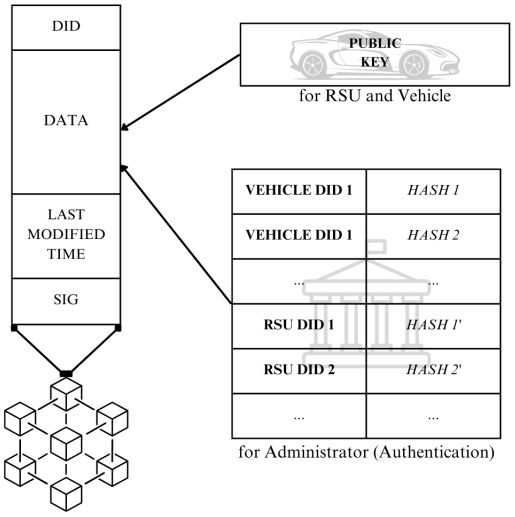



**Figure 2: DID Format**

Furthermore, the decentralized storage of all user blockchain data in roadside infrastructure devices makes Man-in-the-Middle (MITM) attacks impractical. Figure 2 shows the system architecture diagram of the format used for storing data on the blockchain.

## 2.3 Authentication and Key Establishment

In the context of vehicle-to-vehicle(V2V) communication, authentication and key establishment are two critical aspects for safeguarding the communication process against malicious adversaries. Authentication involves verifying whether the identities of the communicating parties are legitimate, serving as a prerequisite for secure communication and laying the groundwork for the establishment of session keys. Key establishment occurs after mutual authentication, where entities generate a symmetric encryption key through certain methods. By using the established key, the communication content is encrypted for transmission. Compared to asymmetric key communication, using symmetric session keys for communication is undoubtedly more efficient.

# 3. DESIGN

Our proposed design for solving phantom traffic jams incorporates several key components that collectively enhance communication, authentication, and synchronization within vehicular networks. This design addresses the "Two Generals' Problem" in vehicular communication by utilizing RSUs to ensure reliable message transmission, thus preventing infinite confirmation loops and enhancing overall system integrity and efficiency. The detailed design covers identity authentication, group assignment, secure communication, and the protocols for adding and exiting group members, providing a robust framework for dynamic vehicular environments,as shown in Figure 3.

## 3.1 Architecture Components

In our proposed design for solving the phantom traffic jam and enhancing the traffic flow, there are several important elements that form the framework of the whole program. The following is a detailed description of these elements:

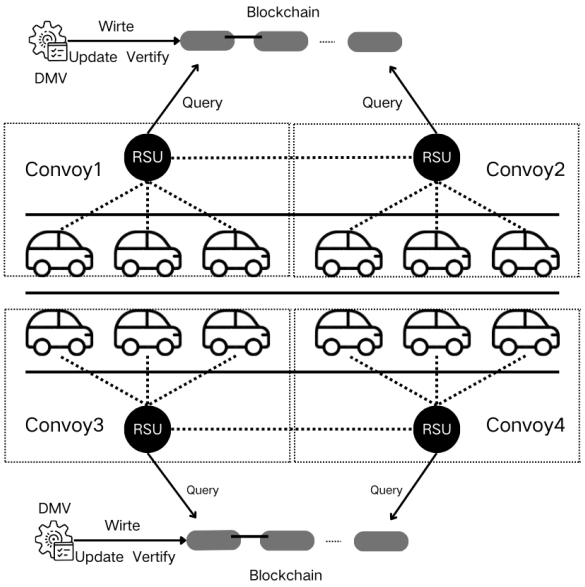



**Figure 3: Scene Design**

1. Vehicle: The vehicle is the basic unit of communication, authentication and information synchronization for the entire traffic and is the most important entity in our design solution. Vehicles are equipped with OBU (On board Unit) which is used to communicate with RSU (Road Side Unit) and other vehicles. The database inside the vehicle stores a large amount of information, such as the DID of the vehicle itself, relevant information about the vehicle, a table of information about the vehicles of the same group of teammates, and its own public and private keys. In addition, the vehicle has its own decision-making system for processing and acting on the information it receives. Since the decision-making and scheduling issues are out of the scope of our design discussion, we assume that the vehicle's decision-making system is absolutely secure and will not be damaged or receive attacks.

2. RSU(Road Side Unit): The road test units are distributed around the road and are set up at regular intervals.The RSUs will communicate with the vehicle wirelessly for authentication and grouping of the vehicle. At the same time, RSUs will also communicate with each other through wired communication for message delivery and synchronization between different groups. Since RSUs are set up by the government, it is assumed that RSUs are unattackable and communication with RSUs is absolutely trustworthy.

3. DMV(Department of Motor Vehicles): Each vehicle has to register the unique DID and related information of the vehicle in the corresponding DMV before leaving the factory, and the DMV will write the DID and the hashed related information into the corresponding block of the blockchain so as to facilitate the subsequent query of identity authentication. At the same time, each vehicle has to undergo annual inspection within the prescribed annual inspection cycle, and after passing the annual inspection, provide proof to the DMV, which will verify the vehicle information on the blockchain. If a vehicle fails the annual inspection or is not inspected in a timely manner, the DMV will periodically delete the information of the corresponding vehicle. In our design, we assume that the DMV is absolutely trustworthy and the information it writes is absolutely true and will not be tampered with.

## 3.2 Two Generals' Problem: the reason for RSU

Imagine a scenario where two armies are stationed at the north and south foot of a mountain, planning to attack an enemy force positioned on the mountain. Since neither army can breach the enemy's defenses independently, they decide to synchronize their attack. To coordinate, General A sends a messenger to convey the agreed-upon time of attack to General B. However, due to the presence of enemy ambushes along the route, it remains uncertain whether B has received the message. If General B does receive the message, he must then confirm receipt back to General A. Yet, this introduces a new uncertainty—General B does not know if his confirmation has reached General A, necessitating a return confirmation from A. This recursive process of confirming receipt could theoretically extend indefinitely, as the last general to send a message cannot be sure it was received.

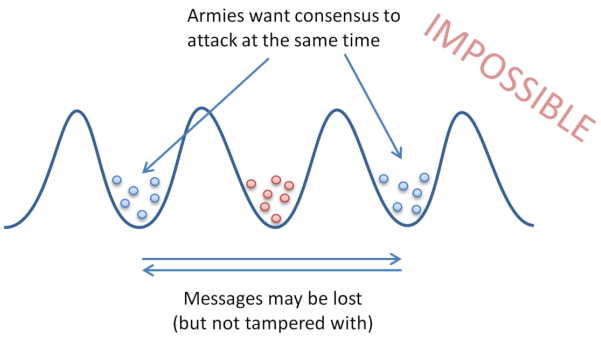



**Figure 4: Two Generals's Problem**

Similarly, communication within a fleet of vehicles encounters analogous challenges. Assuming that the communication channels between vehicles are unreliable, the scenario of perpetual confirmations arises during the synchronization process. To address this, we introduce the use of Road Side Units (RSUs). RSUs, capable of transmitting signals at higher power or making targeted optimizations for signal reception and transmission, effectively render communication between the RSU and vehicles as highly reliable. This transformation allows us to convert an unreliable communication channel into two reliable ones, thereby providing an engineering solution to the classic "two generals problem."

The deployment of Road Side Units (RSUs) effectively resolves the communication challenges in vehicular networks, similar to the "two generals problem." As reliable intermediaries, RSUs enhance the integrity and efficiency of vehicle-to-vehicle and vehicle-to-infrastructure communications by preventing infinite confirmation loops. Consequently, a critical function of RSUs is to provide robust communication security in dynamic vehicular environments.

## 3.3 Detailed Design

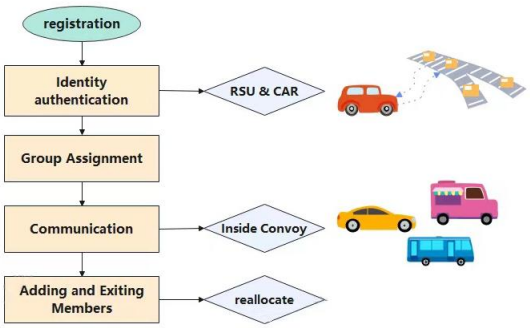



**Figure 5: Design**

### *3.3.1 Identity Authentication*

This stage contains mutual authentication between vehicles and RSUs. Suppose vehicles have finished registration. In the authentication stage, vehicles first verify RSUs' identity by checking its signature in broadcast messages. Once verified, vehicle send a group assignment request and using vehicle's private key to generate a signature.

For each vehicle's request, RSU first search the blockchain trying to find a block recording this vehicle's identity. Once found, compute the Hash value of vehicle's public key and compare with value in the block. If consistent, authentication passed.

RSU performs batch verification on a set of request messages with signatures to complete the identity authentication of a group of vehicles.

### *3.3.2 Group Assignment*

In this stage, RSU computes and sends group session key to vehicles assigned in the same group. RSU first uses all the DIDs of one group and a random number seed to generate a group session key for this group[3]:

$$K_{Group} = F\{DIDs, seed\}$$

Then in the key-distribution stage, RSU encrypts the session key with vehicles' public keys and sends to corresponding cars respectively. When vehicles receive message, they decrypt it with private key to obtain group session key. Meanwhile, RSU tells every vehicle some information about their teammates.

Vehicles with the same group session key belong to a common group. This process can be analogized to WeChat's face-to-face Group building, where the session key is the same number chosen.

### 3.3.3 *Communication*

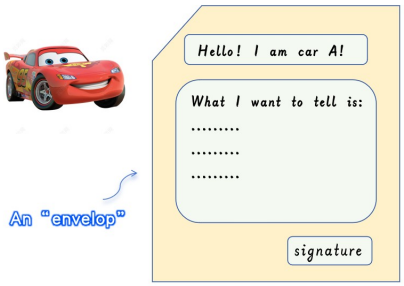



**Figure 6: An Analog Of Message Envelop**

If car A wants to communicate with car B in the same convoy, let's say, message is m. A should give a digital signature to this message. The specific way is to compute the digest of m using Hash function, then encrypt the digest with A's private key. A then encapsulates who it is, message m, and A's signature in an "envelop" and encrypts the "envelop" using group session key. After that, A send this whole "envelop" to car B. An analog of message envelop is like Figure 6. The specific encryption process is shown in Figure 7.

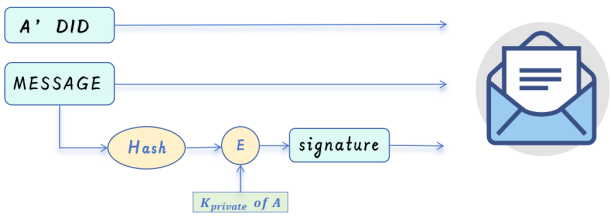



**Figure 7: Composition Of The Envelope**

When car B receive this envelop, B first decrypts it with session key, then B will get A's "name", message, followed by A's signature. B can verify message source by decrypt signature with A's public key and compare the results with the hash value of message.

### 3.3.4 *Adding and Exiting Members*

Before a car gets on the road, it needs to search the nearby RSU. In detail, the new car broadcast a request to check whether there are RSU nearby.After verification the new car's identity, every RSU that receives the message responds to it. The reply message contains its group number and other necessary information. If the car receive more than one responses,it will choose the first one as it is either the closest or the fastest. The car then send a new request to the chosen RSU, telling that it want to get a group identity.

The RSU reply with group identity for the new car,using which the new car can communicate with other cars in the same group, along with a list of other cars information in this group.The RSU also send a notification message to all the cars in this group except the new car, which contains information of the new car. After receiving this message, other car add the new car's information to its member list. As shown in the Figure 8, after the 4-way handshake, the new car will formally join a group.And the RSU will begin to server for this car.

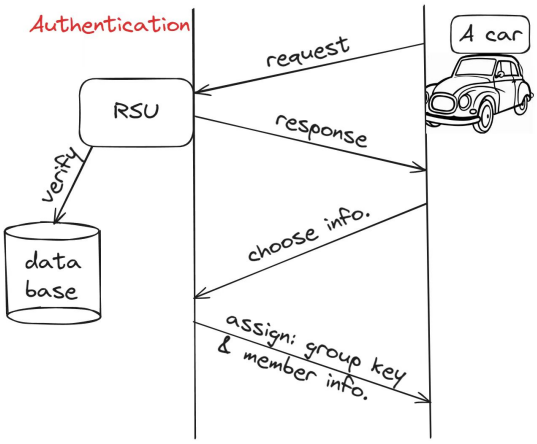



**Figure 8: Adding Members Handshake**

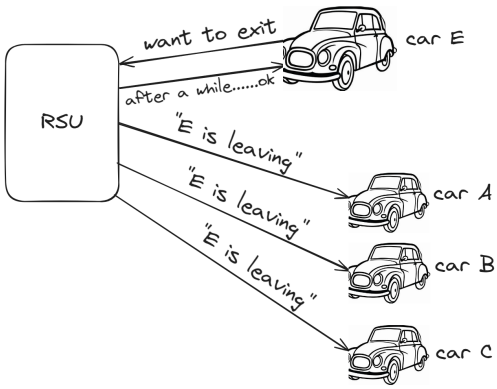



**Figure 9: RSU Response When Vehicle Exits**

Leaving the convoy is a bit more complicated.

In general, since the RSU has fixed position, the position shift of the vehicle may trigger the exit and re-entry. Since the vehicle needs to maintain communication with the RSU while driving all the time, leaving the convoy triggers a handover process.

Each RSU has a area called marginal region. Just like its name, marginal region is the area relatively far away from RSU center, but also under the control of RSU. Two RSU can have overlapping marginal region. When car enters the marginal region, it is probably going to leave the convoy. Firstly, the car sends a message to RSU expressing it wants to leave and its original RSU finds the next RSU that may provide service based on the direction it is traveling. The

original RSU sends a message to new RSU, info it that a new car will enter it's area.

At the meanwhile, RSU sends the leaving car information to new RSU. The leaving car starts a 2-way handshake,same as the last two steps of 4-way handshake above. When the new RSU accepts the vehicle, it notifies the original RSU.

The original RSU then delete the info of that vehicle,and send member-exit message to all its existing members. Existing members simply delete its relative info and will be unable to communicate to that car directly.

Eventually, the car officially dropped out of the group that it was previously in.

# 4. CONCLUSIONS

In this paper we propose a secure communication scheme for self-driving vehicle Networking. The integration of Decentralized Public Key Infrastructure (DPKI) and blockchain technology presents a groundbreaking approach to addressing the authentication and communication challenges inherent in the realm of self-driving vehicles. This shift towards decentralization not only streamlines the authentication process but also significantly enhances the security and reliability of vehicular communication networks, thereby meeting the dynamic demands of future smart transportation ecosystems.

Within our proposed DPKI framework, the utilization of Distributed Identifiers (DIDs) recorded on a blockchain ensures a decentralized storage of individual vehicle information. This innovative approach empowers entities such as the Department of Motor Vehicles (DMV) with the capability to record pertinent vehicle details directly onto the blockchain, facilitating efficient identity verification processes.

Furthermore, our design addresses the communication challenges akin to the "two generals problem" in vehicular networks by equipping Roadside Units (RSUs) with advanced functionalities. This strategic enhancement not only optimizes the utilization of road infrastructure in smart cities but also paves the way for seamless and secure vehicular communication, marking a significant advancement towards the realization of efficient and safe autonomous transportation systems.

# 5. REFERENCES


[1] Qingsen Zhu, Research on Security Authentication in UAV Ad Hoc Networks, pages 2-3, April 2023.

[2] Christopher Allen, Arthur Brock,Vitalik Buterin,Jon Callas,Duke Dorje,Christian Lundkvist,Pavel Kravchenko,Jude Nelson,Drummond Reed, Markus Sabadello,Greg Slepak, Noah Thorp,and Harlan T Wood, Decentralized Public Key Infrastructure, A White Paper from Rebooting the Web of Trust, pages 7-9, 2015.

[3] Zhiwang He,Blockchain-based Internet of Vehicles Authentication Scheme, pages 24-34, April 2023.

# Quantum Guard
## A Quantum Resistant Blockchain Model

Wang Shirui
Huazhong University of Science and Technology
wangshirui53@gmail.com

Xu Jingbo
Harbin Institute of Technology
xujingbo172@gmail.com

Radhika Raina
Vellore Institute of Technology
radhika.raina@outlook.com

Liu Yilei
University of Electronic Science and Technology of China
liuyolanda@163.com

Wang Jiamin
Beijing University of Posts and Telecommunications
wjm5490@gmail.com

## ABSTRACT
In today's digital landscape, blockchain technology is revolutionizing the way trust and transactions are managed across various sectors.Blockchain's decentralized structure and cryptographic methods ensure data integrity, security, and transparency. This paper investigates the incorporation of quantum-resistant algorithms into blockchain systems. It highlights the need for Post-Quantum Cryptography (PQC) methods, such as lattice-based and multivariate polynomial techniques, to secure blockchain technology.It also provides a comprehensive review of the cryptographic algorithms used in Bitcoin and Ethereum, and examines post-quantum algorithms like BLISS and the McEliece encryption scheme.The paper presents a quantum-resistant blockchain model featuring SPHINCS+-SHAKE256f-robust for digital signatures and employs a Parameter-Based Quantum Resistant Hash Function (PQH) for hashing. This model is designed to strengthen blockchain against quantum threats while balancing security and performance. The findings suggest that incorporating PQC techniques can significantly improve blockchain security, offering a robust framework for future-proof digital ledgers.

## KEYWORDS
Blockchain technology,Quantum-resistant cryptography,Post-Quantum Cryptography (PQC),Bitcoin,Ethereum,Quantum computing

## 1. INTRODUCTION
Blockchain technology is transforming trust and transactions across various sectors by providing a decentralized, distributed, and immutable database. It ensures data integrity through a sequential chain of blocks, each containing a hash of the previous block, making alterations or deletions difficult and maintaining a transparent, secure, and traceable digital environment.

Decentralization allows anyone to join the blockchain network and verify transactions without prior trust in others [1]. Blockchain security relies on cryptographic techniques: hashing functions compress transaction data into unique hash values, while public-key cryptography creates digital signatures to verify authenticity and prevent tampering [28]. Algorithms like RSA, ECDSA, ECDH, and DSA are commonly used, with ECDSA being prevalent in blockchain applications. Their security is based on hard mathematical problems, resisting brute-force attacks.

However, advancements in quantum computing, through algorithms like Grover's and Shor's, could compromise traditional encryption methods [20]. This highlights the need for quantum-resistant algorithms. Research into Post-Quantum Cryptography (PQC) is underway, exploring lattice-based and multivariate polynomial approaches to secure blockchain against quantum threats.

This article examines the use of quantum-resistant algorithms in blockchain, including an overview of blockchain technology, security measures, and performance criteria. It analyzes traditional and quantum algorithms, compares current encryption schemes, and evaluates the performance of emerging post-quantum cryptosystems in blockchain nodes. Key findings and conclusions are summarized at the end.

## 2. CRYPTOGRAPHY FOR SECURE BLOCKCHAIN

### 2.1 Foundations of Blockchain
Blockchain is a decentralized digital ledger that ensures secure, transparent, and immutable record-keeping through cryptographic methods such as hash functions, public-key encryption, and digital signatures. Public blockchains like Bitcoin and Ethereum are open and decentralized, whereas private blockchains are restricted for enhanced security. These cryptographic methods ensure that once data is added to the blockchain, it remains immutable and verifiable [4, 9].

## 2.2 Performance

Blockchain performance depends on consensus mechanisms and cryptographic algorithms. Proof of Work (PoW) is secure but computationally demanding, while Proof of Stake (PoS) is more energy-efficient. Alternatives like Proof of Activity, Proof of Capacity, and Proof of Burn offer various balances of security, scalability, and decentralization. Advances in cryptography and AI/ML could further optimize these mechanisms for better performance [16].

## 2.3 Security

Key cryptographic techniques for blockchain security include:

- Hash Functions: Ensure block integrity (e.g., Bitcoin's SHA-256, Ethereum's Keccak-256) [22].
- Public and Private Keys: Facilitate secure transactions through asymmetric encryption and digital signatures [5, 22].
- Consensus Algorithms**:** Maintain ledger consistency with methods like PoW, PoS, and PBFT, focusing on efficiency and security [9, 16].
- Privacy-Preserving Techniques: Zero-knowledge proofs and ring signatures protect anonymity and transaction confidentiality [9, 23].

Advanced cryptographic methods, such as multi-signature schemes, homomorphic encryption, and threshold cryptography, further enhance blockchain security and privacy across various applications [23].

## 2.4 Vulnerabilities

Blockchain faces several vulnerabilities:

- 51% Attacks: Occur when a single entity controls over half the network's power, potentially altering the blockchain. This risk is mitigated by decentralized consensus and larger key sizes [5].
- Sybil Attacks: Involve creating multiple false identities to take over the network. Strong identity verification and consensus mechanisms help mitigate this [9].
- Quantum Computing: Could potentially break current cryptographic algorithms. Post-quantum cryptography and hybrid solutions are being developed to address these future threats [15, 22].
- Smart Contract Vulnerabilities: Bugs in smart contracts, as seen in The DAO hack and Parity wallet incident, can cause significant losses. Rigorous security practices and tools like OpenSSL and Mythril are vital for detection and mitigation [22].

# 3. EXISTING ALGORITHMS AND MODELS

This section introduces cryptographic algorithms currently used in blockchain, focusing on Bitcoin and Ethereum, as well as some postquantum cryptographic algorithms and their potential applications.

## 3.1 Algorithms in Traditional Blockchain

*3.1.1 Algorithms in Bitcoin.* Bitcoin employs SHA-256 (Secure Hash Algorithm 256-bit) as its core hash function and ECDSA (Elliptic Curve Digital Signature Algorithm) for digital signatures. SHA256 generates block, transaction, and Merkle tree hashes through multiple rounds of encryption, producing a unique 256-bit hash value [25]. In Bitcoin's proof-of-work system, miners calculate SHA256 hashes of the block header to find one that meets the difficulty target. This process demands substantial computational resources and helps maintain the security and decentralization of the blockchain.

Bitcoin employs ECDSA with the secp256k1 curve to create and verify transaction signatures. This elliptic curve algorithm offers security comparable to RSA but with shorter keys and greater efficiency. Network nodes verify these signatures with public keys to ensure the authenticity of transactions and prevent tampering [11].

*3.1.1 Algorithms in Ethereum.* Ethereum uses SHA-3 (Keccak-256) for hashing, which provides robust resistance to collision and preimage attacks through its sponge construction [3, 26].This hash function ensures the integrity and immutability of transactions and smart contracts.

Additionally, Ethereum relies on ECDSA for verifying transaction signatures and RLP encoding for efficient data transmission [12].These cryptographic techniques support Ethereum's smart contracts by maintaining code integrity and data immutability, thereby securing and decentralizing the ecosystem [29].

## 3.2 Post-Quantum Cryptography in Blockchain

Post-quantum cryptography consists of algorithms designed to withstand quantum computer attacks, based on distinct mathematical problems that offer higher security than traditional methods. Key types include lattice-based (e.g., BLISS [8], NTRU [14]), codebased (e.g., McEliece [19]), multivariate polynomial-based (e.g., Rainbow [6]), and isogeny-based algorithms (e.g., SIDH). These algorithms not only resist quantum attacks but also show promising performance and application prospects. We will explore two representative algorithms as examples.

*3.2.1 Bimodal Lattice Signature Scheme.* The Bimodal Lattice Signature Scheme (BLISS) offers strong quantum resistance by utilizing lattice problems like the Shortest Vector Problem (SVP) and Closest Vector Problem (CVP). It generates signatures using Gaussian sampling and a bimodal distribution, making it suitable for blockchain transactions and smart contracts while resisting side-channel attacks [8].

*3.2.2 McEliece Encryption Scheme.* The McEliece scheme, based on coding theory, provides quantum resistance through the complexity of decoding Goppa codes. It involves generating a public Goppa code and a private decoder. Despite its large key sizes and lower encryption efficiency, it offers robust security against quantum

algorithms like Shor's and Grover's, making it a viable option for post-quantum blockchain applications [19].

# 4. PROPOSED MODEL

We propose a blockchain model that is resilient to the computational power of quantum computers. Our model integrates SPHINCS+SHAKE256f-robust for digital signatures, renowned for its high security and robustness with a 512-bit internal capacity. While slightly slower than alternatives like SHA2 and Haraka, SPHINCS+SHAKE256f-robust prioritizes security in the quantum computing era.

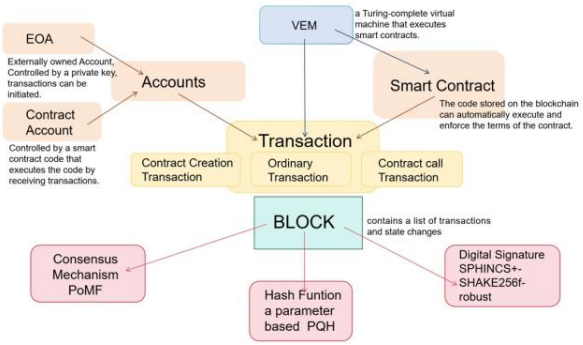



**Figure 1: Blockchain Scheme**

For hashing operations, we implement a parameter-based quantumresistant hash function (PQH) that resists collision attacks and ensures strong avalanche properties. To achieve decentralized consensus, our model employs a weighted multi-factor quantum consensus mechanism (PoMF), integrating multiple factors to evaluate consensus formation securely.

In our algorithm selection, security takes precedence over performance, though we aim for optimal performance under equivalent security levels. The proposed blockchain's data structure includes hash parameters for consensus, verification information for PoMF, and secure handling of signatures and public keys using SPHINCS+SHAKE256f-robust. This structure enhances scalability and versatility with optional control items in transactions.

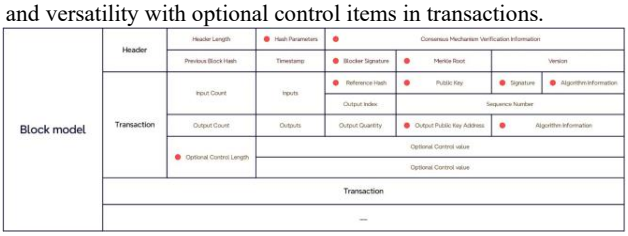



**Figure 2: Data Structure of the Proposed Model**

## 4.1 SPHINCS+-SHAKE256f-robust

Given the advancements in quantum computing, where the speed difference between signature and verification has become negligible, our primary focus is on maximizing security. The standout feature of our proposed blockchain model is its exceptionally high security level. Based on Aumasson et al.'s analysis[2], we assessed nearly all existing post-quantum cryptography (PQC) signature algorithms and identified SPHINCS+, Dilithium, and qTESLA as the top performers in terms of both security and performance for further evaluation.

| Cryptosystem | SubType | Claimed Quantum Security | Public Key Size(bytes) | Private Key Size(bytes) | Signature Size (bytes) |
|---|---|---|---|---|---|
| DILITHIUM 1280x1024 SHAKE | Lattice-based | 128 bits | 1472 | - | 2701 |
| qTESLA-p-III | Lattice-based | 192 bits | 38432 | 12352 | 5664 |
| SPHINCS+-SHAKE-256f | Hashed-based | 256 bits | 64 | 128 | 49856 |

**Figure 3: Security & Space**

| | | Performance(Reference Implementation) | | | Performance(Optimized AVX2 ) | | |
|---|---|---|---|---|---|---|---|
| Algorithm | Evaluation Platform | Key Generation (Cycle) | Signing (Cycle) | Verification (Cycle) | Key Generation (Cycle) | Signing (Cycle) | Verification (Cycle) |
| DILITHIUM 1280x1024 | Intel Core-i7 6600U(Skylake)CPU@2.6GHz | 371,083 | 1,562,215 | 375,708 | 15,677 | 437,638 | 155,784 |
| qTESLA-p-III | Intel Core-i7 6600U(Skylake)CPU@2.6GHz | 13,726,600 | 6,284,600 | 1,830,400 | - | - | - |
| SPHINCS+-SHAKE-256f | Intel Core-i7 4770K CPU@3.5GHz | 82,842,862 | 1,046,811,244 | 20,876,946 | 2,510,894 | 65,870,866 | 1,949,510 |

**Figure 4: Speed**

SPHINCS+ excels in quantum security by minimizing both public and private keys, reducing computational and storage overhead compared to alternatives. Unlike the more complex implementations of Dilithium and qTESLA, SPHINCS+ is straightforward to implement, making it suitable for resource-constrained blockchain environments. This simplicity positions it as an ideal solution for digital signature applications where high security is prioritized over speed Refer to the Appendix section C for more information.

*4.1.1 Security Evaluation.* Our focus is on SPHINCS+-256f, the highest security variant of SPHINCS+, offering a security level of 5. This self-signed scheme relies on robust hash functions and incorporates multi-layer Merkle trees and multiple signatures (FORS and WOTS+), making it highly resistant to quantum attacks. These features significantly increase the difficulty of potential attacks by requiring compromise of multiple independent signature steps and hash functions.

Quantum computers, with algorithms like Shor's, threaten encryption methods based on factorization and discrete logarithms (e.g., RSA and ECC). However, hash-based schemes like SPHINCS+ are naturally resistant to such attacks.

*4.1.2 WOTS+.* WOTS+ signatures are essential for validating and signing FORS signatures, ensuring their integrity and security. The one-time signature feature of WOTS+ guarantees that each signature is unique and used only once, enhancing the overall security of the signature scheme. For detailed information refer to the Appendix section A.

*4.1.3 Hypertree.* The Hypertree employs a Merkle tree structure to organize all WOTS+ signatures into a hierarchical format. Internal nodes and eventual root nodes are generated by hashing leaf nodes. Each layer's WOTS+ key pair signs the root nodes of the subsequent layer, thereby guaranteeing the

authenticity and integrity of each layer's root nodes. For detailed information refer to the Appendix section B.

*4.1.4 Forest Of Random Subset.* FORS signatures are generated at the initial stage of the signing process, distributing the signature of the message across multiple small pieces. This decentralization ensures that each piece possesses an independent signature, enhancing overall security.

*4.1.5 SPHINCS+ Algorithm.* SPHINCS+ integrates FORS, WOTS+, and the Hypertree structure. It also incorporates randomized message compression and verifiable index generation [10].

Algorithm Structure:

(1) Message blocking: Messages $M = \{m_1, m_2, \ldots, m_n\}$ are divided into blocks.
(2) FORSsignaturegeneration: Generate FORS signature FORS_Sigi for each message block $m_i$, serving as input for subsequent WOTS+ signatures.
(3) WOTS+ signature generation: Generate WOTS+ signature WOTS+_Sigi for each FORS signature FORS_Sigi.
(4) Merkle tree creation: Construct a Merkle tree where leaf nodes store WOTS+_Sigi.
(5) Hashing and root node generation: Compute hash values of internal nodes to generate the root node.
(6) Root node signing: Sign the root node with the WOTS+ key pair of the upper layer. Each node in the Merkle trees represents a WOTS+ public key.

The root of each Merkle tree layer is signed by the WOTS+ key pair from the layer above it,establishing a chain that culminates in the topmost root, which represents the public key of the SPHINCS+ scheme.

This design ensures the independence and immutability of each signature while leveraging the Merkle tree structure to enhance verification efficiency and adaptability to large-scale signature requirements.

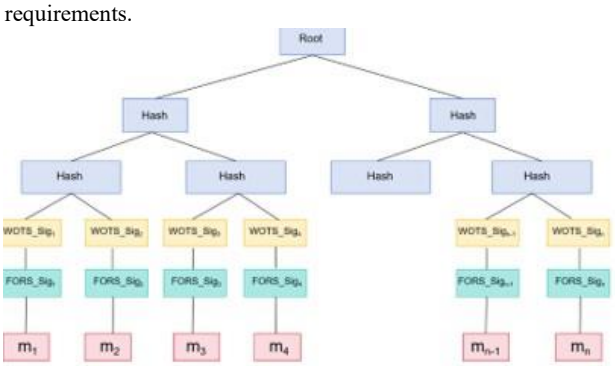



**Figure 5: Merkle Tree Structure**

| | key generation | signing | verification |
|---|---|---|---|
| SPHINCE+-SHAKE-256f-simple | 36 950 136 | 763 942 250 | 19 886 032 |
| SPHINCE+-SHAKE-256f-robust | 72 503 094 | 1 467 095 732 | 39 555 542 |
| SPHINCS+-SHA2-256f-robust | 62 599 672 | 1 312 473 846 | 37 139 082 |
| SPHINCS+-Haraka-256s-robust | 1 077 657 774 | 15 566 614 908 | 22 516 650 |
| Runtime benchmarks for SPHINCS+ | | | |
| | key generation | signing | verification |
| SPHINCE+-SHAKE-256f-simple | 8 535 534 | 176 951 378 | 5 030 988 |
| SPHINCE+-SHAKE-256f-robust | 16 296 098 | 329 696 258 | 9 716 888 |
| SPHINCS+-SHA2-256f-robust | 21 327 470 | 435 984 168 | 14 938 510 |
| SPHINCS+-Haraka-256f-robust | 2 599 368 | 61 706 762 | 1 854 540 |

**Figure 6: Runtime Benchmarks**

*4.1.6 Performance.*

Runtime

When instantiating, comparisons were made between SHAKE, SHA2, and Haraka [10]:

- SHAKE256: Used in SPHINCS+-SHAKE, it features a 512-bit internal capacity. It shows resilience against generic quantum attacks due to its sponge construction, but its second-preimage resistance is limited by the $n$-byte output length.
- SHA2: Implemented in SPHINCS+-SHA2, SHA2 has a 256-bit chaining value. SHAKE256 with a 256-bit output sometimes outperforms SHA2 due to its enhanced capabilities.
- Haraka: Although Haraka offers speed benefits, it lacks NIST certification and requires additional security evaluation due to its recent introduction.
- Simple implementations are faster but rely on security arguments applicable only in the random oracle model. In contrast, robust implementations adopt a more conservative security stance, though they are slower [10].

Space

| | public key size | secret key size | signature size |
|---|---|---|---|
| SPHINCS+-128f | 32 | 64 | 17 088 |
| SPHINCS+-192f | 48 | 96 | 35 664 |
| SPHINCS+-256f | 64 | 128 | 49 856 |

**Figure 7: Key Signature Sizes in Bytes**

## 4.2 A Parameter Based Quantum Resistant Hash Function (PQH)

Due to Grover's algorithm [13], quantum computers could reduce the complexity of hash collisions and decryption by a square root, posing a significant threat to traditional hash functions. Hash functions are crucial for blockchain technology, linking blocks and maintaining system integrity. Collisions could allow attackers to replace block information undetected, compromising the blockchain.

The basic idea of finding a collision is to find a $p2$ such that:

$$H(p1) = H(p2)$$

where $p1$ is the original plaintext, and $p2$ is the collision. Evaluating a hash function involves assessing its collision resistance, digest length, and computational complexity.

PQH is a quantum-resistant hash function based on a theoretically unsolvable mathematical problem (MHP) [17]. It incorporates parameters making collision finding algorithmically infeasible. The principle involves solving equations with:

$$\sum f_i \left(\sum a_{ij} x_j\right) = 0, i \in (1,m), j \in (1,n), m \ll n$$

Refer to the Appendix section D for details.

*4.2.1Algorithm Analysis.* This algorithm has a time complexity of $O(mn^2)$ and a space complexity of $O(mn)$. The parameter $p$ constrains matrix values and determines block size. For efficiency, $p$ should be large and close to but not exceeding a power of 2. In our implementation, $p$ is set to 52,711, resulting in nearly a 16-bit block size. The final digest size (DS) in bits is given by:

$$DS = (\log_2 P) \times n$$

*4.2.2 Algorithm Benchmark.* See Appendix E for benchmarks. Results confirm the expected linear increase in time with input size and quadratic increase with digest size. PQH shows a significant performance gap compared to traditional algorithms, but offers a reasonable trade-off between security and performance.

*4.2.3 Communication Protocol.* Parameters must be included and hashed for security. Regular shifting of the initial vector (IV) and a scheme for random IV generation are recommended.

*4.2.4 Security Discussion.* PQH's reliance on an unsolvable problem theoretically prevents collision attacks. The algorithm shows a 45% avalanche effect, close to the 47-50% range, making intentional collisions highly unlikely.

*4.2.5 Potential Refinements.* PQH shows strong collision resistance and avalanche effects. Performance could be improved by compressing input text and reducing $m$.

*4.2.6 Conclusion.* PQH provides excellent security and performance, making it suitable for the post-quantum blockchain model, especially in the early quantum era. It offers flexibility in input and digest sizes.

## 4.3 A Weighted Multi-factor quantum consensus mechanism (PoMF)

Designing a robust blockchain for a post-quantum world requires a novel consensus mechanism. Traditional methods like Proof of Work (PoW) [21] and Proof of Stake (PoS) [24] may be compromised by quantum computing. PoMF is designed to ensure decentralization, speed, and fairness, incorporating voting, randomness, and grading functions.

*4.3.1 Ideas and Principles.* We propose a new mechanism designed to ensure decentralization and introduce a novel way of designing consensus mechanisms in blockchain. This weighted multi-factor quantum consensus mechanism incorporates voting and randomness elements. It is based on several grading functions and a weighting function. In this chapter, we will discuss the principles of these functions. Implementing such a mechanism requires selecting appropriate functions judiciously and cautiously. In the proposed mechanism, important factors include stake, history (reputation), activeness, and randomness. Each factor is calculated and then synthesized by a weighting function into a single quantified score, which determines the blocker. Based on our proposal, the corresponding factors should be calculated as: - Stake (S) - History (H) - Activeness (A) - Randomness (R) The weighting function is given as:

$$W(S, H, A, R)$$

For detailed expectations of the corresponding functions, please refer to Appendix section F.

*4.3.2 Discussion of Performance, Security, and Features.* PoMF is expected to be faster than PoW but slower than PoS or Proof of Burn (PoB) due to its scoring algorithms. The mechanism is adaptable, with features like incentives, timeouts, and penalties, and is resistant to quantum attacks as it doesn't rely on computational resources.

*4.3.3 Conclusion.* The proposed consensus mechanism provides a hybrid-patterned method to evaluate and reach consensus. PoMF uses distinct factors with a weighting function to quantify the trustworthiness of each participant. Every grading is verifiable among all participants, ensuring security and efficiency. Moreover, this mechanism is resistant to quantum attacks since it does not rely on computational resources or any other kind of physical resources.

# 5. REFLECTION

## 5.1 Purpose review

The purpose of the proposal for our model is to offer a possible quantum resistant blockchain solution. In our model, quantum security is the top concern, which means the blockchain must not be compromised even under threat of extensive computational resources of quantum computer. After security, we focus on assuring decentralization and fairness. In addition, we cautiously choose algorithm and block data structure to achieve optimal performance in time consumption and storage demand.

## 5.2 Components comparison

In this part, we briefly give comparison between typical implementation and ours.

| Signature Algorithm | Claimed quantum security (bit) | Public Key Size (byte) | Private Key Size (byte) | Signature Size | Signing Time consumption | Verification Time Consumption | Draw backs |
|---|---|---|---|---|---|---|---|
| SPHINCS+-SHAKE256 | 256 | 64 | 128 | 49856 | Slow | Fast | Large signature sizes |
| ECDSA (Bitcoin, Ethereum) | No | 64 | 32 | 64 | Medium | Medium | Security |
| ED25519 (Solana) | No | 32 | 64 | 64 | Fast | Fast | Security |

**Figure 8: Signature Functions**

| Algorithm | Theoretical basis | Time consumption | Quantum resistant |
|---|---|---|---|
| PQH | Mathematical Unsolvable Problems | Higher | Yes |
| SHA-256 (Bitcoin) | Merkle-Damgård Construction (**Reference** The cryptographic hash function. SHA-256 CRIPTOGRAFIA MAII - FIB) | Lower | No |
| Keccak-256 (Ethereum) | Sponge Construction (**Reference** Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. Keccak and the SHA-3 Standardization. NIST ) | Lower | No |

**Figure 9: Hash Functions**

| Consensus mechanism | Throughput | Decentralization | Drawbacks | Resource reliability |
|---|---|---|---|---|
| PoMF | High | Highest | Mathematical function computation overhead | Lowest |
| PoW (Bitcoin) | Lowest | Lowest | Energy consumption | Highest |
| PoS (Ethereum) | Highest | Lower | Centralization | Higher |

**Figure 10: Consensus Mechanisms**

The block structure is generally composed based on the implementation of "Bitcoin". However, some components are modified or added to fit in algorithms used. Added components includes "Hash parameters", "Consensus Mechanism Verification Information", "Blocker Signature", "Algorithm Information". Other modified components include "Merkle Root", "Reference Hash", "Output Public Key Address". In general, block size has been increased to implement mentioned algorithm to ensure security, which will influence performance.

## 5.3 Overall analysis

Bitcoin's Proof of Work (PoW) demands significant resources, while Ethereum's Proof of Stake (PoS) offers energy efficiency but risks centralization. Solana combines Proof of History (PoH) with PoS to enhance decentralization. Our Weighted Multi-Factor (PoMF) consensus mechanism incorporates these advantages, but quantum computing poses a threat to PoW and PoS by efficiently performing hash collisions and reverse computations.

Grover's algorithm reduces the collision difficulty of SHA-2 and Keccak-256 hashes, creating an urgent need for more secure hashing methods. Similarly, Shor's algorithm can break elliptic curve signatures like ECDSA and ED25519 by factoring large integers.

Our "QuantumGuard" scheme addresses these issues with SPHINCS+-SHAKE256f for secure, hash-based signatures resistant to quantum attacks. It uses a parameter-based Quantum Hash Function (PQH) to balance security and performance. The PoMF consensus mechanism integrates Stake (S), History (H), Activeness (A), and Randomness (R) to enhance decentralization and adaptability. This solution is designed to withstand quantum threats, ensuring blockchain integrity and privacy while supporting smart contracts and scalability for diverse applications.

## 5.4 Discussion for improvements

Our model meet with the need for a quantum resistant blockchain. It has achieved great security against quantum computer. However, the model compared to typical implementation has following disadvantages.

Firstly, in order to achieve security, performance is compromised. New algorithm should be devised to achieve better performance.

Secondly, because of the algorithms used, transaction size expanded, which brings limits to the number of transactions a block can hold, thus affect throughput of the system. In order to address the problem, signing signature with smaller sum of public key and signature should be developed.

# 6. CONCLUSION

In this paper, we explored the critical role of cryptography in enhancing blockchain technology, focusing on how it strengthens transaction security through encryption, digital signatures, and hashing. The immutable and decentralized nature of blockchain provides a robust foundation for secure and transparent transactions.

We analyzed various cryptographic techniques that protect blockchain systems from common threats such as double-spending and fraud, while also addressing the limitations of current methods and the impact of emerging technologies like quantum computing on blockchain security.

Our proposed model integrates advanced cryptographic protocols with blockchain technology to address existing vulnerabilities. By incorporating multi-signature schemes, zero-knowledge proofs, and post-quantum cryptography, the model aims to improve transaction security, scalability, and efficiency.
The implementation of our model highlights its effectiveness in enhancing blockchain security and emphasizes the need for continued research in cryptography and blockchain. This synergy between cryptography and blockchain is crucial for developing secure, efficient, and scalable systems, paving the way for future innovations in secure digital transactions.

| BlockChain | Consensus Mechanism | Hash Function | Signature | Security | Smart contract support | Expandability |
|---|---|---|---|---|---|---|
| Bitcoin | PoW | SHA256 | ECDSA | 4 | No | Limited |
| Ethereum | PoS | Keccak256 | ECDSA | 3.5 | Yes | Medium |
| Solana | PoH+PoS | SHA256 | ED25519 | 3 | Yes | High |
| QuantumGuard | PoMF | A parameter-based PQH | SPHINCS+-SHAKE256 | 5 | Yes | High |

**Figure 11: Blockchain & Cryptography**

## 7. ACKNOWLEGEMENTS

We would like to offer our most sincere appreciation and thanks to Professor Hugh Anderson for his guidance, enthusiasm and encouragement during the project. We would also like to thank our teaching assistant Shen Jiamin for his insight and expertise. It is with their careful guidance and attention that we were able to complete this project.


## 8. REFERENCES


[1] Sartaj Ahmad, Shubham Kumar Arya, Shobhit Gupta, Puneeta Singh, and Sanjeev Kumar Dwivedi. 2023. Study of Cryptographic Techniques Adopted in Blockchain. In *2023 4th International Conference on Intelligent Engineering and Management (ICIEM)*. 1–6. https://doi.org/10.1109/ICIEM59379.2023.10166591

[2] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. 2022. SPHINCS+ Submission to the NIST postquantum project, v.3.1. https://sphincs.org/data/sphincs+-r3.1-specification.pdf

[3] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2011. The Keccak Reference. *NIST SHA-3 Submission* (2011). https://keccak.team/files/Keccak-reference-3.0.pdf

[4] CFTE. 2024. What is Cryptography in Blockchain? How Does it Work? *CFTE Blog* (2024). https://blog.cfte.education/what-is-cryptography-in-blockchain/

[5] Coincentral. 2024. How Cryptographic Algorithms and Hashing Secure Blockchains. *Coincentral* (2024). https://coincentral.com/cryptographicalgorithms-hashing-secure-blockchains/

[6] Jintai Ding and Dieter Schmidt. 2005. Efficient Multivariate Public Key Schemes Based on Rainbow. *International Workshop on Post-Quantum Cryptography* (2005).

[7] John Doe and Jane Doe. 2024. An Example Article. *Journal of Examples* 1, 1 (2024), 1–10.

[8] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. BLISS: Bimodal Lattice Signature Scheme. *Journal of Cryptology* 26, 4 (2013), 824–841.

[9] Sanjeev Dwivedi et al. 2024. Study of Cryptographic Techniques Adopted in Blockchain. *ResearchGate* (2024). https://www.researchgate.net/profile/SanjeevDwivedi-6/publication/372120695_Study_of_Cryptographic_Techniques_Adopted_in_Blockchain/links/65436e440426ef6369f6574b/Study-of-Cryptographic-Techniques-Adopted-in-Blockchain.pdf

[10] T. M. Fernández-Caramès and P. Fraga-Lamas. 2020. Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks. *IEEE Access* 8 (2020), 21091–21116. https://doi.org/10.1109/ACCESS.2020.2968985

[11] Standards for Efficient Cryptography Group (SECG). 2010. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Technical Report. Standards for Efficient Cryptography Group. https://www.secg.org/sec2-v2.pdf

[12] Ethereum Foundation. 2015. RLP (Recursive Length Prefix) Encoding.https://ethereum.org/en/developers/docs/data-structures-and-encoding/rlp/

[13] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 212–219.

[14] Jeffery Hoffstein, Jill Pipher, and Joseph H Silverman. 1998. *NTRU Cryptosystems*. Springer.

[15] IEEE. 2024. Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks. *IEEE Xplore* (2024). https://ieeexplore.ieee.org/document/8967098

[16] Investopedia. 2024. What Are Consensus Mechanisms in Blockchain and Cryptocurrency? *Investopedia* (2024). https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp

[17] Sergey Krendelev and Polina Sazonova. 2018. Parametric Hash Function Resistant to Attack by Quantum Computer. In *Proceedings of the Federated Conference on Computer Science and Information Systems*. IEEE, 825–828.

[18] Wenting Li, Sebastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Year. Securing Proof-of-Stake Blockchain Protocols. *NEC Laboratories Europe, Germany* (Year).

[19] Robert J. McEliece. 1978. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *DSN Progress Report* (1978).

[20] Michele Mosca. 2018. Cybersecurity in an Era with Quantum Computers: Will We Be Ready? *IEEE Security & Privacy* 16, 5 (2018), 38–41. https://doi.org/10.1109/MSP.2018.3761723

[21] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Business Review* 2008, 1 (2008).

[22] Netizen. 2024. Blockchain Security: The Power of Cryptographic Algorithms. *Netizen News* (2024). https://www.netizen.net/news/post/4397/blockchain-securitythe-power-of-cryptographic-algorithms

[23] Nextrope. 2024. Advanced Cryptographic Techniques for Secure Blockchain Development. *Nextrope* (2024). https://nextrope.com/advanced-cryptographictechniques-for-secure-blockchain-development

[24] Cong T. Nguyen, Dinh Thai Hoang, Diep N. Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. 2019. Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access* 7 (2019), 85727–85745.

[25] National Institute of Standards and Technology (NIST). 2015. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication 180-4. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/ NIST.FIPS.180-4.pdf

[26] National Institute of Standards and Technology (NIST). 2015. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standards Publication 202. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[27] Sarah Rothrie. [n. d.]. *HOW CRYPTOGRAPHIC ALGORITHMS AND HASHING SECURE BLOCKCHAINS*. https://coincentral.com/cryptographic-algorithmshashing-secure-blockchains/

[28] ryanboulrice. [n. d.]. *Blockchain Security: The Power of Cryptographic Algorithms*. https://blog.netizen.net/2024/06/10/blockchain-security-the-power-ofcryptographic-algorithms/

[29] Gavin Wood. 2014. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Technical Report. Ethereum. https://ethereum.github.io/yellowpaper/paper.pdf

[30] Shuang Yao and Dawei Zhang. 2022. An Anonymous Verifiable Random Function with Applications in Blockchain. *Wireless Communications and Mobile Computing* 2022 (2022).


# APPENDIX

## A. WOTS+

### A.1 WOTS+ Parameters

- n: the parameter determining the level of security
- w: the Winternitz parameter, which can be set to 4,16 or 25
- len: the quantity of n-byte-string elements in a WOTS+ private key, public key, and signature, calculated as len = len1 + len2

$$\mathtt{len}_1 = \left\lceil \frac{8n}{\log(w)} \right\rceil, \ \mathtt{len}_2 = \left\lfloor \frac{\log\left(\mathtt{len}_1(w-1)\right)}{\log(w)} \right\rfloor + 1$$

**Figure 12: Equation**

### A.2 WOTS+ Chaining Function

Calculates an iteration of F on an n-byte input with a WOTS+ hash address ADRS and a public seed PK.seed. The ADRS address must have the first seven 32-bit words set to represent the address of the chain. For each iteration, the address is updated to reflect the current position in the chain before using ADRS to process the input with F.

- Input: an n-byte string X, a starting index i, a number of steps s, ADRS, and PK.seed
- Output: applying the function F to the input X for s iterations

## A.3 WOTS+ Private Key (sk)

- Provided: skADRS, SK.seed
- ‘sk‘ is an array of length ‘len‘, containing n-byte strings. This private key must be used to sign only one message and is used implicitly.
- Each n-byte string in the WOTS+ private key is generated from a secret seed ‘SK.seed‘, which is included in the SPHINCS+ secret key, and a WOTS+ key generation address ‘skADRS‘, using a pseudorandom function (PRF).

## A.4 WOTS+ Public Key Generation (pk)

- Provided: ADRS, PK.seed, SK.seed
- A WOTS+ key pair establishes a virtual structure comprising ‘len‘ hash chains, each with a length of ‘w‘. Each of the ‘len‘ n-byte strings:
  - The private key specifies the initial node for each hash chain.
  - The public key is obtained by applying a tweakable hash function to the end nodes of these hash chains.
- The address ADRS must encode the location of the WOTS+ key pair within the SPHINCS+ framework.

## A.5 WOTS+ Signature Generation

- Provided: ADRS, PK.seed, SK.seed
- It is created by converting a message M into ‘len‘ integers ranging from 0 and w - 1.
- len1: The message is converted into len1 base-w numbers using the base_w function.
- len2: A checksum of the message M is calculated and added to the transformed message len2 base-w numbers using the base_w function.
- Each base-w integer is used to choose a node from a different hash chain, and the signature is created by concatenating these selected nodes.

## A.6 WOTS+ Compute Public Key from Signature

- Provided: ADRS, public seed PK.seed
- SPHINCS+ uses implicit signature verification for WOTS+.To verify the signature, the verifier generates a WOTS+ public key from the signature and then uses this value to compute the SPHINCS+ public key for verification.

# B. HYPERTREE

## B.1 XMSS (eXtended Merkle Signature Scheme)

WOTS+ is integrated with a binary hash tree, resulting in a version of XMSS with a fixed input length. It is then used to perform HT signatures.

- Node: an n-byte value obtained by applying a tweakable hash function to the concatenation of its two child nodes
- Root node: XMSS public key
- Leaves: the public keys generated by WOTS+

The XMSS secret key is a single seed used to generate all WOTS+ secret keys. An XMSS signature includes both a WOTS+ signature and an authentication path. The authentication path is a set of tree nodes that enables the verifier to compute the root value of the tree from the WOTS+ signature. This path helps the verifier confirm the validity of the root value.

The verification process involves the verifier calculating the root value using the authentication path and the WOTS+ signature, and then confirming its correctness. The root value, which is the top node of the tree, is derived from this calculation.

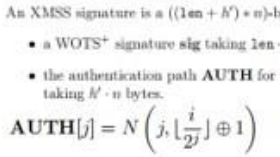



**Figure 13: XMSS Signature**

## B.2 Hypertree (HT)

A HT (Hierarchical Tree) consists of multiple layers of XMSS trees. The higher and intermediate layers are responsible for signing the public keys, or root nodes, of the XMSS trees in the layer directly below. The lowest layer trees are used to sign the actual messages, which are FORS public keys in SPHINCS+. All XMSS trees within a HT have the same height.

- HT total height $h$ that has $d$ layers of XMSS trees of height $h' = h/d$.
- Layer $d-1$ contains one XMSS tree, layer $d-2$ contains $2^{h'}$ XMSS trees, and so on.
- Finally, layer 0 contains $2^{h-h'}$ XMSS trees.

A HT signature $S_{HT}$ is a sequence of bytes with a length of $(h + d \cdot len) \cdot n$. It is made up of $d$ XMSS signatures , each of which is $(h/d + len) \cdot n$ bytes.

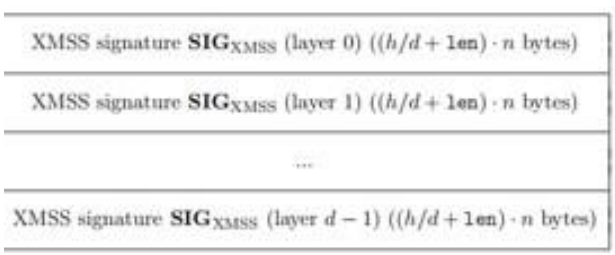



**Figure 14: HT Signature**

# C. SPHINCS+

## C.1 SPHINCS+ Key Generation

- Private Key: n-byte SK.seed, n-byte SK.prf
- Public Key: the HT public key, n-byte PK.seed

Since spx_sign does not receive the public key but requires access to PK.seed (and potentially PK.root for fault attack protection), the SPHINCS+ secret key includes a copy of the public key.

## C.2 SPHINCS+ Signature

$S_{HT}$ is a string of bytes with a length of $(1+k(a+1)+h+d \cdot len) \cdot n$.

It consists of:

- an n-byte string used for randomization, R
- a FORS signature $SIG_{FORS}$ consisting of $k(a+1)$ n-byte strings
- • a HT signature $S_{HT}$ of $(h + d \cdot len) \cdot n$ bytes


**Figure 15: Sphincs+ Signature**

## D. ALGORITHM SCHEME

Firstly, several parameters should be provided, including input length $m$, expected hash length $n$, an initial vector $iv$ of length $m$, a big prime $p$, and an input $I$ which directly determines output block size.

Secondly, a calculation matrix $M$ of $(m+n) \times n$ will be created, where

$$M_i = \begin{cases} (0,\dots,1\dots,0), i \le m \\ (0,\dots,0), i > m \end{cases}$$

$M_i$ represents the $i$-th row of $M$. After that, the first $m$ rows are an identity matrix.

Thirdly, we start the hash calculation, following the scheme:

$$M_I = \left( \sum_{j=0}^{n} iv_j . M_{i-j} \bmod p \right)$$

$$M_i = M_i \cdot H(I) \bmod p$$

where $H(I)$ is a polynomial hash function to calculate the corresponding hash convert.

In the original paper, the author used:

$$H(I) = a_{ij} \cdot I(i) \cdot I(i+1) \bmod p$$

In our implementation, we use

$$H(I) = a_{ij} \cdot (I(i)^2 + I(i+1)^2) \bmod p$$

which improves robustness and efficiency. That's because the original operation is to do a cyclic shift to avoid turning into 0, while our method reduces the chance of doing so.

Finally, we get the desired hash outcome

$$H(I) = M(m+n)$$

## E. BENCHMARK

We used "Query Performance Frequency" and "Query Performance Counter" in C to measure the precise time consumption. We created plain text files ranging from 16KB to 1MB to test the performance. We also conducted a performance comparison between PQH and SHA-256, a traditional hash function broadly implemented.

```
12th Gen Intel(R) Core(TM) i7-12700H 14c20t

48G DDR5 4800 RAM

Windows11 23H2

Language: C
```

**Figure 16: Benchmark Enviornment**

For every input parameter, we calculated three times and took the average value as data. If there were unusual deviations in the results, we performed a check and measured ten times. The results are as follows:

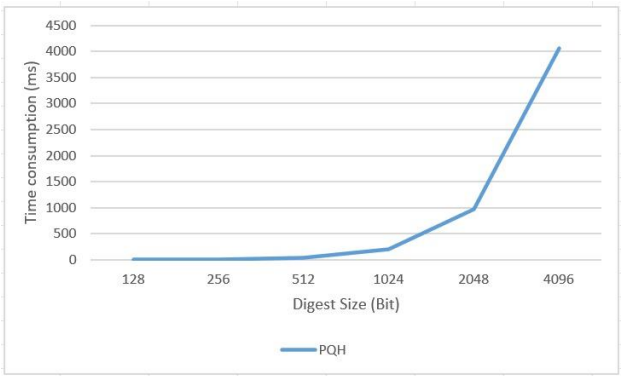



**Figure 17: Hash Function Performance**

## F. A WEIGHTED MULTI-FACTOR QUANTUM CONSENSUS MECHANISM (POMF)

In the following part, we will discuss the principles for each factor, then we will give a brief suggestion on the weighting function:

## F.1 Stake

The stake factor represents the funds participants possess on the blockchain. A typical idea is that the more funds participants hold and the longer they hold them, the more trustworthy they are. However, conventional stake-based proof mechanisms are susceptible to attacks like the "nothing at stake" and "long-range" attacks [18]. In our mechanism, the stake should be represented by the total unspent funds participants hold $u$ and the calculated time of holding the stake $t$, such that

$$S = F_S(u, t)$$

The function $F_S$ in our opinion should have the following features:

- As $u$ increases, the increase of $S$ should slow down, to make sure that $S$ is limited.
- Given specific $u$, as $t$ decreases, $S$ decreases, to ensure sudden attack attempts are mitigated.
- $t$'s influence on $S$ gets stronger as $u$ increases, to make sure that large funds are regulated properly.

## F.2 History (Reputation)

The history factor represents participants' longevity of involvement. If a participant has joined the blockchain for a long time, they are more trustworthy than new participants.

In our mechanism, we consider join time $jt$, misbehavior count $mc$, and block count $bc$, such that

$$H = F_H(jt, mc, bc)$$

In $F_H$, a misbehavior rate $mr$ will be calculated. As $mr$ goes down, $H$ will be significantly impacted. Additionally, block count as public behavior is more considered than transaction count. $jt$ serves as a multiplier here. In this function, we hope $H$ can have a log-like limitation, but still pose enough influence on $S$.

## F.3 Activeness

The activeness factor represents how active a participant is. In general, the more active a participant is, the more ideal they are to serve as a blocker.

In our mechanism, transaction count $tc$, join time $jt$, live fund rate $lfr$, and fund increase rate $fir$ are taken into consideration, such that

$$A = F_A(jt, tc, lfr, fir)$$

Transaction count along with join time can provide an analysis of a participant's activeness, while live fund rate indicates the ratio the participant uses its funds to deal. A judicious algorithm for activeness is required; in our view, activeness is deemed more vital than stake.

## F.4 Randomness

The randomness factor brings diversity to the whole system, greatly decreasing the chances of intentional attacks. The randomness should be verifiable, suggesting the implementation of related verifiable random functions like AVRF. [30]

## F.5 Weighting Function

In the weighting function, randomness should be given a reasonable proportion to ensure security. Activeness and reputation are prioritized over stake, to build a more decentralized blockchain. Moreover, the weighting function should incorporate an appropriate dynamic adjustment algorithm, allowing it to produce different outputs under varying conditions. From our perspective, at the beginning of the blockchain, stake can have a stronger impact because activeness and history shift rapidly. When the blockchain stabilizes, however, the weight should be adjusted accordingly. Other signals and trends should also be considered when designing such an algorithm.

# Taint Analysis for PHP: Tools and Their Evaluations

Fang Leyan*
Sichuan University
Chengdu, Sichuan, China
fangleyan0224@163.com

Ju Xiya
Wuhan University
Wuhan, Hubei, China
2192857893@qq.com

Meng Danling
Sichuan University
Chengdu, Sichuan, China
meng_danling@163.com

Wang Jingyan
Sichuan University
Chengdu, Sichuan, China
2824719763@qq.com

Zhou Shuhan
Huazhong University of Science and Technology
Wuhan, Hubei, China
szhou6114@gmail.com

## Abstract

In the realm of software security, taint analysis plays a vital role in identifying and mitigating vulnerabilities that can lead to severe security breaches. This paper presents a comprehensive investigation into this subject. By leveraging Abstract Syntax Trees and Control Flow Graphs , our works focus on static as well as dynamic taint analysis. We enhanced an open-source taint map project to track taint flows to sinks and modified an XSS detection extension to identify webshells by analyzing tainted parameters of sensitive functions. We also test with existing tools to have a more integral vision of various tools' analysis ability, offering insights for developers and security professionals alike.

***Keywords:*** Taint Analysis, PHP, Vulnerabilities Detection

## 1 Introduction

The advent of web applications has markedly changed our daily life, and PHP is one of the most popular and widely used scripting language geared towards web development. Almost 77% of all websites today had their server-side applications built using it[1]. PHP has a simple syntax that is easy to learn, and its scripts execute faster than many other scripting languages, which is beneficial for web applications. Nevertheless, PHP is often considered less secure compared to some other languages. PHP applications are commonly deployed on web servers and they directly react towards user interactions and requests, e.g., user clicks. Such an interaction mode opens a huge surface for various attacks where attackers can exploit the vulnerabilities in the applications[1].

Taint analysis has a wide variety of compelling applications in security tasks, from software attack detection to data lifetime analysis[2]. Static taint analysis (STA) is a technique to track the flow of potentially harmful data through a program without executing it. STA examines the code without running it, which allows for broad coverage of all possible execution. It helps in identifying vulnerabilities early in the development process, reducing the risk of security issues in deployed applications. There are existing tools for STA, such as RIPS, PHPSafe, which are used to automate the detection process.

Dynamic taint analysis (DTA), on the other hand, monitors the program in real-time to observe how data from untrusted sources propagates through the system. DTA provides insights into the actual execution of the program, considering the detection of vulnerabilities that may not be apparent through static analysis. Monitoring a program in real-time can introduce significant performance overhead, which may impact the system's efficiency.

## 2 BACKGROUND INFORMATION

### 2.1 Abstract Syntax Tree

The abstract syntax tree (AST), a fundamental code feature, is a tree-like data structure used to represent the abstract syntax structure of source code, illustrating the syntactic information of the source code and has been widely used in code representation learning. It is commonly acknowledged that AST-based code representation is critical to solving code-related tasks[3]. It uniquely represents a source code snippet in a given language and grammar, showing the syntactic structure of the code in the form of a tree, where each node represents a structure in the source code, such as statements, expressions, and functions, and the relationship between the nodes reflects the mutual relationship of these structures. AST is designed to be abstract, not to represent every syntactic detail, such as nested brackets being implicit in the tree structure, rather than existing as independent nodes. AST aims to provide a simplified and tractable representation of code to be used in various stages of a compiler or interpreter.

AST does not depend on the concrete syntax of the source language, and can be independent of the context-free grammar used in the syntax analysis stage, so as to avoid introducing redundant details in the syntax analysis process that affect the processing of subsequent stages. It has the characteristics of abstraction, tree structure, syntax representation and information extensibility. With AST, the syntactic structure of the source code is abstracted and simplified,

*Author names are listed in alphabetical order. All authors contributed equally to this paper.

[1]http://w3techs.com/technologies/overview/programming language/all, as of December 2013

and each node represents a code element such as a variable, function, and expression. Being unconstrained by concrete syntax rules, AST can be adapted to different programming languages.

Moreover, once the AST is created, in the subsequent semantic analysis phase, more information can be added to it for tasks such as semantic analysis, code optimization, and generation. AST provides a clear interface for the front-end and back-end of the compiler. By constructing the parse tree independently, unnecessary details can be avoided in the parse phase, making each phase of the compiler clearer and more efficient. In general, AST plays an important role in simplifying code structure, improving processing efficiency and enhancing code analysis ability in compiler design. AST plays an important role in vulnerability analysis in the following aspects:

- `Code parsing and analysis`: AST can be used to parse and analyze source code. By building an AST, the source code can be transformed into a form that is easier to process and analyze. For example, AST can be used by compilers and interpreters to understand the structure of source code and perform operations such as syntax checking, type checking, optimization, and code generation.
- `Code refactoring and rewriting`: AST can be used for code refactoring and rewriting. By analyzing the AST, the source code can be structurally modified, such as adding, deleting or replacing code blocks, to achieve code refactoring and rewriting. This is very useful in code maintenance and refactoring.
- `Static code analysis`: AST can be used for static code analysis. By analyzing AST, it is possible to detect potential problems and errors in the code, such as unused variables, dead code, code duplication, etc. Static code analysis can help developers to improve code quality and performance.
- `Code generation`: AST can be used for code generation. By analyzing the AST, object code can be generated according to the structure of the source code. This is important in compilers and interpreters to transform source code into executable machine code or bytecode.

### 2.2 Control Flow Graph

Control Flow Graph (CFG) is a graph-based data structure, which is used to represent the control flow relationships among the basic blocks in a program. Nodes in a CFG represent basic blocks in a program, and a basic block is a sequence of code without branch jumps. Edges represent possible control flow transitions between basic blocks, such as conditional jumps, loops, and function calls, among others. In the vulnerability analysis of PHP code, CFG plays the following important roles:

- `Code path analysis`: With CFG, all possible execution paths of a program can be analyzed to identify potential security vulnerabilities. For example, it is possible to detect if unfiltered user input enters a database query on certain paths that can trigger SQL injection vulnerabilities.
- `Loop and conditional branch detection`: CFG can help identify loops and conditional branches in programs and analyze their correctness and safety. For example, detecting whether the loop condition is correct and avoiding infinite loops or incorrect condition judgments.
- `Combining static and dynamic analysis`: CFGS can be used in conjunction with asts to provide more comprehensive code analysis. For example, syntactic structure analysis through AST and control flow analysis through CFG can be combined to detect and fix security vulnerabilities in code more effectively.

## 3 Implementation

PHP is vulnerable to various security issues. Taint analysis tracks data from untrusted sources, propagates it through execution paths, and checks its status at critical points (taint sinks). This method effectively addresses security concerns such as software attacks, information flow control, data leaks, malware analysis, among others[2]. Our investigations into this technique are as follows.

### 3.1 Static Analysis

#### 3.1.1 Taint Sources, Sinks, Propagation and Sanitizers.

Taint **sources** are entry points where tainted data infiltrates the application, including user inputs, data read from files or any information fetched from external services or databases.

Taint **sinks** are locations in the software where tainted data, if not properly handled, leads to vulnerabilities, including database queries, system calls, or outputting data to a web page.

Tainted data moves through the application and contaminate more data. Specific **propagation** rules are defined to determine how taint spreads. If a tainted variable is assigned to another variable, the new variable also becomes tainted. If tainted data is passed as an argument to a function, the corresponding parameter in the function becomes tainted. The analysis continues until the tainted data reaches a taint sink, which is a point where the data could cause harm if not properly sanitized.

During the propagation of tainted data, it may pass through **sanitizing** modules, which ensure that, after processing, the data no longer carries sensitive information or poses any threat to the system. To realize this, they perform operations such as data encryption, input validation, and removal of harmful elements, thereby preventing security vulnerabilities. The effectiveness of taint analysis largely depends on

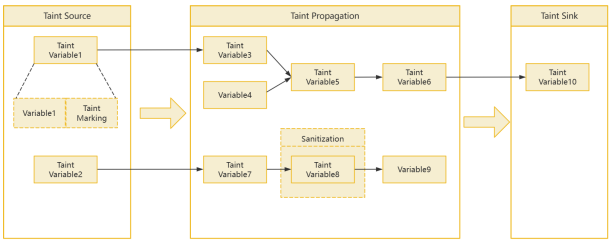



**Figure 1.** An Intuitive Example of Taint Flow

accurately identifying and implementing these sanitizers to reduce false positives and improve the overall security of the system. Overall work flow is shown in Figure 1.

**3.1.2 Generating AST.** We use PHP Parser to generate and visualize an AST for a given piece of PHP code, as shown in Figure 2. Currently, there are many AST parsing methods available for users. It is noteworthy that these parsers would generate distinct ASTs for the same source code due to different lexical rules and grammar rules[3]. Each node is represented as a node in the graph, and edges are created between parent and child nodes to reflect the hierarchical structures of the code, ultimately generating a PNG file that displays the tree structure.

After Generating AST, *nodeAnalysis* function is responsible for identifying and tracking the flow of tainted data through the program. It analyzes each node to determine whether it represents a source (where tainted data originates), a sink (where tainted data could lead to a vulnerability), or an intermediary point where data might be transformed or propagated. By systematically evaluating each node, the function helps in constructing a comprehensive map of how tainted data moves through the code, enabling the detection of potential security vulnerabilities and ensuring that appropriate sanitization measures are in place.

**3.1.3 From AST to CFG.** The *forwardFlowSensitiveAnalysis* function tracks the propagation of tainted data through the program in a forward direction and analyzes the flow of data from sources (where tainted data originates) to sinks (where tainted data could potentially cause vulnerabilities) by examining each statement and control structure in the code. It ensures that the analysis is sensitive to the order of execution, meaning it takes into account the sequence in which statements are executed. This helps in accurately identifying how tainted data moves through different branches, loops, and function calls, thereby providing a detailed understanding of potential security risks. By using this forward flow-sensitive approach, the analysis can detect vulnerabilities that might be missed by simpler, non-flow-sensitive methods, ultimately enhancing the security and robustness of the software.

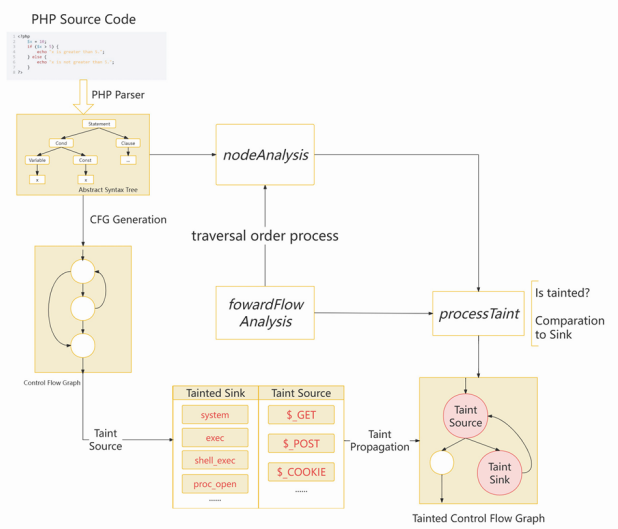



**Figure 2.** General Process of Taint Analysis

**3.1.4 And after that...** The *ForwardFlowSensitiveAnalysis* function's output is utilized by the *processTaint* function to track and manage tainted data flow within a program. This function identifies tainted sources, such as user inputs, and marks them. It then monitors the propagation of this tainted data through various operations and function calls, ensuring that any derived data is also marked as tainted. This comprehensive tracking helps identify points where tainted data might reach sensitive sinks, such as database queries or file operations, potentially leading to security risks like SQL injection (SQLi) or cross-site scripting (XSS). By maintaining a detailed map of potential vulnerabilities, *processTaint* ensures appropriate sanitization and validation measures are applied before tainted data reaches critical parts of the application. When *processTaint* marks data as tainted, the analysis tracks its propagation through the code. The next step involves checking if this tainted data reaches any sensitive sinks. If the data remains tainted upon reaching a sink, it indicates a potential vulnerability. Taint analysis compares the tainted data against a list of known sinks, flagging it as a security risk if it is not properly sanitized or validated[1].

We enhanced an open-source taint map project by incorporating an analysis feature that tracks whether taint flows reach taint sinks. This improvement enables the detection of vulnerabilities in PHP files, surpassing the original project's capability of merely identifying tainted variables.

## 3.2 Dynamic analysis

Dynamic analysis is a technique used to identify potential security vulnerabilities and performance issues by monitoring and analyzing the behavior of a program at runtime. Unlike static analysis, dynamic analysis does not solely rely on the static structure of the source code but collects data during

the actual execution of the program. This approach can detect issues that only manifest during runtime, making it a crucial tool for ensuring software security and reliability.

**3.2.1 Realization idea.** Webshells are a malicious scripts that can remotely control a webserver to execute arbitrary commands, steal sensitive files, and further invade the internal network. Existing webshell detection methods, such as using pattern matching for webshell detection, can be easily bypassed by attackers using the file include and user-defined functions[4]. In our work, we have modified an XSS detection extension to detect webshell. Our goal is to track and monitor the flow of data during the execution of a program to identify potential security vulnerabilities, particularly those involving user input data taint propagation and dangerous function calls.

**3.2.2 Hook mechanism.** In dynamic analysis, the Hook mechanism is a core technology for monitoring program behavior. Hook Mechanism as shown in Figure 3. The entire Hook mechanism consists of three key components:

- `Hook Plugin Manager Class`: This is the core file, a global object for the application. It has three main responsibilities:
  - `Monitor all registered plugins and instantiate their objects`: Through a monitoring mechanism, ensure that all registered plugins are correctly invoked when specific conditions are met.
  - `Register all plugins`: Manage and maintain the registration information of the plugins, ensuring that all necessary plugins are correctly identified and invoked.
  - `Trigger the corresponding object methods`: When the hook conditions are met, invoke the corresponding plugin methods to intervene and monitor the program execution process.
- `Plugin Implementation`: The specific implementation of the plugins is usually completed by third-party developers, but it must adhere to the rules defined by the manager class. These rules are stipulated by the plugin mechanism and may vary depending on different plugin mechanisms. Plugin developers need to ensure that their implementations can correctly integrate with the Hook mechanism and meet the specified functional requirements.
- `Plugin Triggering`: The trigger conditions for the plugins are crucial for the implementation of the Hook mechanism. This is typically a small piece of code placed where the plugin needs to be invoked, used to trigger the hook. When certain conditions are met, this code will call the registered plugin methods to perform the necessary operations.

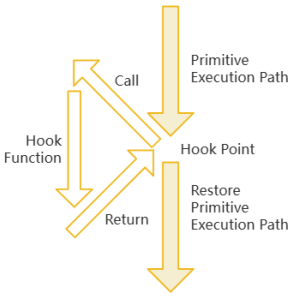



**Figure 3.** Hook Mechanism

**3.2.3 Taint Propagation.** Taint propagation was briefly mentioned in 3.1.1 of the article, which has the following two principles:

- `Taint can propagate`: "Taint" refers to data that comes from an untrusted source, such as user input. Taint propagation is that if a piece of data is tainted, any variable that gets its value from this tainted data also becomes tainted.
- `A taint function is only considered tainted if its argument is tainted at execution`: When the taint function is executed, it will invoke a hook function designed to verify whether the function has been executed and whether the parameter is tainted. For example, in the following code, Initially, $a gets its value from user input ($_GET['1']), making it tainted. The eval function is called with $a as its parameter. Since $a is tainted, this use of eval is dangerous because it will execute potentially harmful code. Later, $a is reassigned a new value 'ls', which is a safe, non-tainted string. The system function is then called with $a as its parameter. Since $a is not tainted, this use of system is considered safe and not dangerous.

```
<?php
    $a = $_GET['1'];
    eval($a);
    $a = 'ls';
    system($a)
?>
```

**3.2.4 Test.** When there's a need to dynamically execute code and pass parameters, we opt to package the program into a Docker container. Using Go, we've implemented functionality to parse GET and POST data formats from the source code, then launched an HTTP service. This service handles incoming requests in accordance with the source code's specifications. By adopting this approach, we deploy our application within a Docker container and interact with

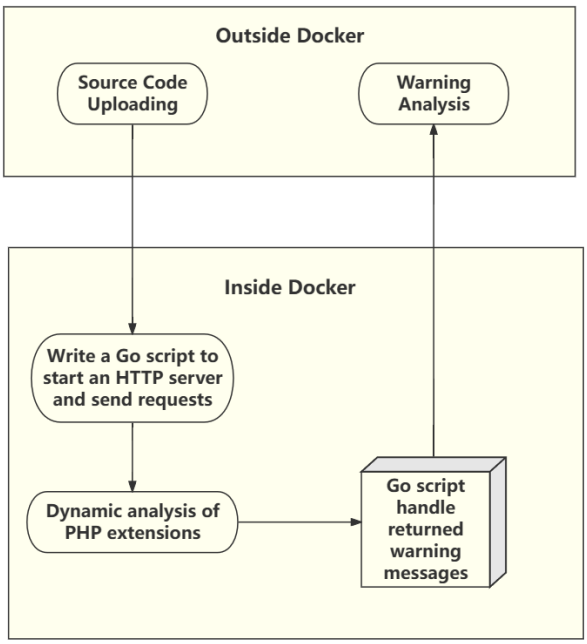



**Figure 4.** Docker Structure

| sample | Our Static Tool | Our Dynamic Tool | RIPS | PHPStan Security |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 5 | 0 | 1 | 1 | 0 |

**Figure 5.** Comparison Result

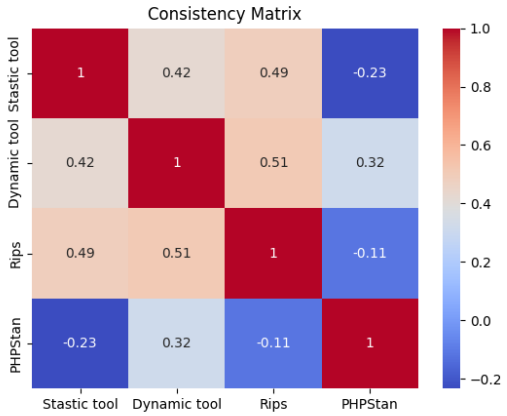



**Figure 6.** Performances Consistency

external systems via HTTP services, enabling dynamic execution and parameter passing. Overall Architecture of Docker as shown in Figure 4. This design not only ensures the reliability and security of our application but also enhances development and deployment efficiency.

# 4 Analysis

## 4.1 Existing tools: RIPS and PHPStan Extension

RIPS, developed by Johannes Dahse, is an open-source static code analysis tool for PHP, with the latest version being 0.55. It employs static analysis to detect security vulnerabilities such as XSS, SQL injection, file disclosure, and header injection. The tool is compact, under 500KB, and offers precise PHP syntax analysis, enabling cross-file variable and function tracking with a low false positive rate. This efficiency allows penetration testers to review analysis results without inspecting the entire codebase[5][6].

PHPStan, developed on top of Nikita Popov's PHP Parser, is a static analysis tool written in PHP. It is designed to detect various errors in PHP code, including syntax, type, and logical errors. PHPStan Security, an extension of PHPStan, focuses on identifying security vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). This extension aids developers in pinpointing potential performance and readability issues in their code[7].

We will run them separately, along with our implemented static and dynamic tools, to observe and compare the results.

## 4.2 Evaluations

By using the same dataset, we make a comparison between the static tool, dynamic tool, RIPS and PHPStan Security, in order to have a more integral vision of various tools' analysis ability. Figure 5 shows the comparison result of the first 5 test points. More details can be witnessed in Appendix A.

We calculated the Pearson correlation coefficients between four tools pairwise, as shown in Figure 6.

Our analysis shows that our static vulnerability detection tool and dynamic vulnerability detection tool have about a 50% positive correlation with RIPS. This demonstrates that our tools are somewhat consistent with RIPS, indicating a certain level of analysis capability. Our two self-made detection tools also show a 42% correlation with each other. This indicates that there is some consensus between the two tools in detecting vulnerabilities, although the consistency is not particularly high. This could be due to the different analysis methods they employ. Meanwhile, PHPStan Security's results show a slight negative correlation with both RIPS and our static analysis tool, indicating that PHPStan Security's accuracy is relatively low, or it detects different types of vulnerabilities compared to RIPS.

To statistically analyze false positives (where correct code is mistakenly reported as erroneous), we assume that the probability of multiple tools generating false positives for the same code is very low. Therefore, we use the number of

unique detection(where only one tool regards it as erroneous) as a proxy for false positives. We have tallied the occurrences of unique detection for each tool, as shown in Table 1.

**Table 1.** Unique Detection Count

| Our Static Tool | Our Dynamic Tool | RIPS | PHPStan |
|---|---|---|---|
| 0 | 41 | 0 | 65 |

It can be seen that both the Static Tool and RIPS had no unique detection, while the Dynamic Tool and PHPStan had 41 and 65 unique detection samples, respectively. This suggests that these two tools might have a higher probability of generating false positives. However, there are also special cases where code samples with vulnerabilities can bypass three of the four tools, resulting in only one tool detecting them. Additionally, there are instances where two tools may generate false positives for the same code sample.

### 4.3 Combination

Combining static and dynamic analysis could leverage the strengths of both methods. Static analysis provides early detection of potential vulnerabilities by examining code without execution, while dynamic analysis identifies real-time vulnerabilities during execution, reducing false positives. Integrating these approaches enhances overall detection accuracy, balances the trade-offs between false positives and missed vulnerabilities, and optimizes resource allocation for security assessments[8].

## 5 Conclusion and Future Work

In this paper, we enhanced an open-source taint map project by adding a feature to track taint flows to sinks, allowing for detecting vulnerabilities in PHP files, beyond just identifying tainted variables. We also modified an XSS detection extension to detect webshells by analyzing whether parameters of sensitive functions (e.g., system, eval) are tainted, determining if these functions are tainted.

Further improvements to our static analysis work may include strengthening the detection and analysis of specific bypass techniques, and conducting some more detailed examinations of certain nodes. Additionally, due to time deficiency, we were unable to implement filter recognition and forward flow analysis, which may result in some misjudgments of vulnerabilities.

## Acknowledgments

We would like to thank our beloved Professor Hugh Anderson and our teaching assistant Shen Jiamin, for their invaluable help, guidance and patience throughout the project. We are also very grateful to NUS School of Computing for the organization and preparation of the workshop, which has been an amazing journey to us all.

## References

[1] C. Luo, P. Li and W. Meng. 2022. “TChecker: Precise Static Inter-Procedural Analysis for Detecting Taint-Style Vulnerabilities in PHP Applications”. *Proceedings of the 2022 ACMSIGSACConferenceon-Computer and Communications Security (CCS ’22), November 7–11, 2022, Los Angeles, CA, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/ 3548606.3559391

[2] Jiang Ming, Dinghao Wu, Gaoyao Xiao, Jun Wang, and Peng Liu, 2007. *TaintPipe: Pipelined Symbolic Taint Analysis*, 24th USENIX Security Symposium, USENIX Association, Washington, D.C. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ming

[3] Weisong Sun, Chunrong Fang, Yun Miao, Yudu You, Mengzhe Yuan, Yuchen Chen, Quanjun Zhang, An Guo, Xiang Chen, Yang Liu, Zhenyu Chen, "Abstract Syntax Tree for Programming Language Understanding and Representation: How Far Are We?", arXiv:2312.00413v1 [cs.SE] 1 Dec 2023

[4] Jiazhen Zhao, Yuliang Lu, Xin Wang, Kailong Zhu and Lu Yu, "WTA:AStatic Taint Analysis Framework for PHP Webshell" Appl. Sci. 2021, 11(16), 7763; https://doi.org/10.3390/app11167763

[5] Johannes Dahse, "RIPS - A static source code analyser for vulnerabilities in PHP scripts", GitHub, 2010.

[6] https://github.com/robocoder/rips-scanner: "RIPS - free PHP security scanner using static code analysis", SourceForge, 2013.

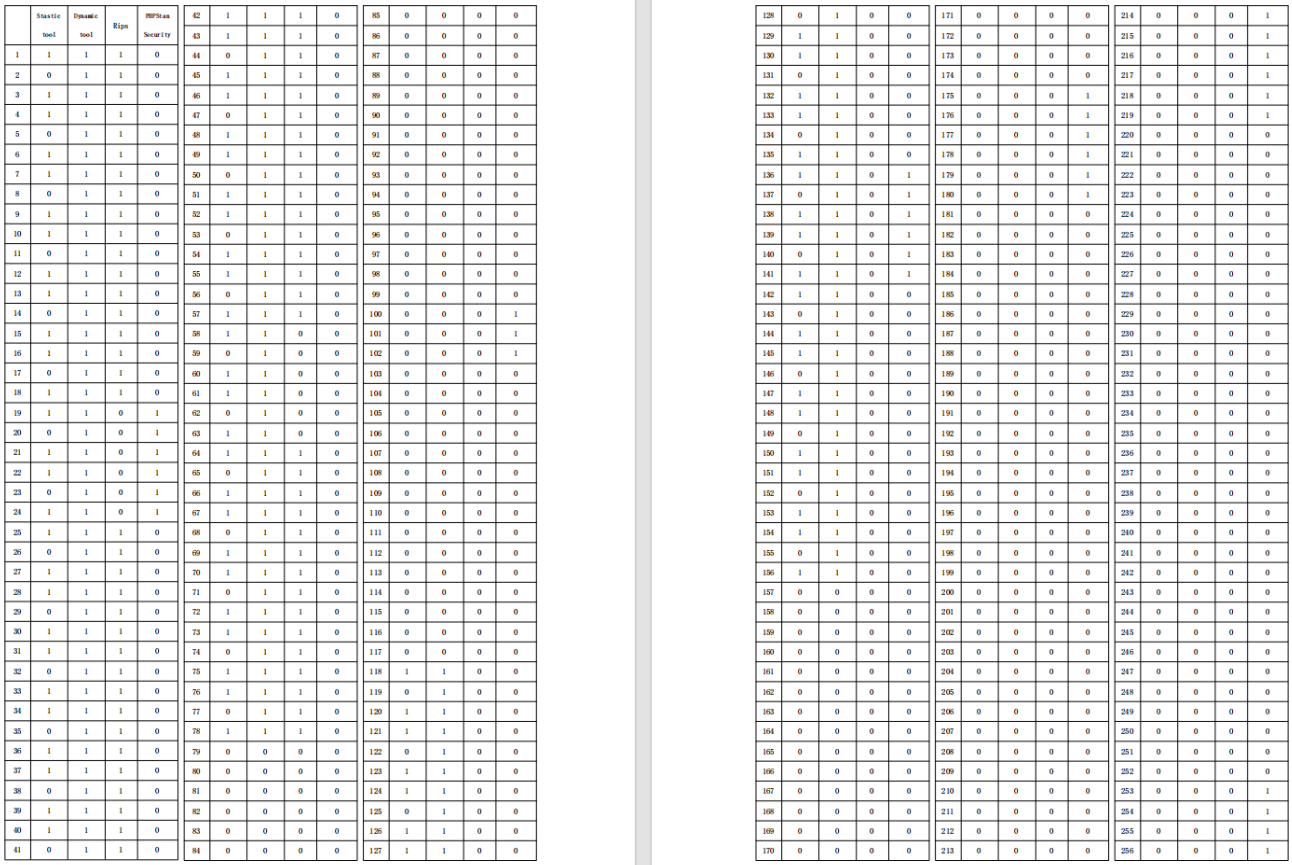
[7] https://phpstan.org/user-guide/extension-library: Nikita Popov, "PHP Parser", GitHub, 2012.

[8] A. Damodaran, F. D. Troia, V. A. Corrado, T. H. Austin and M. Stamp, "A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection", arXiv:2203.09938 [cs.CR]

# APPENDIX

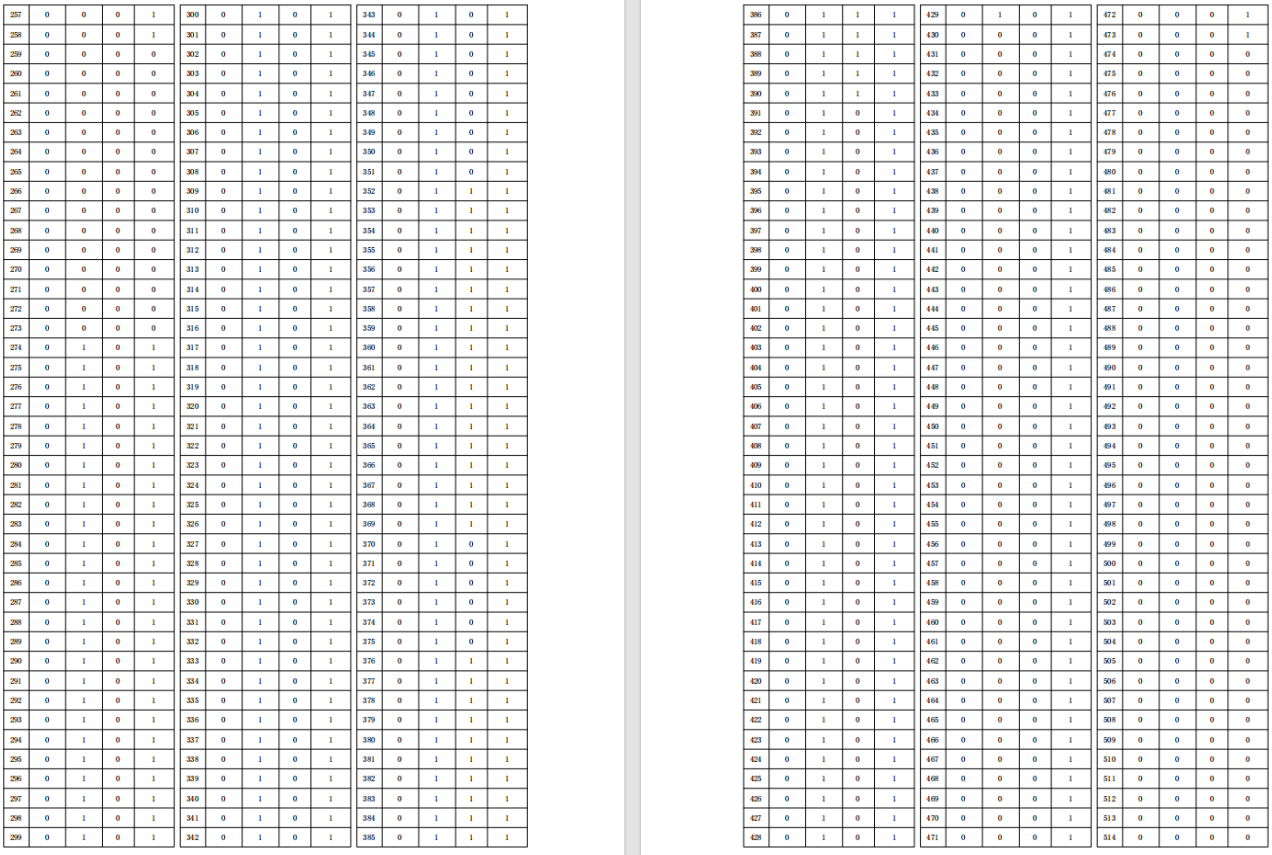
## A. COMPARISON RESULTS

Comparison Results on our dataset are shown in Figure 7, 8 and 9.



| | Stastic tool | Dynamic tool | Rips | PHPStan Security |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 |
| 4 | 1 | 1 | 1 | 0 |
| 5 | 0 | 1 | 1 | 0 |
| 6 | 1 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 |
| 8 | 0 | 1 | 1 | 0 |
| 9 | 1 | 1 | 1 | 0 |
| 10 | 1 | 1 | 1 | 0 |
| 11 | 0 | 1 | 1 | 0 |
| 12 | 1 | 1 | 1 | 0 |
| 13 | 1 | 1 | 1 | 0 |
| 14 | 0 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 0 |
| 16 | 1 | 1 | 1 | 0 |
| 17 | 0 | 1 | 1 | 0 |
| 18 | 1 | 1 | 1 | 0 |
| 19 | 1 | 1 | 0 | 1 |
| 20 | 0 | 1 | 0 | 1 |
| 21 | 1 | 1 | 0 | 1 |
| 22 | 1 | 1 | 0 | 1 |
| 23 | 0 | 1 | 0 | 1 |
| 24 | 1 | 1 | 0 | 1 |
| 25 | 1 | 1 | 1 | 0 |
| 26 | 0 | 1 | 1 | 0 |
| 27 | 1 | 1 | 1 | 0 |
| 28 | 1 | 1 | 1 | 0 |
| 29 | 0 | 1 | 1 | 0 |
| 30 | 1 | 1 | 1 | 0 |
| 31 | 1 | 1 | 1 | 0 |
| 32 | 0 | 1 | 1 | 0 |
| 33 | 1 | 1 | 1 | 0 |
| 34 | 1 | 1 | 1 | 0 |
| 35 | 0 | 1 | 1 | 0 |
| 36 | 1 | 1 | 1 | 0 |
| 37 | 1 | 1 | 1 | 0 |
| 38 | 0 | 1 | 1 | 0 |
| 39 | 1 | 1 | 1 | 0 |
| 40 | 1 | 1 | 1 | 0 |
| 41 | 0 | 1 | 1 | 0 |
| 42 | 1 | 1 | 1 | 0 |
| 43 | 1 | 1 | 1 | 0 |
| 44 | 0 | 1 | 1 | 0 |
| 45 | 1 | 1 | 1 | 0 |
| 46 | 1 | 1 | 1 | 0 |
| 47 | 0 | 1 | 1 | 0 |
| 48 | 1 | 1 | 1 | 0 |
| 49 | 1 | 1 | 1 | 0 |
| 50 | 0 | 1 | 1 | 0 |
| 51 | 1 | 1 | 1 | 0 |
| 52 | 1 | 1 | 1 | 0 |
| 53 | 0 | 1 | 1 | 0 |
| 54 | 1 | 1 | 1 | 0 |
| 55 | 1 | 1 | 1 | 0 |
| 56 | 0 | 1 | 1 | 0 |
| 57 | 1 | 1 | 1 | 0 |
| 58 | 1 | 1 | 0 | 0 |
| 59 | 0 | 1 | 0 | 0 |
| 60 | 1 | 1 | 0 | 0 |
| 61 | 1 | 1 | 0 | 0 |
| 62 | 0 | 1 | 0 | 0 |
| 63 | 1 | 1 | 0 | 0 |
| 64 | 1 | 1 | 1 | 0 |
| 65 | 0 | 1 | 1 | 0 |
| 66 | 1 | 1 | 1 | 0 |
| 67 | 1 | 1 | 1 | 0 |
| 68 | 0 | 1 | 1 | 0 |
| 69 | 1 | 1 | 1 | 0 |
| 70 | 1 | 1 | 1 | 0 |
| 71 | 0 | 1 | 1 | 0 |
| 72 | 1 | 1 | 1 | 0 |
| 73 | 1 | 1 | 1 | 0 |
| 74 | 0 | 1 | 1 | 0 |
| 75 | 1 | 1 | 1 | 0 |
| 76 | 1 | 1 | 1 | 0 |
| 77 | 0 | 1 | 1 | 0 |
| 78 | 1 | 1 | 1 | 0 |
| 79 | 0 | 0 | 0 | 0 |
| 80 | 0 | 0 | 0 | 0 |
| 81 | 0 | 0 | 0 | 0 |
| 82 | 0 | 0 | 0 | 0 |
| 83 | 0 | 0 | 0 | 0 |
| 84 | 0 | 0 | 0 | 0 |
| 85 | 0 | 0 | 0 | 0 |
| 86 | 0 | 0 | 0 | 0 |
| 87 | 0 | 0 | 0 | 0 |
| 88 | 0 | 0 | 0 | 0 |
| 89 | 0 | 0 | 0 | 0 |
| 90 | 0 | 0 | 0 | 0 |
| 91 | 0 | 0 | 0 | 0 |
| 92 | 0 | 0 | 0 | 0 |
| 93 | 0 | 0 | 0 | 0 |
| 94 | 0 | 0 | 0 | 0 |
| 95 | 0 | 0 | 0 | 0 |
| 96 | 0 | 0 | 0 | 0 |
| 97 | 0 | 0 | 0 | 0 |
| 98 | 0 | 0 | 0 | 0 |
| 99 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 1 |
| 101 | 0 | 0 | 0 | 1 |
| 102 | 0 | 0 | 0 | 1 |
| 103 | 0 | 0 | 0 | 0 |
| 104 | 0 | 0 | 0 | 0 |
| 105 | 0 | 0 | 0 | 0 |
| 106 | 0 | 0 | 0 | 0 |
| 107 | 0 | 0 | 0 | 0 |
| 108 | 0 | 0 | 0 | 0 |
| 109 | 0 | 0 | 0 | 0 |
| 110 | 0 | 0 | 0 | 0 |
| 111 | 0 | 0 | 0 | 0 |
| 112 | 0 | 0 | 0 | 0 |
| 113 | 0 | 0 | 0 | 0 |
| 114 | 0 | 0 | 0 | 0 |
| 115 | 0 | 0 | 0 | 0 |
| 116 | 0 | 0 | 0 | 0 |
| 117 | 0 | 0 | 0 | 0 |
| 118 | 1 | 1 | 0 | 0 |
| 119 | 0 | 1 | 0 | 0 |
| 120 | 1 | 1 | 0 | 0 |
| 121 | 1 | 1 | 0 | 0 |
| 122 | 0 | 1 | 0 | 0 |
| 123 | 1 | 1 | 0 | 0 |
| 124 | 1 | 1 | 0 | 0 |
| 125 | 0 | 1 | 0 | 0 |
| 126 | 1 | 1 | 0 | 0 |
| 127 | 1 | 1 | 0 | 0 |
| 128 | 0 | 1 | 0 | 0 |
| 129 | 1 | 1 | 0 | 0 |
| 130 | 1 | 1 | 0 | 0 |
| 131 | 0 | 1 | 0 | 0 |
| 132 | 1 | 1 | 0 | 0 |
| 133 | 1 | 1 | 0 | 0 |
| 134 | 0 | 1 | 0 | 0 |
| 135 | 1 | 1 | 0 | 0 |
| 136 | 1 | 1 | 0 | 1 |
| 137 | 0 | 1 | 0 | 1 |
| 138 | 1 | 1 | 0 | 1 |
| 139 | 1 | 1 | 0 | 1 |
| 140 | 0 | 1 | 0 | 1 |
| 141 | 1 | 1 | 0 | 1 |
| 142 | 1 | 1 | 0 | 0 |
| 143 | 0 | 1 | 0 | 0 |
| 144 | 1 | 1 | 0 | 0 |
| 145 | 1 | 1 | 0 | 0 |
| 146 | 0 | 1 | 0 | 0 |
| 147 | 1 | 1 | 0 | 0 |
| 148 | 1 | 1 | 0 | 0 |
| 149 | 0 | 1 | 0 | 0 |
| 150 | 1 | 1 | 0 | 0 |
| 151 | 1 | 1 | 0 | 0 |
| 152 | 0 | 1 | 0 | 0 |
| 153 | 1 | 1 | 0 | 0 |
| 154 | 1 | 1 | 0 | 0 |
| 155 | 0 | 1 | 0 | 0 |
| 156 | 1 | 1 | 0 | 0 |
| 157 | 0 | 0 | 0 | 0 |
| 158 | 0 | 0 | 0 | 0 |
| 159 | 0 | 0 | 0 | 0 |
| 160 | 0 | 0 | 0 | 0 |
| 161 | 0 | 0 | 0 | 0 |
| 162 | 0 | 0 | 0 | 0 |
| 163 | 0 | 0 | 0 | 0 |
| 164 | 0 | 0 | 0 | 0 |
| 165 | 0 | 0 | 0 | 0 |
| 166 | 0 | 0 | 0 | 0 |
| 167 | 0 | 0 | 0 | 0 |
| 168 | 0 | 0 | 0 | 0 |
| 169 | 0 | 0 | 0 | 0 |
| 170 | 0 | 0 | 0 | 0 |
| 171 | 0 | 0 | 0 | 0 |
| 172 | 0 | 0 | 0 | 0 |
| 173 | 0 | 0 | 0 | 0 |
| 174 | 0 | 0 | 0 | 0 |
| 175 | 0 | 0 | 0 | 1 |
| 176 | 0 | 0 | 0 | 1 |
| 177 | 0 | 0 | 0 | 1 |
| 178 | 0 | 0 | 0 | 1 |
| 179 | 0 | 0 | 0 | 1 |
| 180 | 0 | 0 | 0 | 1 |
| 181 | 0 | 0 | 0 | 0 |
| 182 | 0 | 0 | 0 | 0 |
| 183 | 0 | 0 | 0 | 0 |
| 184 | 0 | 0 | 0 | 0 |
| 185 | 0 | 0 | 0 | 0 |
| 186 | 0 | 0 | 0 | 0 |
| 187 | 0 | 0 | 0 | 0 |
| 188 | 0 | 0 | 0 | 0 |
| 189 | 0 | 0 | 0 | 0 |
| 190 | 0 | 0 | 0 | 0 |
| 191 | 0 | 0 | 0 | 0 |
| 192 | 0 | 0 | 0 | 0 |
| 193 | 0 | 0 | 0 | 0 |
| 194 | 0 | 0 | 0 | 0 |
| 195 | 0 | 0 | 0 | 0 |
| 196 | 0 | 0 | 0 | 0 |
| 197 | 0 | 0 | 0 | 0 |
| 198 | 0 | 0 | 0 | 0 |
| 199 | 0 | 0 | 0 | 0 |
| 200 | 0 | 0 | 0 | 0 |
| 201 | 0 | 0 | 0 | 0 |
| 202 | 0 | 0 | 0 | 0 |
| 203 | 0 | 0 | 0 | 0 |
| 204 | 0 | 0 | 0 | 0 |
| 205 | 0 | 0 | 0 | 0 |
| 206 | 0 | 0 | 0 | 0 |
| 207 | 0 | 0 | 0 | 0 |
| 208 | 0 | 0 | 0 | 0 |
| 209 | 0 | 0 | 0 | 0 |
| 210 | 0 | 0 | 0 | 0 |
| 211 | 0 | 0 | 0 | 0 |
| 212 | 0 | 0 | 0 | 0 |
| 213 | 0 | 0 | 0 | 0 |
| 214 | 0 | 0 | 0 | 1 |
| 215 | 0 | 0 | 0 | 1 |
| 216 | 0 | 0 | 0 | 1 |
| 217 | 0 | 0 | 0 | 1 |
| 218 | 0 | 0 | 0 | 1 |
| 219 | 0 | 0 | 0 | 1 |
| 220 | 0 | 0 | 0 | 0 |
| 221 | 0 | 0 | 0 | 0 |
| 222 | 0 | 0 | 0 | 0 |
| 223 | 0 | 0 | 0 | 0 |
| 224 | 0 | 0 | 0 | 0 |
| 225 | 0 | 0 | 0 | 0 |
| 226 | 0 | 0 | 0 | 0 |
| 227 | 0 | 0 | 0 | 0 |
| 228 | 0 | 0 | 0 | 0 |
| 229 | 0 | 0 | 0 | 0 |
| 230 | 0 | 0 | 0 | 0 |
| 231 | 0 | 0 | 0 | 0 |
| 232 | 0 | 0 | 0 | 0 |
| 233 | 0 | 0 | 0 | 0 |
| 234 | 0 | 0 | 0 | 0 |
| 235 | 0 | 0 | 0 | 0 |
| 236 | 0 | 0 | 0 | 0 |
| 237 | 0 | 0 | 0 | 0 |
| 238 | 0 | 0 | 0 | 0 |
| 239 | 0 | 0 | 0 | 0 |
| 240 | 0 | 0 | 0 | 0 |
| 241 | 0 | 0 | 0 | 0 |
| 242 | 0 | 0 | 0 | 0 |
| 243 | 0 | 0 | 0 | 0 |
| 244 | 0 | 0 | 0 | 0 |
| 245 | 0 | 0 | 0 | 0 |
| 246 | 0 | 0 | 0 | 0 |
| 247 | 0 | 0 | 0 | 0 |
| 248 | 0 | 0 | 0 | 0 |
| 249 | 0 | 0 | 0 | 0 |
| 250 | 0 | 0 | 0 | 0 |
| 251 | 0 | 0 | 0 | 0 |
| 252 | 0 | 0 | 0 | 0 |
| 253 | 0 | 0 | 0 | 1 |
| 254 | 0 | 0 | 0 | 1 |
| 255 | 0 | 0 | 0 | 1 |
| 256 | 0 | 0 | 0 | 1 |

**Figure 7.** Comparison Results



| | | | | |
|---|---|---|---|---|
| 257 | 0 | 0 | 0 | 1 |
| 258 | 0 | 0 | 0 | 1 |
| 259 | 0 | 0 | 0 | 0 |
| 260 | 0 | 0 | 0 | 0 |
| 261 | 0 | 0 | 0 | 0 |
| 262 | 0 | 0 | 0 | 0 |
| 263 | 0 | 0 | 0 | 0 |
| 264 | 0 | 0 | 0 | 0 |
| 265 | 0 | 0 | 0 | 0 |
| 266 | 0 | 0 | 0 | 0 |
| 267 | 0 | 0 | 0 | 0 |
| 268 | 0 | 0 | 0 | 0 |
| 269 | 0 | 0 | 0 | 0 |
| 270 | 0 | 0 | 0 | 0 |
| 271 | 0 | 0 | 0 | 0 |
| 272 | 0 | 0 | 0 | 0 |
| 273 | 0 | 0 | 0 | 0 |
| 274 | 0 | 1 | 0 | 1 |
| 275 | 0 | 1 | 0 | 1 |
| 276 | 0 | 1 | 0 | 1 |
| 277 | 0 | 1 | 0 | 1 |
| 278 | 0 | 1 | 0 | 1 |
| 279 | 0 | 1 | 0 | 1 |
| 280 | 0 | 1 | 0 | 1 |
| 281 | 0 | 1 | 0 | 1 |
| 282 | 0 | 1 | 0 | 1 |
| 283 | 0 | 1 | 0 | 1 |
| 284 | 0 | 1 | 0 | 1 |
| 285 | 0 | 1 | 0 | 1 |
| 286 | 0 | 1 | 0 | 1 |
| 287 | 0 | 1 | 0 | 1 |
| 288 | 0 | 1 | 0 | 1 |
| 289 | 0 | 1 | 0 | 1 |
| 290 | 0 | 1 | 0 | 1 |
| 291 | 0 | 1 | 0 | 1 |
| 292 | 0 | 1 | 0 | 1 |
| 293 | 0 | 1 | 0 | 1 |
| 294 | 0 | 1 | 0 | 1 |
| 295 | 0 | 1 | 0 | 1 |
| 296 | 0 | 1 | 0 | 1 |
| 297 | 0 | 1 | 0 | 1 |
| 298 | 0 | 1 | 0 | 1 |
| 299 | 0 | 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 300 | 0 | 1 | 0 | 1 |
| 301 | 0 | 1 | 0 | 1 |
| 302 | 0 | 1 | 0 | 1 |
| 303 | 0 | 1 | 0 | 1 |
| 304 | 0 | 1 | 0 | 1 |
| 305 | 0 | 1 | 0 | 1 |
| 306 | 0 | 1 | 0 | 1 |
| 307 | 0 | 1 | 0 | 1 |
| 308 | 0 | 1 | 0 | 1 |
| 309 | 0 | 1 | 0 | 1 |
| 310 | 0 | 1 | 0 | 1 |
| 311 | 0 | 1 | 0 | 1 |
| 312 | 0 | 1 | 0 | 1 |
| 313 | 0 | 1 | 0 | 1 |
| 314 | 0 | 1 | 0 | 1 |
| 315 | 0 | 1 | 0 | 1 |
| 316 | 0 | 1 | 0 | 1 |
| 317 | 0 | 1 | 0 | 1 |
| 318 | 0 | 1 | 0 | 1 |
| 319 | 0 | 1 | 0 | 1 |
| 320 | 0 | 1 | 0 | 1 |
| 321 | 0 | 1 | 0 | 1 |
| 322 | 0 | 1 | 0 | 1 |
| 323 | 0 | 1 | 0 | 1 |
| 324 | 0 | 1 | 0 | 1 |
| 325 | 0 | 1 | 0 | 1 |
| 326 | 0 | 1 | 0 | 1 |
| 327 | 0 | 1 | 0 | 1 |
| 328 | 0 | 1 | 0 | 1 |
| 329 | 0 | 1 | 0 | 1 |
| 330 | 0 | 1 | 0 | 1 |
| 331 | 0 | 1 | 0 | 1 |
| 332 | 0 | 1 | 0 | 1 |
| 333 | 0 | 1 | 0 | 1 |
| 334 | 0 | 1 | 0 | 1 |
| 335 | 0 | 1 | 0 | 1 |
| 336 | 0 | 1 | 0 | 1 |
| 337 | 0 | 1 | 0 | 1 |
| 338 | 0 | 1 | 0 | 1 |
| 339 | 0 | 1 | 0 | 1 |
| 340 | 0 | 1 | 0 | 1 |
| 341 | 0 | 1 | 0 | 1 |
| 342 | 0 | 1 | 0 | 1 |

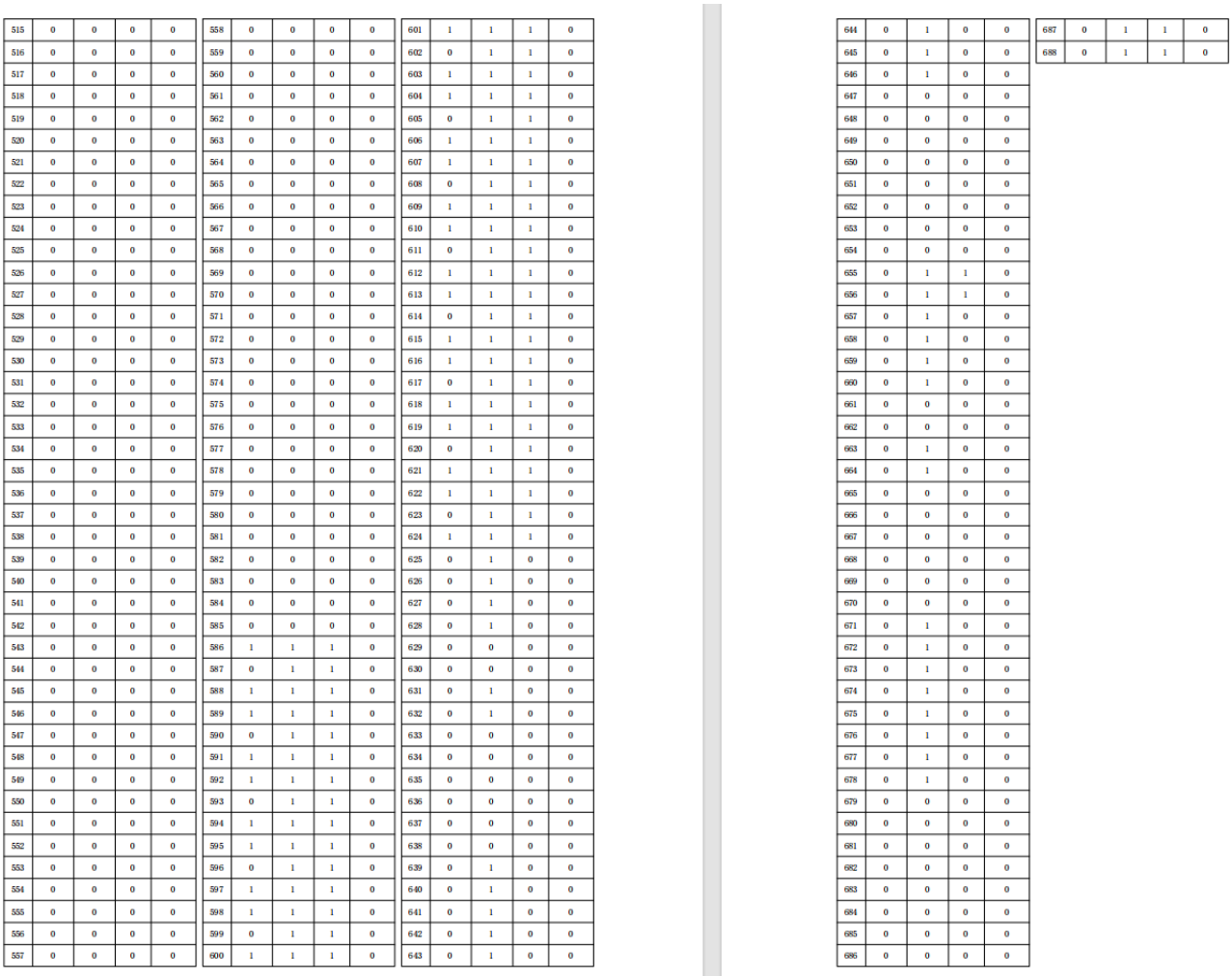
| | | | | |
|---|---|---|---|---|
| 343 | 0 | 1 | 0 | 1 |
| 344 | 0 | 1 | 0 | 1 |
| 345 | 0 | 1 | 0 | 1 |
| 346 | 0 | 1 | 0 | 1 |
| 347 | 0 | 1 | 0 | 1 |
| 348 | 0 | 1 | 0 | 1 |
| 349 | 0 | 1 | 0 | 1 |
| 350 | 0 | 1 | 0 | 1 |
| 351 | 0 | 1 | 0 | 1 |
| 352 | 0 | 1 | 1 | 1 |
| 353 | 0 | 1 | 1 | 1 |
| 354 | 0 | 1 | 1 | 1 |
| 355 | 0 | 1 | 1 | 1 |
| 356 | 0 | 1 | 1 | 1 |
| 357 | 0 | 1 | 1 | 1 |
| 358 | 0 | 1 | 1 | 1 |
| 359 | 0 | 1 | 1 | 1 |
| 360 | 0 | 1 | 1 | 1 |
| 361 | 0 | 1 | 1 | 1 |
| 362 | 0 | 1 | 1 | 1 |
| 363 | 0 | 1 | 1 | 1 |
| 364 | 0 | 1 | 1 | 1 |
| 365 | 0 | 1 | 1 | 1 |
| 366 | 0 | 1 | 1 | 1 |
| 367 | 0 | 1 | 1 | 1 |
| 368 | 0 | 1 | 1 | 1 |
| 369 | 0 | 1 | 1 | 1 |
| 370 | 0 | 1 | 0 | 1 |
| 371 | 0 | 1 | 0 | 1 |
| 372 | 0 | 1 | 0 | 1 |
| 373 | 0 | 1 | 0 | 1 |
| 374 | 0 | 1 | 0 | 1 |
| 375 | 0 | 1 | 0 | 1 |
| 376 | 0 | 1 | 1 | 1 |
| 377 | 0 | 1 | 1 | 1 |
| 378 | 0 | 1 | 1 | 1 |
| 379 | 0 | 1 | 1 | 1 |
| 380 | 0 | 1 | 1 | 1 |
| 381 | 0 | 1 | 1 | 1 |
| 382 | 0 | 1 | 1 | 1 |
| 383 | 0 | 1 | 1 | 1 |
| 384 | 0 | 1 | 1 | 1 |
| 385 | 0 | 1 | 1 | 1 |

| | | | | |
|---|---|---|---|---|
| 386 | 0 | 1 | 1 | 1 |
| 387 | 0 | 1 | 1 | 1 |
| 388 | 0 | 1 | 1 | 1 |
| 389 | 0 | 1 | 1 | 1 |
| 390 | 0 | 1 | 1 | 1 |
| 391 | 0 | 1 | 0 | 1 |
| 392 | 0 | 1 | 0 | 1 |
| 393 | 0 | 1 | 0 | 1 |
| 394 | 0 | 1 | 0 | 1 |
| 395 | 0 | 1 | 0 | 1 |
| 396 | 0 | 1 | 0 | 1 |
| 397 | 0 | 1 | 0 | 1 |
| 398 | 0 | 1 | 0 | 1 |
| 399 | 0 | 1 | 0 | 1 |
| 400 | 0 | 1 | 0 | 1 |
| 401 | 0 | 1 | 0 | 1 |
| 402 | 0 | 1 | 0 | 1 |
| 403 | 0 | 1 | 0 | 1 |
| 404 | 0 | 1 | 0 | 1 |
| 405 | 0 | 1 | 0 | 1 |
| 406 | 0 | 1 | 0 | 1 |
| 407 | 0 | 1 | 0 | 1 |
| 408 | 0 | 1 | 0 | 1 |
| 409 | 0 | 1 | 0 | 1 |
| 410 | 0 | 1 | 0 | 1 |
| 411 | 0 | 1 | 0 | 1 |
| 412 | 0 | 1 | 0 | 1 |
| 413 | 0 | 1 | 0 | 1 |
| 414 | 0 | 1 | 0 | 1 |
| 415 | 0 | 1 | 0 | 1 |
| 416 | 0 | 1 | 0 | 1 |
| 417 | 0 | 1 | 0 | 1 |
| 418 | 0 | 1 | 0 | 1 |
| 419 | 0 | 1 | 0 | 1 |
| 420 | 0 | 1 | 0 | 1 |
| 421 | 0 | 1 | 0 | 1 |
| 422 | 0 | 1 | 0 | 1 |
| 423 | 0 | 1 | 0 | 1 |
| 424 | 0 | 1 | 0 | 1 |
| 425 | 0 | 1 | 0 | 1 |
| 426 | 0 | 1 | 0 | 1 |
| 427 | 0 | 1 | 0 | 1 |
| 428 | 0 | 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 429 | 0 | 1 | 0 | 1 |
| 430 | 0 | 0 | 0 | 1 |
| 431 | 0 | 0 | 0 | 1 |
| 432 | 0 | 0 | 0 | 1 |
| 433 | 0 | 0 | 0 | 1 |
| 434 | 0 | 0 | 0 | 1 |
| 435 | 0 | 0 | 0 | 1 |
| 436 | 0 | 0 | 0 | 1 |
| 437 | 0 | 0 | 0 | 1 |
| 438 | 0 | 0 | 0 | 1 |
| 439 | 0 | 0 | 0 | 1 |
| 440 | 0 | 0 | 0 | 1 |
| 441 | 0 | 0 | 0 | 1 |
| 442 | 0 | 0 | 0 | 1 |
| 443 | 0 | 0 | 0 | 1 |
| 444 | 0 | 0 | 0 | 1 |
| 445 | 0 | 0 | 0 | 1 |
| 446 | 0 | 0 | 0 | 1 |
| 447 | 0 | 0 | 0 | 1 |
| 448 | 0 | 0 | 0 | 1 |
| 449 | 0 | 0 | 0 | 1 |
| 450 | 0 | 0 | 0 | 1 |
| 451 | 0 | 0 | 0 | 1 |
| 452 | 0 | 0 | 0 | 1 |
| 453 | 0 | 0 | 0 | 1 |
| 454 | 0 | 0 | 0 | 1 |
| 455 | 0 | 0 | 0 | 1 |
| 456 | 0 | 0 | 0 | 1 |
| 457 | 0 | 0 | 0 | 1 |
| 458 | 0 | 0 | 0 | 1 |
| 459 | 0 | 0 | 0 | 1 |
| 460 | 0 | 0 | 0 | 1 |
| 461 | 0 | 0 | 0 | 1 |
| 462 | 0 | 0 | 0 | 1 |
| 463 | 0 | 0 | 0 | 1 |
| 464 | 0 | 0 | 0 | 1 |
| 465 | 0 | 0 | 0 | 1 |
| 466 | 0 | 0 | 0 | 1 |
| 467 | 0 | 0 | 0 | 1 |
| 468 | 0 | 0 | 0 | 1 |
| 469 | 0 | 0 | 0 | 1 |
| 470 | 0 | 0 | 0 | 1 |
| 471 | 0 | 0 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 472 | 0 | 0 | 0 | 1 |
| 473 | 0 | 0 | 0 | 1 |
| 474 | 0 | 0 | 0 | 0 |
| 475 | 0 | 0 | 0 | 0 |
| 476 | 0 | 0 | 0 | 0 |
| 477 | 0 | 0 | 0 | 0 |
| 478 | 0 | 0 | 0 | 0 |
| 479 | 0 | 0 | 0 | 0 |
| 480 | 0 | 0 | 0 | 0 |
| 481 | 0 | 0 | 0 | 0 |
| 482 | 0 | 0 | 0 | 0 |
| 483 | 0 | 0 | 0 | 0 |
| 484 | 0 | 0 | 0 | 0 |
| 485 | 0 | 0 | 0 | 0 |
| 486 | 0 | 0 | 0 | 0 |
| 487 | 0 | 0 | 0 | 0 |
| 488 | 0 | 0 | 0 | 0 |
| 489 | 0 | 0 | 0 | 0 |
| 490 | 0 | 0 | 0 | 0 |
| 491 | 0 | 0 | 0 | 0 |
| 492 | 0 | 0 | 0 | 0 |
| 493 | 0 | 0 | 0 | 0 |
| 494 | 0 | 0 | 0 | 0 |
| 495 | 0 | 0 | 0 | 0 |
| 496 | 0 | 0 | 0 | 0 |
| 497 | 0 | 0 | 0 | 0 |
| 498 | 0 | 0 | 0 | 0 |
| 499 | 0 | 0 | 0 | 0 |
| 500 | 0 | 0 | 0 | 0 |
| 501 | 0 | 0 | 0 | 0 |
| 502 | 0 | 0 | 0 | 0 |
| 503 | 0 | 0 | 0 | 0 |
| 504 | 0 | 0 | 0 | 0 |
| 505 | 0 | 0 | 0 | 0 |
| 506 | 0 | 0 | 0 | 0 |
| 507 | 0 | 0 | 0 | 0 |
| 508 | 0 | 0 | 0 | 0 |
| 509 | 0 | 0 | 0 | 0 |
| 510 | 0 | 0 | 0 | 0 |
| 511 | 0 | 0 | 0 | 0 |
| 512 | 0 | 0 | 0 | 0 |
| 513 | 0 | 0 | 0 | 0 |
| 514 | 0 | 0 | 0 | 0 |

**Figure 8.** Comparison Results (Continued)



| | | | | |
|---|---|---|---|---|
| 515 | 0 | 0 | 0 | 0 |
| 516 | 0 | 0 | 0 | 0 |
| 517 | 0 | 0 | 0 | 0 |
| 518 | 0 | 0 | 0 | 0 |
| 519 | 0 | 0 | 0 | 0 |
| 520 | 0 | 0 | 0 | 0 |
| 521 | 0 | 0 | 0 | 0 |
| 522 | 0 | 0 | 0 | 0 |
| 523 | 0 | 0 | 0 | 0 |
| 524 | 0 | 0 | 0 | 0 |
| 525 | 0 | 0 | 0 | 0 |
| 526 | 0 | 0 | 0 | 0 |
| 527 | 0 | 0 | 0 | 0 |
| 528 | 0 | 0 | 0 | 0 |
| 529 | 0 | 0 | 0 | 0 |
| 530 | 0 | 0 | 0 | 0 |
| 531 | 0 | 0 | 0 | 0 |
| 532 | 0 | 0 | 0 | 0 |
| 533 | 0 | 0 | 0 | 0 |
| 534 | 0 | 0 | 0 | 0 |
| 535 | 0 | 0 | 0 | 0 |
| 536 | 0 | 0 | 0 | 0 |
| 537 | 0 | 0 | 0 | 0 |
| 538 | 0 | 0 | 0 | 0 |
| 539 | 0 | 0 | 0 | 0 |
| 540 | 0 | 0 | 0 | 0 |
| 541 | 0 | 0 | 0 | 0 |
| 542 | 0 | 0 | 0 | 0 |
| 543 | 0 | 0 | 0 | 0 |
| 544 | 0 | 0 | 0 | 0 |
| 545 | 0 | 0 | 0 | 0 |
| 546 | 0 | 0 | 0 | 0 |
| 547 | 0 | 0 | 0 | 0 |
| 548 | 0 | 0 | 0 | 0 |
| 549 | 0 | 0 | 0 | 0 |
| 550 | 0 | 0 | 0 | 0 |
| 551 | 0 | 0 | 0 | 0 |
| 552 | 0 | 0 | 0 | 0 |
| 553 | 0 | 0 | 0 | 0 |
| 554 | 0 | 0 | 0 | 0 |
| 555 | 0 | 0 | 0 | 0 |
| 556 | 0 | 0 | 0 | 0 |
| 557 | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 558 | 0 | 0 | 0 | 0 |
| 559 | 0 | 0 | 0 | 0 |
| 560 | 0 | 0 | 0 | 0 |
| 561 | 0 | 0 | 0 | 0 |
| 562 | 0 | 0 | 0 | 0 |
| 563 | 0 | 0 | 0 | 0 |
| 564 | 0 | 0 | 0 | 0 |
| 565 | 0 | 0 | 0 | 0 |
| 566 | 0 | 0 | 0 | 0 |
| 567 | 0 | 0 | 0 | 0 |
| 568 | 0 | 0 | 0 | 0 |
| 569 | 0 | 0 | 0 | 0 |
| 570 | 0 | 0 | 0 | 0 |
| 571 | 0 | 0 | 0 | 0 |
| 572 | 0 | 0 | 0 | 0 |
| 573 | 0 | 0 | 0 | 0 |
| 574 | 0 | 0 | 0 | 0 |
| 575 | 0 | 0 | 0 | 0 |
| 576 | 0 | 0 | 0 | 0 |
| 577 | 0 | 0 | 0 | 0 |
| 578 | 0 | 0 | 0 | 0 |
| 579 | 0 | 0 | 0 | 0 |
| 580 | 0 | 0 | 0 | 0 |
| 581 | 0 | 0 | 0 | 0 |
| 582 | 0 | 0 | 0 | 0 |
| 583 | 0 | 0 | 0 | 0 |
| 584 | 0 | 0 | 0 | 0 |
| 585 | 0 | 0 | 0 | 0 |
| 586 | 1 | 1 | 1 | 0 |
| 587 | 0 | 1 | 1 | 0 |
| 588 | 1 | 1 | 1 | 0 |
| 589 | 1 | 1 | 1 | 0 |
| 590 | 0 | 1 | 1 | 0 |
| 591 | 1 | 1 | 1 | 0 |
| 592 | 1 | 1 | 1 | 0 |
| 593 | 0 | 1 | 1 | 0 |
| 594 | 1 | 1 | 1 | 0 |
| 595 | 1 | 1 | 1 | 0 |
| 596 | 0 | 1 | 1 | 0 |
| 597 | 1 | 1 | 1 | 0 |
| 598 | 1 | 1 | 1 | 0 |
| 599 | 0 | 1 | 1 | 0 |
| 600 | 1 | 1 | 1 | 0 |

| | | | | |
|---|---|---|---|---|
| 601 | 1 | 1 | 1 | 0 |
| 602 | 0 | 1 | 1 | 0 |
| 603 | 1 | 1 | 1 | 0 |
| 604 | 1 | 1 | 1 | 0 |
| 605 | 0 | 1 | 1 | 0 |
| 606 | 1 | 1 | 1 | 0 |
| 607 | 1 | 1 | 1 | 0 |
| 608 | 0 | 1 | 1 | 0 |
| 609 | 1 | 1 | 1 | 0 |
| 610 | 1 | 1 | 1 | 0 |
| 611 | 0 | 1 | 1 | 0 |
| 612 | 1 | 1 | 1 | 0 |
| 613 | 1 | 1 | 1 | 0 |
| 614 | 0 | 1 | 1 | 0 |
| 615 | 1 | 1 | 1 | 0 |
| 616 | 1 | 1 | 1 | 0 |
| 617 | 0 | 1 | 1 | 0 |
| 618 | 1 | 1 | 1 | 0 |
| 619 | 1 | 1 | 1 | 0 |
| 620 | 0 | 1 | 1 | 0 |
| 621 | 1 | 1 | 1 | 0 |
| 622 | 1 | 1 | 1 | 0 |
| 623 | 0 | 1 | 1 | 0 |
| 624 | 1 | 1 | 1 | 0 |
| 625 | 0 | 1 | 0 | 0 |
| 626 | 0 | 1 | 0 | 0 |
| 627 | 0 | 1 | 0 | 0 |
| 628 | 0 | 1 | 0 | 0 |
| 629 | 0 | 0 | 0 | 0 |
| 630 | 0 | 0 | 0 | 0 |
| 631 | 0 | 1 | 0 | 0 |
| 632 | 0 | 1 | 0 | 0 |
| 633 | 0 | 0 | 0 | 0 |
| 634 | 0 | 0 | 0 | 0 |
| 635 | 0 | 0 | 0 | 0 |
| 636 | 0 | 0 | 0 | 0 |
| 637 | 0 | 0 | 0 | 0 |
| 638 | 0 | 0 | 0 | 0 |
| 639 | 0 | 1 | 0 | 0 |
| 640 | 0 | 1 | 0 | 0 |
| 641 | 0 | 1 | 0 | 0 |
| 642 | 0 | 1 | 0 | 0 |
| 643 | 0 | 1 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 644 | 0 | 1 | 0 | 0 |
| 645 | 0 | 1 | 0 | 0 |
| 646 | 0 | 1 | 0 | 0 |
| 647 | 0 | 0 | 0 | 0 |
| 648 | 0 | 0 | 0 | 0 |
| 649 | 0 | 0 | 0 | 0 |
| 650 | 0 | 0 | 0 | 0 |
| 651 | 0 | 0 | 0 | 0 |
| 652 | 0 | 0 | 0 | 0 |
| 653 | 0 | 0 | 0 | 0 |
| 654 | 0 | 0 | 0 | 0 |
| 655 | 0 | 1 | 1 | 0 |
| 656 | 0 | 1 | 1 | 0 |
| 657 | 0 | 1 | 0 | 0 |
| 658 | 0 | 1 | 0 | 0 |
| 659 | 0 | 1 | 0 | 0 |
| 660 | 0 | 1 | 0 | 0 |
| 661 | 0 | 0 | 0 | 0 |
| 662 | 0 | 0 | 0 | 0 |
| 663 | 0 | 1 | 0 | 0 |
| 664 | 0 | 1 | 0 | 0 |
| 665 | 0 | 0 | 0 | 0 |
| 666 | 0 | 0 | 0 | 0 |
| 667 | 0 | 0 | 0 | 0 |
| 668 | 0 | 0 | 0 | 0 |
| 669 | 0 | 0 | 0 | 0 |
| 670 | 0 | 0 | 0 | 0 |
| 671 | 0 | 1 | 0 | 0 |
| 672 | 0 | 1 | 0 | 0 |
| 673 | 0 | 1 | 0 | 0 |
| 674 | 0 | 1 | 0 | 0 |
| 675 | 0 | 1 | 0 | 0 |
| 676 | 0 | 1 | 0 | 0 |
| 677 | 0 | 1 | 0 | 0 |
| 678 | 0 | 1 | 0 | 0 |
| 679 | 0 | 0 | 0 | 0 |
| 680 | 0 | 0 | 0 | 0 |
| 681 | 0 | 0 | 0 | 0 |
| 682 | 0 | 0 | 0 | 0 |
| 683 | 0 | 0 | 0 | 0 |
| 684 | 0 | 0 | 0 | 0 |
| 685 | 0 | 0 | 0 | 0 |
| 686 | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 687 | 0 | 1 | 1 | 0 |
| 688 | 0 | 1 | 1 | 0 |

**Figure 9.** Comparison Results (Continued)

# Demonstration and Defense of GoFetch Attack

Zebang Fei
Southern University of Science and Technology
Shenzhen, Guangdong, China
12110608@mail.sustech.edu.cn

Ben Chen
Southern University of Science and Technology
Shenzhen, Guangdong, China
chenb2022@mail.sustech.edu.cn

Jiarun Zhu
Southern University of Science and Technology
Shenzhen, Guangdong, China
12210212@mail.sustech.edu.cn

Zhongwen Chen
SiChuan University
Chendu, Sichuan, China
2022141480020@stu.scu.edu.cn

Qingyang Zhang
Xi'an Jiao Tong University
Xi'an, Shaanxi, China
2203314931@xjtu.edu.cn

## ABSTRACT

Prefetcher has been widely applied in the caches of modern processors to boost the cache efficiency. However, its existence violates the constant-time programming paradigm that resists the Cache-based Side-Channel Attacks (SCA), making processors exposed to SCAs once more. To be worse, reseachers have found that Apple's M-series processors adapt a more vulnerable prefetcher that would dereference/prefetch the data pointed by cache content, which magnifies the flaw. GoFetch attack is thereby proposed to trigger that vulnerability. With the aim to give a comprehensive explanation about this brand-new attack, we develope a educational website with animation and reproduce the attack to clearly address it. We also adapt similar detection and prevention from cache SCAs to mitigate it, which is verified by our experiment. We believe that this paper will provide programmers with an distinct idea of principles and prevention of GoFetch attack, to avoid such issue in software.

## KEYWORDS

Side-channel attack, prefetcher-based side channel, Visualization of cache side-channel, Side-channel detection and prevention

## 1 INTRODUCTION

Side-channel attacks are a type of security exploit that leverage physical characteristics of a system, rather than exploiting algorithmic vulnerabilities, to extract sensitive information. Recent studies have demonstrated the evolving complexity and efficacy of side-channel attacks. Jin et al. presented a new side-channel attack on Intel CPUs that relies on timing transient execution to extract sensitive information [7]. Similarly, Kogler et al. introduced the Collide+Power attack, which measures CPU power consumption to deduce cryptographic keys [9]. Moreover, Pinto and Rodrigues showcased a side-channel attack on ARM TrustZone, highlighting the vulnerability of microcontrollers to these attacks [13].

GoFetch is a recently proposed side-channel attack method that was first introduced in 2023 [4]. This attack exploits the Data Memory-dependent Prefetcher (DMP) found in Apple's M-series CPUs. Unlike traditional cache-based SCAs, GoFetch targets at the behaviour of pre-dereference of pointer present in caches. By carefully crafting malicious eviction set and attacking with Prime&Probe, the attacker can infer sensitive information from the timing variations caused by the prefetching mechanism. This allows GoFetch to break constant-time cryptographic implementations.

In this paper, we will delve into the GoFetch attack, attempt to explain the principles of the GoFetch attack in a novel and comprehensive way via visualization website of the attack scheme. Based on the understanding, we try to reproduce, detect, defend against it. We managed to reproduce the chosen-input attack against the Constant-time Swap and Go's RSA encryption. To go beyond, we implement effective defense methods against the attack on software level. While the defense scheme suffers from performance loss known as "security tax", we measure the actual lost for each defense method, providing detail for trade-off. We also adapt side-channel detection techniques to detect attacks exploiting DMP, using dynamic detection of malicious inputs and priming in macOS.

## 2 BACKGROUND

### 2.1 Side Channel Attacks

In contrast to the mathematical models used in cryptanalysis, side-channel attacks target at the actual hardware systems implementing the cryptographic algorithms. These attacks exploit certain characteristics of specific hardware systems and carry out physical attacks on devices to steal secret from them. Compared with direct attack, SCAs are more powerful but less accurate.

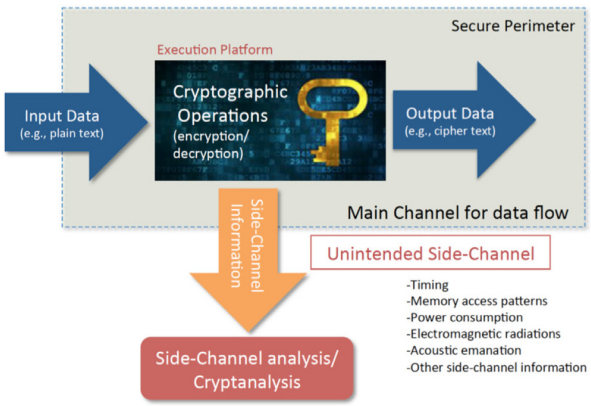



**Figure 1: Side-channel attack model**

As Figure 1 indicates[1], side-channel attacks (SCAs) exploit devices by the unintentional and largely unavoidable signals of it to steal secrets like private keys and sensitive information from

victim machines. Although the victim might be a black or grey box for attacker, the attacker can utilize the information gathered at their best.

## 2.2 Cache

Cache is a smaller, faster memory component located closer to the CPU, designed to store copies of frequently accessed data from the main memory. Typically, a modern CPU has one isolated L1 cache for each core and shared L2 cache within a bundle of core. Note that to ensure memory consistency, content of L2 caches is inclusive of that of L1 caches. In other words, what is not present in L2 will never be present in L1. For specification of Apple's M1 processor, it uses 128 KB, 8-way associative L1 cache for each core and shared 12 MB, 12-way associative L2 cache for each 4 cores [16].

## 2.3 Cache-based Side Channel Attacks

Cache side-channel attacks leverage the differences in access times between cache hits (when data is present from the cache) and cache misses (when data is retrieved from the main memory). By measuring these access times, attackers can infer sensitive information about the data being processed [3, 11, 12]. Time-driven attacks measure the total execution time of cryptographic operations [3]. Trace-driven attacks monitor the processor's cache access patterns [11]. Figure 2 shows the attack path of Prime&Probe attack[8],

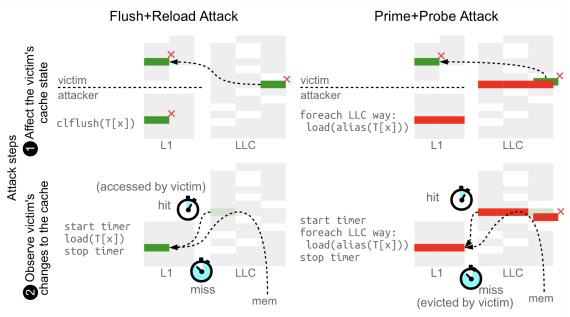



**Figure 2: Flush&Reload and Prime&Probe Attack**

which will used in GoFetch. Prime&Probe attacks involve the attacker filling the cache with their data (Prime) and measuring access times to infer cache usage by the victim process (Probe) [12]. From the statistical analysis of the timing difference in access, we can deduce the threshold that distinguish whether the target memory is accessed by victim.

## 2.4 Data Memory-dependent Prefetcher (DMP)

As of 2024, data prefetching has already been a common feature in modern CPUs, which fetches the data from memory before the actual access, by monitoring memory access patterns and predicting the possible next access. To boost more performance, the data memory-dependent prefetcher (DMP) takes the step further to search cache content for possible pointer values, and pre-fetch the target data of pointers under some criteria and restriction [14]. The detail found by Chen et al[4] will be explained at Section 3.1

# 3 UNDERSTANDING GOFETCH ATTACK

In GoFetch attack, attackers can exploit the behavior of DMP to infer the protected data patterns. The DMP in Apple's M1 processor architecture was demonstrated to be used as a memory side-channel in an attack published in early 2024 [6]. The DMP was subsequently discovered to be even more opportunistic than previously thought, and has now been demonstrated to be able to be used to effectively attack the constant-time cryptographic algorithms [4].

## 3.1 Apple's DMP Mechanism

Apple's DMP behaviour can be generalized as follows. DMP continuously monitors the processor's memory accesses and records the addresses and data being accessed. It analyzes these patterns and contents to identify potential pointers within the data. Once a pointer is identified, DMP attempts to predict future memory accesses by preloading the data located at these pointer addresses into the cache. This proactive data loading enhance the cache-hit rate. Furthermore, DMP's sophisticated filtering mechanisms, such as the history filter and do-not-scan hint, help in maintaining efficiency by avoiding redundant operations. However, the extreme proactivity makes it much simpler for attacker to maliciously trigger the DMP, raising the security concerns.

Through preliminary work by Chen et al[4], we can understand that the DMP used by Apple has the following characteristics, which are precisely utilized in subsequent attacks.

*Accessing Pattern.* DMP is an advanced prefetcher as mentioned, meaning that accessing an array of data will activate the pre-fetch of adjacent data. Beyond that, DMP will automatically dereference/fetch `data` pointed by `ptr` when accessing `ptr` with both implicit and explicit dereference of `ptr`. Experiment also suggests that DMP dereferences the adjacent `ptrs` with the same array and within the neighboring cache lines. These loose activation patterns imply a critical security issue.

*Activation.* To avoid redundant prefetches, DMP uses a history filter to record pointers that have already been dereferenced. If a pointer exists in the history filter, DMP will not dereference it again. Additionally, DMP uses a "do-not-scan" hint in complement of history filter. If a target data is brought and then evicted from caches with its `ptr` present in cache, DMP does not re-dereference `ptr` even if no related record in history filter, unless `ptr` is evicted from both caches. This mechanism plays significant role in GoFetch exploitation.

*Restriction.* DMP will only dereference pointers located within the same 4GB-aligned region. This means that DMP will only dereference pointers whose upper 32 bits of their addresses match those of the target address. The restriction should be considered essentially since it's the major reasons for dysfunction of eviction set.

We can utilized these characteristics of Apple's DMP in leaking sensitive information. Specifically, GoFetch leverages the DMP's content-driven prediction mechanism to deduce the victim program's behaviour. Based on the understanding of DMP's behaviour and by carefully crafting memory content and access patterns, an attacker can manipulate the DMP's activation on special condition (when the victim program is running a snippet or not), and then monitor that activity.

## 3.2 Threat Model

The GoFetch attack targets various cryptographic implementations. For example, although constant-time implementations aim to defend against side-channel attacks by eliminating timing differences, GoFetch circumvents these defenses by leveraging the characteristics of DMP. . The section provides an overview of exploitation using the characteristics of DMP.

*3.2.1 Prerequisite.* The capabilities of attacker are assumed as

a) **Observation of Memory Access Patterns**: The attacker should share the L2 cache with victim. Concurrent high-load application will bring noise to the observation which must be avoided.
b) **Chosen Input**: The attacker can inject arbitrary data into the victim to trigger the DMP to prefetch, thereby leaking sensitive data. Otherwise attacker should possess sufficient information.
c) **Limited System Access**: The attacker does not need full control over the system, restricted privilege as a normal user program would be enough for the attack.

*3.2.2 Attack Scheme.* Below shows the attack routine

i. **Timing Source**: As the unprivileged attacker has no access to the high-resolution counter provided by OS, it has to prepare the timer as an independent thread to gain high precision of record. The timer thread acts normally as a user thread.
ii. **Generate Evictset**: To perform a standard Prime&Probe attack, the attack will have to generate a compound eviction set that evicts both victim array (memory that leaks secret) and `ptr` (address of probe array).
iii. **Inject Inductive Patterns**: The attacker injects specific data access patterns to make the DMP start prefetching data from the target memory region.
iv. **Evicting Target**: Attacker will keep evicting the target array and `ptr` with evictset generated in ii. The objective is to ensure that the DMP only activates under specific situation, which is explained in detail at Appendix A.
v. **Capture Leaked Data**: By monitoring the access time of probe array `ptr`, attacker can identify the activation of DMP so that it can capture the side channel signals to deduce the secret.
vi. **Analysis**: Attacker will finally analyze the leakage data and recover the secret through statistics, i.e., set a threshold by the latency with peak frequency, shown in Section 6.2

# 4 VISUALIZATION

With sufficient understanding of the GoFetch attack, we develop an animated website for education purpose where the audience receives a quick understanding of the attack by interacting with the pages. In this section, the design principles and expected effects of our website are comprehensively explained.

## 4.1 Design Principles

The website is mainly intended for the public, rather than the professional. The webpage starts with a introduction showing up. As in Figure 3, the audience will at once read the digest of GoFetch attack. If a term confuses the audience, he can naturally click the term before the brief explanation box pops up. In case that the audience ignores the clickable term, the webpage highlights three terms with large blocks in background knowledge section.

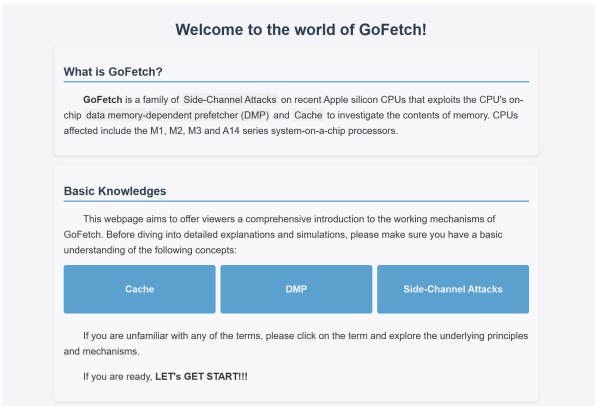



**Figure 3: Homepage of GoFetch introduction website**

The website provides a detailed exploration of these three terms. Next, we put a explanation of GoFetch attack on constant-time swap in simple language, after which the audience will enter the interactive part of our website.

## 4.2 Animation and Interaction

Understanding the text might be tricky for some of the audience. Therefore, we build three interactive webpages with animation for demo of Cache, Side-Channel and GoFetch.

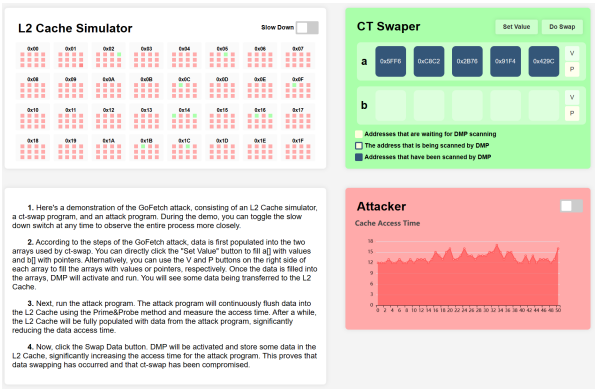



**Figure 4: Interactive demo of GoFetch attack on ct-swap**

Figure 4 shows the interface of GoFetch simulation where the audience can act like an attacker and victim at the same time. The items display the micro view of L2 cache, constant-time swap victim and GoFetch attacker. We highlight victim with green and attacker with red in their items and L2 cache. Once attacker initializes its attack, the audience could immediately be notified by the changes in cache and attacker's timing diagram. To simulate the attack, the audience will perform the victim's swap action, after which the changes in attacker's eviction begin to display.

Additionally, we also develop the simulation for the background knowledge about Cache and Side-Channel. For cache simulation, the audience could act as an user to fetch data, and then see how

it's stored in cache. For side channel simulation, we borrow an interesting scenario that the theft breaks the lock by sound emitted in trial of unlock. The audience under guidance of slight difference in sound could guess the password effectively.

## 5 MITIGATION

In this section, we introduce the implementation of defense methods against GoFetch attack. In next section, we analyze the trade-off of these defenses, providing information for selection of them.

### 5.1 Disabling DMP

The intuitive idea is simple and straight: disable the flaw opponent, DMP. For now as the architectural flaw has not been completely patched, we can only disable DMP in trade for a better security when performing security-critical tasks, temporarily. This could be done by setting the DIT bit through syscall of macOS. A famous hacker Hector Martin found that Apple has already done so, by only allowing DMP work at user mode in M2/M3 processors [10]. Surely, disabling DMP will cost a slowdown in performance and raise in cache miss rate. We measured the "security tax" to find out how much the, whose results are compared in Section 6.2

### 5.2 Using Efficiency Core

As Augury reads [14], the DMP is only available on efficiency core (Icestorm). Thereby, it could be a short-term solution for latency-insensitive applications. macOS provides system interfaces for manual setting of scheduler, like binding program to specific core using `CPU_SET()`. However, no one knows if Apple will enable DMP on efficiency cores someday, and by then, nothing could be done to prevent GoFetch unless Apple had a better solution.

To give a more intuitive impression on how much slower it would cause, we also measure the time consumption on a typical algorithm. To magnify the slowdown caused by accessing cache/memory, we use sorting algorithm that takes a relatively high portion in accessing data, as the experiment sample.

### 5.3 Blinding

Blinding is a generalized name for the optimized solution which shared the same characteristics. It obfuscates the secrets in side-channel signals by adding noises to either the secret or the pointers (specially for DMP). The obfuscation mostly uses masking techniques like exclusive-or and symmetric cryptography. Note that we have to maintain the normal execution and ensure the correctness of program when blinding.

To fight against DMP-related attack, the idea is to disable the pointers. For instance, in constant-time swap, we can add a mask to the inputs to hide the pointer, as in Listing 1

Without any pointers present in arrays, the DMP never gets activation and attacker cannot effectively guess the value of `mask` either. Attacker fail to distinguish and DMP activation and the secret bit. Another binding for Constant-time Swap is that, as masking always produce `secret=0` for attacker since DMP never activates, we can also try to making `secret=1` by swapping a and b at the beginning and again at the end to blind attacker.

Similarly in RSA decryption, we can apply a mathematical mask rather than bitwise xor to ensure the correct decryption result.

```
void ct-swap(uint64_t secret, uint64_t *a, uint64_t *b,
        size_t len) {
    uint64_t tmp_a[len+1], tmp_b[len+1];
    uint64_t mask = (uint64_t) rand();
    for (size_t i = 0; i < len; i++) {
        tmp_a[i] = a[i] ^ mask; // masking
        tmp_b[i] = b[i] ^ mask;
    }

    // swap the temp arrays in constant time

    for (size_t i = 0; i < len; i++) {
        a[i] = tmp_a[i] ^ mask; // unmasking
        b[i] = tmp_b[i] ^ mask;
    }
}
```

**Listing 1: Masking Pointers in CT-Swap**

```
void rsa-decrypt(uint64_t ...) {
    uint64_t mask = (uint64_t) rand();
    uint64_t inv = pow(mask, -1, phi(N)); // masking
    cipher = pow(cipher, mask, N);

    // normal RSA decryption

    plain = pow(plain, inv, N); // unmasking
}
```

**Listing 2: Masking ciphertext using Math**

Listing 2 is barely a scratch, as CRT decryption acceleration can be adapted here for the masking and unmasking, too. However, it does double the time consumption of decryption, and possibly makes the CRT acceleration meaningless.

### 5.4 Detection

As mentioned in previous sections, all methods that can defence GoFetch attack will reduce the performance. To improve, an intuitive idea is to detect the possible existence of attacker, after which the compromised computer could protect the secret in victim from being stolen. Researchers found that most of the cache SCA detection techniques are based on Hardware Performance Counter (HPC) [1].

The GoFetch attacker need to build eviction set to evict the data of victim out from the cache and record time with precision of nanoseconds[4]. The characteristic of GoFetch attacker shown in Section 3.2 inspires us with a clear instruction to locate it. Similar to techniques in previous work [1, 2, 5, 15, 16], we come up multiple methods to detect the attack on DMP. List below is the methods that we try to adapt in the detection. Notice that we use the constant-swap as victim example, which can be generalized with little modification.

*5.4.1 Static Detection of Binary File.* The first idea we come up with is to scan the binary file to find out whether it frequently

invokes dangerous syscalls like `kpc_get_counter_count()` to use high-precision timer and/or request something in memory again and again in a short time period. However, both of them seems unreliable due to the limitation of attacker's access privilege.

It is trivial to find the frequent load/store instructions. The problem is, most of the harmless programs will also access memory frequently. For heavy load applications like video games, the memory usage can be far more frequent, even than the attacker needs. As a result, a mere detection of memory instructions can't help to distinguish attacker and folks.

*5.4.2 Dynamic Detection of Pointer Array.* Since it's impossible for us to detect GoFetch attack only by statistic detection, we turn to the dynamic detection to examine the inputs when the attacker behaves. One of the requirements to activate DMP is that the data present in cache lines looks like pointers. Under most situation, these pointer-like data need to be injected by GoFetch attacker. It seems possible to detect attacking through detect the parameter passing and search for arrays containing continuous pointer-like data.

This method seems better than above, but we met difficulties when trying to monitor the communication between progresses. To resolve this in interprocess communication, we create a isolated environment with the suspect process in it and attach a pipe monitor to suspect using system tools. For TCP communication using loopback, we can simply cover the listening port of vulnerable victim with proxy and expose the proxy port.

Another identified obstacle forbids us to distinguish the malicious process and normal process, since a normal process could've contained arrays of pointers. Therefore, the only way out is combining the detection of pointer array along with detection of malicious priming behaviour, which leads to the section.

*5.4.3 Dynamic Detection of Priming.* As the threat model suggests, GoFetch attack takes effect only under victim process is running. This means that we can choose to attach the detector on victim to detect possible attacker, instead of building a daemon that keeps monitoring all processes. Prime&Probe attack can be detected by measuring conflict miss rates, which caused a much longer time to read/write cache in the victim process.

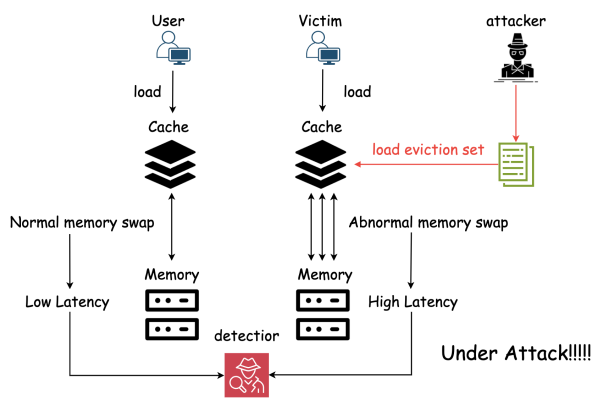



**Figure 5: GoFetch attack detection model**

In our model, when the progress begin, it record the time of access an array in cache without the eviction from attacker as the threshold. After that, before and when constant-time swap performs, detector measures access time of the array in charge. We shall note that for security concern, a fake swap might be done before actual swap. If no attacker is active, the chance of the array missing in cache is trivial, while it will almost immediately swapped out if there's an attacker evict it. So the timing difference again

For implementation, we integrate the detector process into the victim process as its own guard. Since HPC requires the kernel privilege, we use similar approach from the attacker and launch a independent thread of timer. Next, we keep monitoring the access time of victim's arrays and count the number of cache miss. We consider it abnormal for large amount of cache miss since constant-swap will firstly access the arrays to obtain `delta` in Appendix. Through fine-tune, we find a optimized threshold for the abnormal outcomes.

# 6 EVALUATION

In this section, we will present thoroughly the result of our experiments on reproducing the attack and effectively defending against it. Next, we're going to analyze the data to demonstrate the actual meaning of it in proof of the success in attack and defend.

## 6.1 Environment Setup

The experiment is done under the following environment

| | |
|---|---|
| SoC | Apple M1 8G Unified Memory |
| OS | macOS Sonoma 14.5 (23F79) |
| Model | MacBook Air (A2337) |

To ensure that DMP is genuinely available in the environment, we also reproduce the reverse engineering of DMP as described in Augury and GoFetch and the result matches that present in previous work[4, 14]. Meanwhile, we found a way to disable DMP[10] which distinctly shows the access-time difference between with and without DMP, supported by the real latency of 335 cycles and 585 cycles at average, respectively.

## 6.2 Latency Analysis in Attack

For the attack discussed in Section 3, we implement a attacker with techniques to identify victim address space, generate eviction set and monitor the latency in accessing the probe array. Before attacking, the process will ensure the functionality of eviction sets. We conducted 10 experiments for the two attacks with 8 of them success for constant-time swap, and the failure is discussed in Appendix A. During the experiment, we measured the actual latency in accessing the probe array `data`. The result is shown in Figure 6

The attack on constant-time swap and RSA-2048 has shown similar latency diagram. As we can see from Figure 6, the threshold between `secret=0` and `secret=1` lies around 700 cycles. For the key recovery in RSA-2048, the result analysis shows in Table

## 6.3 Performance Loss for Defense

To estimate the performance loss of hardware defenses in Section 5.1 and Section 5.2, we conducted 100 experiments to record the

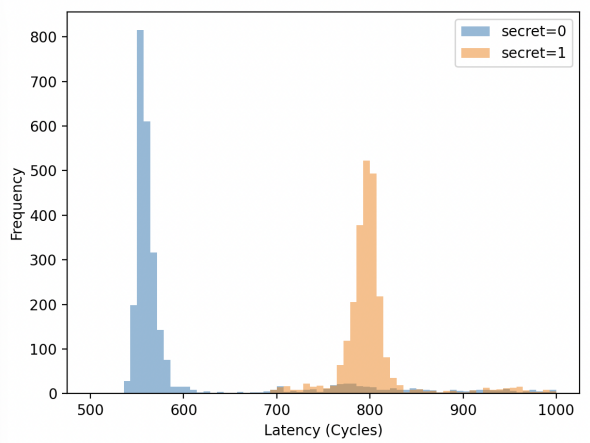



**Figure 6: Latency difference in Prime&Probe**

**Table 1: Success rate RSA key recovery of $p$**

| Key Recovery Rate | Success Rate | Avg. Running Time |
|---|---|---|
| 512 of 1024 bits | 99.8% | 53 min |
| 560 of 1024 bits | 90.7% | 72 min |

execution time with defense and without defense, with each experiment running the same algorithm for 100,000 times. The overall average result in shown in Table 2

| | DMP Enabled | DMP Disabled | Efficiency Core |
|---|---|---|---|
| Time | 81.9 us | 82.3 us | 378.9 us |
| Miss Rate | 0.09% | 0.83% | 2.17% |
| Slowdown | 0 | 1.46% | 362.64% |

**Table 2: Performance with DMP enabled/disabled, and efficiency core**

The slowdown of disabling DMP seems acceptable. With practice in designing processors, we consider the result normal, largely for the experience that improvement in cache hit rate eventually performs less satisfyingly than improvement in cache capacity or cache access latency. In contrast, the chips that cannot disable DMP will have to transfer to a slower core, resulting in a nearly 4 times slower overhead. As in the specification of experiment machine, the default clock rate is 2.064 GHz, compared with 3.2 GHz in performance core. The increase in cache miss rate also contributes to the slowdown. Nevertheless, all the mitigation is proven to be successful.

We also measure the cost for adding extra instructions to mask the pointers. For this experiment, due to the negligible operation, we repeat the codes for 100,000 times to magnify the execution time and acquire a high precision in record. For Constant-Swap, we test swapping an array of 1,000,000 elements; for RSA, we rewrote the program and used library in C language with the same implementation (mostly the RSA and math-related libraries).

| | CT-Swap | Blind CT-Swap | RSA | Blind RSA |
|---|---|---|---|---|
| Time | 1073 us | 3234 us | 159 us | 1063 us |
| Slowdown | 0 | 201.5% | 0 | 568.6% |
| Defense | fail | 100% | fail | 100% |

**Table 3: Performance loss of Blinding in CT-Swap and RSA**

Defense in software level could be much more lagging than hardware prevention since extra program code increases the time complexity by multiplying a constant value, as shown in Table 3.

## 6.4 Detection Accuracy

We conduct 10 experiment to examine our detector model in various situation with result shown below.

| | Attack | Normal | Attack (Noise) | Swap (Noise) |
|---|---|---|---|---|
| `secret=0` | Y | N | Y/N: 70%/30% | Y/N: 40%/60% |
| `secret=1` | Y | N | Y/N: 80%/20% | Y/N: 40%/60% |

**Table 4: Detection result under specific situation (Y/N for attacker detected or not**

Table 4 shows that our implementation effectively distinguish the attacker and user. However, detector is still under influence of the environment noise. We hope to improve our detection techniques with more consideration on attacker's profile and conduct more experiment to obtain higher success rate in the future.

# 7 CONCLUSION

In this paper, we first explained the principles of the recent GoFetch attack and introduce our education website for its demonstration. By carefully design the webpages on content, layout, animation and interaction, the website can provide a quick and comprehensive understanding of GoFetch attack. We also successfully reproduce the attack to confirm the side-channel leakage, and implement several mitigation methods along with detection of possible GoFetch attacks. For attack, we analyze the latency result to show the leakage. For prevention, we measure and compare the overall performance loss in different methods. For detection, we identify the difficulties in monitoring attacker's behaviour and implement a accurate GoFetch detector with a combination of detection.

# REFERENCES

[1] Ayaz Akram, Maria Mushtaq, Muhammad Khurram Bhatti, Vianney Lapotre, and Guy Gogniat. Meet the sherlock holmes' of side channel leakage: A survey of cache sca detection techniques. *IEEE Access*, 8:70836–70860, 2020.

[2] Mohammad-Mahdi Bazm, Thibaut Sautereau, Marc Lacoste, Mario Südholt, and Jean-Marc Menaud. Cache-Based Side-Channel Attacks Detection through Intel Cache Monitoring Technology and Hardware Performance Counters. In *FMEC 2018 - Third IEEE International Conference on Fog and Mobile Edge Computing*, pages 1–6, Barcelona, Spain, April 2018. IEEE.

[3] Daniel J Bernstein. Cache-timing attacks on aes. *University of Illinois at Chicago*, 152:1–5, 2005.

[4] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W. Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. Gofetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *USENIX Security*, 2024.

[5] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Prefetch-guard: Leveraging hardware prefetches to defend against cache timing channels. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 187–190, 2018.
[6] Dan Goodin. Unpatchable vulnerability in apple chip leaks secret encryption keys. *Ars Technica*, 2024.
[7] Yu Jin, Hans-Wolfgang Loidl, and Joe Smith. Timing the transient execution: A new side-channel attack on intel cpus. *arXiv*, 2023.
[8] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *DAC '16: Proceedings of the 53rd Annual Design Automation Conference*. ACM, 6 2016.
[9] Andreas Kogler, Stefan Mangard, and Michael Schwarz. Collide+power: A new side-channel attack. *SecurityWeek*, 2023.
[10] Hector Martin. Found the DMP disable chicken bit, 4 2024.
[11] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.
[12] Colin Percival. Cache missing for fun and profit. In *BSDCan 2005*, volume 11, 2005.
[13] Sandro Pinto and Cristiano Rodrigues. New side-channel attack on arm trustzone. In *Proceedings of the Black Hat Asia Conference*, 2023.
[14] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W. Fletcher, and David Kohlbrenner. Augury: Using data memory-dependent prefetchers to leak data at rest. *IEEE Symposium on Security and Privacy (SP)*, pages 1491–1505, 2022.
[15] Younis A. Younis, Kashif Kifayat, and Abir Hussain. Preventing and detecting cache side-channel attacks in cloud computing. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, New York, NY, USA, 2017. Association for Computing Machinery.
[16] Jiyong Yu, Aishani Dutta, Trent Jaeger, David Kohlbrenner, and Christopher W. Fletcher. Synchronization storage channels (S2C): Timer-less cache Side-Channel attacks on the apple m1 via hardware synchronization instructions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1973–1990, Anaheim, CA, August 2023. USENIX Association.

# A EXPLOITATION

Section 3 analyzed the threat model and feasibility of GoFetch attack. To apply the attack in practice as a proof-of-concept, we conduct the experiment of two practical attacks. In this section, we firstly briefly introduce the background of the victim software and the techniques to leverage DMP's activation as leakage. Next, we will identify the difficulties in experiment.

## A.1 Attacking Constant Time Swap

Constant Time Programming (CTP) paradigm provides a effective practice to mitigate the Cache SCAs. The basic idea is that the execution time of program cannot be decided by the secret. For instance, Listing 3 shows a typical vulnerable swap function.

```
void vul-swap(uint64_t secret, uint64_t *a, uint64_t *b,
        size_t len) {
    uint64_t xored;
    if (secret) for (size_t i = 0; i < len; i++) {
        xored = a[i] ^ b[i];
        a[i] = a[i] ^ xored;
        b[i] = b[i] ^ xored;
    }
}
```

**Listing 3: Vulnerable Swap Code Snippet**

The code snippet above exposes two vulnerabilities related to SCA. The first is that for a large enough `len`, the execution time of `secret=1` can be clearly distinguished from `secret=0`, as the former is a non-zero number and the latter is nearly zero. While the other leakage is that if attacker performs a classical cache SCA, it can definitely observe the usage of `a` and `b` and thereby guess the secret with a probability close to 1. Listing 4 below shows the implementation of CTP.

```
void ct-swap(uint64_t secret, uint64_t *a, uint64_t *b,
        size_t len) {
    uint64_t delta;
    uint64_t mask = ~(secret-1);
    for (size_t i = 0; i < len; i++) {
        delta = (a[i] ^ b[i]) & mask;
        a[i] = a[i] ^ delta;
        b[i] = b[i] ^ delta;
    }
}
```

**Listing 4: Constant-time Swap Code Snippet**

The constant-time swap always accesses both `a` and `b` array, and iterates through the loop, which mostly eliminates the timing and cache difference. To attack on CTPs with DMP, we have an intuitive idea that to identify the swap action, the pointer must be dereferenced when being swapped. To achieve the assumption, we can carefully deploy the activation criteria and steps in Section 3.2.2. Figure 7 shows the difference in cache triggered by DMP on swap/no swap.

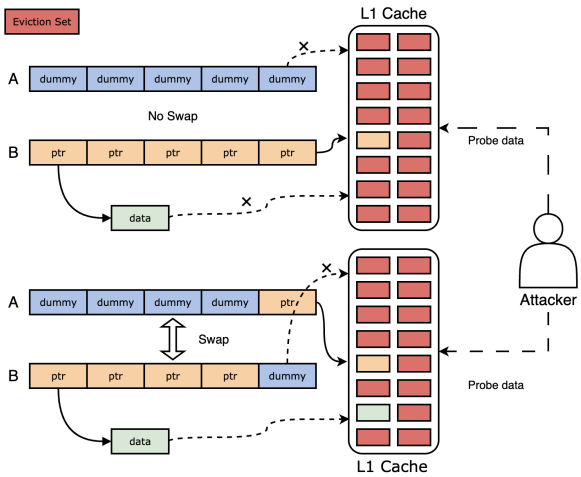



**Figure 7: CT-Swap Attack**

On input `a` with dummy values and `b` with pointers, the compound eviction set keeps evicting `a` and target data in caches. Restricted by the "do-not-scan hint", the `ptr` will never be dereferenced in cache if no swap is done (`secret=0`), and therefore, this indicates a L1 cache miss of target `data`. On the other hand, if a swap is done (`secret=1`), the attacker can observe a low latency in accessing `data`. Note that we can evict `a` and `data` from L1 simply by evicting them from L2, which is mentioned in Section 2. In short summary, the specific condition is that `ptr` only gets dereference by swapping from `b` to `a`.

## A.2 Attacking Go's RSA

The standard RSA encryption would have not been compromised, whose decryption method uses $c^d \bmod n$. However, since the power calculation is computation-consuming, in real-world implementation, the RSA decryption adapts Chinese Remainder Theorem (CRT) to accelerate the calculation. In Go's RSA-2048, in replacement of $c^d \bmod n$, it calculates $m_1 = (c \bmod p)^{D_p} \bmod p$ and $m_2 = (c \bmod q)^{D_q} \bmod q$, where $D_p = d \bmod (p-1)$ and $D_q = d \bmod (q-1)$. And finally $m = m_2 + h \times q$, where $h = (m_1 - m_2)(q^{-1} \bmod p) \bmod p$. Without lose of generosity, we assume that $p \geq q$ and we focus on the attack of $p$. The correctness of equation is not shown here. Note that $p, q$ are 1024 bits.

With this in hand, the vulnerable routine is that if the attacker construct a malicious ciphertext $c$ containing pointers since

- The first step in power is to calculate $t = c \bmod p$
- If $c < p$, then $t$ becomes $c$ and activates DMP.
- Otherwise, $t = c - k * p$ for some $k$ and $t$ has little chance to activate DMP, since $c$ is unlikely to contain pointer.

Next, the attacker can recover $p$ bit by bit starting from the most significant bit, with the input ciphertext for RSA decryption in Figure 8.

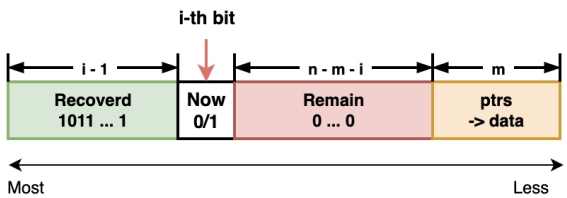



**Figure 8: Malicious input of ciphertext construction**

For the $i$-th bit of ciphertext $c$ with the most $i-1$ bits recovered, we put a random bit into it and feed $c$ to the decryption oracle (victim), and monitor whether the DMP activates. Since the most $i-1$ bits of $c$ are the same as $p$'s, the $i$-th bit immediately implies that $c < p$ or otherwise, with possibility of $1 - 1/(n-m-i) \approx 1$ for $i < n - m$, where $m$ is the total length of pointers. As the $i$ approximates, the probability of success keeps reducing, for which we stop at $i = n-m+16$ and the $p$ is not fully recovered. Nevertheless, the remaining part will be uncomplicated to be brute forced, just to check $gcd(n, p) = 1$. Thus, the attacker has a non-negligible chance to succeed with the hint from DMP.

## A.3 Identifying the Difficulties

Similar to other cache SCAs, the biggest difficulty lies in the generation of standard eviction set and compound eviction set. As discussed in Section 3, the DMP restricts the search region to be within the same 4GB page, which makes it hard to find the `ptr` and `data` that locates in 4GB page. Meanwhile, for a simulation of real-world attack, the victim's virtual address is transparent to the attacker process. So the other problem would be to identify the victim's address of array `a`. Another possible failure could be that `a` is not sit in page of `data`.

To resolve the problems mentioned above, the GoFetch researchers found that macOS allocates a identical virtual base address to the user process, which can be recovered by a unprivileged process. Once the attacker knows the boundary of virtual addresses, it will be able to generate the eviction set and test the effect.

However, another common obstacle in side-channel attack is also critical, that the environment noise is hard to avoid. For the environment noise detection, the attacker will concurrently measure the access time of eviction set and compare it with target access time.

# B WHY RANDOMIZING ADDRESS NOT WORKING

Due to the allocation policy in macOS/Linux, the virtual address in randomized only during boot time, and macOS allocates stably the pages upon the same base address even with ASLR enabled.

The randomization of address is supposed to be done on page allocation that will affects the ancient implementation of continuous memory management, i.e., the first-fit, next-fit, best-fit and worse-fit. The research to implement the randomized memory allocation is yet done to out best knowledge, and we consider it to be future work of the related research on defense of cache SCAs or other exploitation.

# Cardioactive: An Authentication System Based on ECG

Kaiwei Yi
Beijing University of Posts and Telecommunications
yalikiwi@bupt.edu.cn

Zhiyu Zhang
Southern University of Science and Technology
12110611@mail.sustech.edu.cn

Yutong Leng
Shanghai Jiao Tong University
enimdisy@sjtu.edu.cn

Yifu Hou
Fudan University
yfhou22@m.fudan.edu.cn

Kefei Wu
SiChuan University
fytwkf2jm@163.com

## Abstract

This paper presents an ECG-based identification system addressing the challenges of information and privacy security in the digital age. It explores the feasibility, advantages, and drawbacks of using ECG for identity authentication, and implements an ECG identification system using smart wearable devices. The study highlights the unique biological characteristics of ECG signals, which make them difficult to replicate and highly distinctive, demonstrating their potential for secure identity verification. The paper also compares ECG with EEG and EMG, discussing their respective features and practical applications in authentication systems. Experimental results confirm the feasibility and effectiveness of the ECG-based identification system.

***Keywords:*** Authentication, ECG, Biometrics

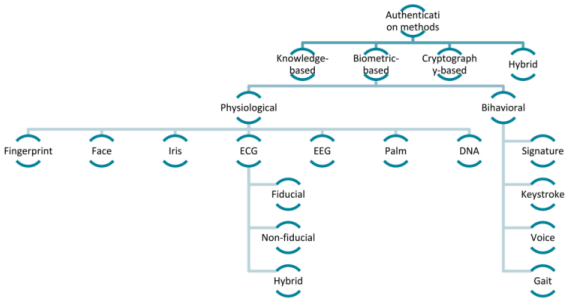



**Figure 1.** Authentication Methods

## 1 Introduction

### 1.1 Backgrounds

In the digital era, information and privacy security are increasingly challenged, with frequent privacy leaks and property theft linked to password system vulnerabilities. Popular password systems, including character and graphical passwords, can be broken through peeping, guessing, and brute-force attacks. Users must design complex, unique passwords for different systems, which are hard to remember and can be easily leaked through storage systems.

Biometric authentication, which includes biological, behavioral, and morphological biometrics, offers an alternative. Biological biometrics analyze DNA, behavioral biometrics analyze actions like gait, and morphological biometrics analyze unique physical traits like fingerprints and facial recognition. Despite improved security over traditional passwords, biometric methods also have issues. Fingerprints can be easily stolen, and facial recognition can be fooled by twins.

Under this status quo, it is especially important to develop a new biometric-based password authentication system with high security that cannot be stolen or imitated and does not need to be memorized. The three kinds of electrocardiogram (ECG), electromyogram (EMG) and electroencephalogram (EEG) belong to the bioelectrical signal measurement technology, which are used to monitor the electrical activities of the heart, muscle and brain, respectively, and are mainly used in medical diagnosis and physiological research at present.Compared with existing identification systems, these three types of bioelectric signals are (i) difficult to replicate, unlike traditional passwords that can be snooped or stolen, an individual's bioelectric signals are difficult to replicate. Even if they can be replicated theoretically, it is extremely difficult to implement them because precise signal acquisition equipment and complex analysis algorithms are required; (ii) absolute mutual dissimilarity, even if they are identical twins, the difference of these three types of bioelectric signals is very obvious, and there is no such phenomenon as misidentification of facial ID; (iii) indelibility, as long as the user's basic biological functions are normal, these three types of electrical signals always exist, and there is no such thing as fingerprints in fingerprint identification; (iv) indelible. (iii) indelibility, as long as the user's basic biological functions are normal, the three electrical signals are always present, and there is no phenomenon that fingerprints in fingerprint identification can be permanently destroyed under certain circumstances, resulting in failure to authenticate; (iv) user-friendliness, by using the bioelectrical signals for identification, the user does not need to memorize complex passwords or carry physical tokens. As long as the user is within the effective range of the monitoring device, authentication can be performed automatically, which improves the user experience. Therefore these three types of bioelectric

signals are more than suitable for the development of the new generation of biometric systems.

To this end, our group tried the identity authentication system based on electrocardiogram (ECG), electromyogram (EMG) and electroencephalogram (EEG) respectively, comprehensively researched and elaborated the feasibility and advantages and disadvantages of these three kinds of biometric features in the identity authentication on the basis of existing research, and practically realized the ECG identity identification system based on the acquisition of smart wearable devices according to the existing equipment conditions for teachers and students to For teachers and students to experience on-site.

### 1.2 EEG and EMG

As a signal originating from an individual organism, EEG has the characteristics of being difficult to forge and having a large individual variation. Taking advantage of these characteristics, many researches have begun to try to apply EEG to identification systems. The brain waves of the human brain are classified into alpha, beta, and theta waves according to their frequencies, in which alpha waves are generated when the human brain is in a stable state, and beta waves occur when there is strong mental or nervous activity, such as anxiety and nervousness.Gi-Chul Yang et al. proposed a personal authentication system based on machine learning and gave a detailed description in terms of the processing of the data features[1], the design of the classification algorithm of the machine learning, and the workflow of the system. The detailed description is made in terms of data feature processing, the design of machine learning classification algorithm and the workflow of the system。 Isao Nakanishi et al. designed an authentication system that utilizes artifacts in EEG and encodes them as passwords[2]. The system first collected EMG artifacts of eight facial movements to demonstrate the feasibility of accurately distinguishing between the tasks of "left eye blinking" and "right eye blinking", and constructed an identification system that utilizes EMG artifacts generated by facial muscles as passwords. It is important to note that although some of the current EEG devices do not require brain invasion and have good data acquisition, they are not portable and recyclable because most of the EEG caps require batteries of a certain size and the EEG electrodes become obsolete after a single acquisition. Electromyography (EMG) is a method of assessing and diagnosing muscle and nerve function by recording the electrical activity of muscles. The basic principle is that when a nerve impulse reaches a muscle, it triggers potential changes in the muscle fibers, and these changes are recorded by electrodes to form an EMG signal.

EMG signals are highly individual-specific due to differences in individual muscle structure, nerve conduction pathways, skin thickness, muscle fiber types, and other factors. Therefore, EMG signals can be used as a basis for identity authentication. By analyzing the features of EMG signals, such as spectral features of muscle electrical activity, Power Spectral Density (PSD), and time-frequency distribution, individual identification can be effectively performed.

Most EMG electrodes are single-use, costly and require special handling for reuse, increasing maintenance costs and complexity of use. Portable EMG devices, although commercially available, usually still require a certain volume to accommodate the battery and signal processing unit, and prolonged wear may lead to discomfort, limiting the convenience of daily use. In addition, portable EMG devices can usually only acquire a limited number of muscle group data simultaneously, limiting their application in complex motion and multi-task recognition.EMG signals are susceptible to external electromagnetic interference and environmental noise, especially in high-noise environments, where the signal quality may be severely affected and additional signal processing and filtering techniques are required to improve the reliability of the data.

### 1.3 ECG

ECG authentication involves capturing an individual's ECG biosignal and extracting unique features, which are then compared to a stored template to verify identity. Various feature extraction methods and classification algorithms were used for the study and publicly available databases were used for the study. ECG biometric systems generally include the following steps: (a)Data Collection: The ECG signals of the collected individuals were collected through a medical device. (b)Preprocessing: Using digital filters to remove noise from the signal, including low and high frequency noise. (c)Signal segmentation: identifies P-, QRS-, and T-waves in the ECG signal and segments the signal into manageable segments. (d)Feature extraction: key features are extracted from the segmented ECG signals, including reliable point and non-reliable point features. (e)Signal standardization: In order to ensure consistency of features, the signal needs to be aligned and standardized to eliminate the effects of physiological differences between individuals. (f)Classification training: the extracted features are classified and recognized using K Nearest Neighbor (KNN) classifier. (g)Authentication: the system collects the ECG signals of the individuals to be authenticated and repeats the preprocessing, signal segmentation, and feature extraction steps described above, and then uses the authentication algorithms to compare with the templates generated in the registration phase to decide whether to grant access.

It has been shown that the recognition accuracy of ECG signals after one month can reach 98.02% on average. In long-term applications [3], it is recommended to maintain the timeliness of the features by real-time updating the latest ECG data collected after each successful authentication.

In addition, the updating mechanism can employ adaptive learning algorithms, such as exponentially weighted moving average, for dynamic adjustment. In ECG analysis, the time intervals between consecutive R-waves are defined as RR intervals (RTTs), while the changes in RR intervals are analogous to RTTd. When the system captures a new ECG waveform, the features of the ECG signal can be updated by the following equation: $R_{new} = (1-\alpha) \cdot R_{old} + \alpha \cdot R_{new}$, where the parameter $\alpha$ can be optimized during the model training process to determine the best weights. In addition, the updating mechanism proposed in this study is not only applicable to electrocardiogram (ECG) signals, but can be equally extended to other biometric authentication modalities such as electromyogram (EMG) and electroencephalogram (EEG), thus enhancing the versatility and applicability of biometric identification systems.

This project uses the K-Nearest Neighbor algorithm, KNN is a non-parametric classification algorithm based on a distance metric, and its core idea is: given a sample to be classified, find the K closest samples by calculating its distance from each sample in the training set, and based on the categories (or values) of these K samples, decide the category of the sample to be classified (or the numerical value). The distance metric is the core of the KNN algorithm, and commonly used distance metrics include Euclidean distance, Manhattan distance, or Minkowski distance. When selecting the K value, the choice of K value has a large impact on the performance of the algorithm, and the optimal K value is usually selected by cross-validation method. The steps of the algorithm include calculating the distance, selecting the nearest neighbors, performing voting, and returning the results.

## 2 Design of the ECG subsystem

### 2.1 Data Acquisition

We utilized the ECG function of the Apple Watch device to capture the subject's ECG. Subjects were required to wear the Apple Watch on their wrist and place the fingers of their other hand on the crown and hold it for 30 seconds. Next, the generated ECG in pdf format is placed in the 'data' folder (the file name is 'name.pdf', where name is the subject's name). Restricted by system permissions, we could not obtain the original ECG data, so we obtained the ECG data through image processing and recognition algorithms. Firstly, the ECG part of the pdf file is cropped and saved in image format, and then the image is binarized and traversed to search the horizontal coordinates to find the corresponding ECG potential value.

### 2.2 Data Processing and Peak Detection

In ECG signals, baseline drift, electromyographic noise and power disturbances can interfere with signal processing. The acquisition of ECG receives industrial frequency interference (50Hz or 60Hz) from the equipment, as well as high frequency

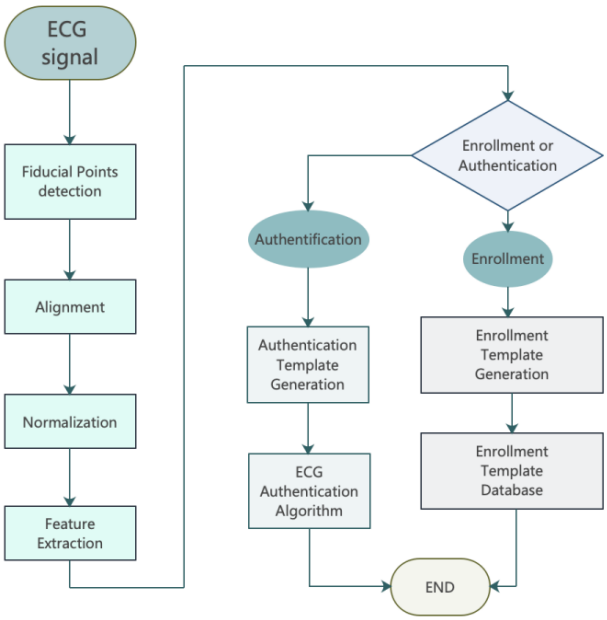



**Figure 2.** System Flow Chart

noise (greater than 100Hz) such as myofibrillation noise and power supply ripple introduced by the reference voltage of the sampling circuits, where 99% of the energy of the ECG is concentrated in the range of 0Hz-35Hz.[4]We utilize a low-pass filter that filters noise above 50 Hz to obtain ECG data with a high signal-to-noise ratio, while avoiding information loss as much as possible.

Baseline wander is a low-frequency disturbance in the electrocardiogram signal that shifts the baseline and affects the accuracy of the signal due to respiratory activity, poor and shifting electrode contacts, patient movement, and changes in skin impedance.To filter baseline fluctuations[5]. To filter baseline wander, we designed a baseline filter to filter some of the effects of baseline fluctuations on the ECG by means of a median filtering algorithm, which suppresses point fluctuations to some extent.

Due to individual differences, acquisition methods, etc., there are differences in the amplitude of the collected ECG potentials, and since this algorithm is based on the ECG peaks are with features for individual recognition, it may be worthwhile to use amplitude normalization to normalize the data. The normalization process helps to reduce the relative influence of these noises, thus improving the signal-to-noise ratio of the signal and facilitating the subsequent peak identification . For P, Q, R, S, T peak recognition, the consistency of the input signal is critical to the performance of the algorithm. The normalization process can ensure that the amplitude range of the input signal is consistent, thus improving the stability and robustness of the algorithm . The normalization process can enhance the contrast of the signal and make

the features such as P-wave, QRS-wave and T-wave more obvious, thus improving the accuracy of peak recognition. Especially for small amplitude signals, normalization can significantly enhance the visibility of these waveform features[6]. According to the relevant medical data, it is known that the normal human ECG and its characteristic peaks are shown in Figure 2. We (1) first identify all valid R-peaks by a certain

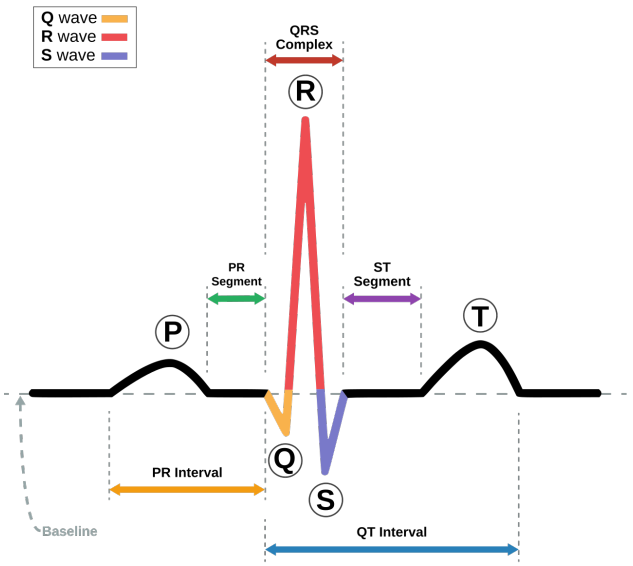



**Figure 3.** EEG with peaks marked

threshold and record the positions; (2) find a valid trough between every two R-peaks, denoted as S-peak; (3) find the highest valid peak between each S-peak and the next R-peak, denoted as T-peak; (4) find the highest valid peak between each R-peak and the previous T-peak, denoted as P-peak; (5) find a valid peak between each P peak and the corresponding R peak to find an effective trough, noted as Q peak.

After the above steps were completed, the data were cropped so that the various peaks were equal in number and conformed to the full P-Q-R-S-T cycle.

Based on the conclusions presented by Israel et al. for ECG analysis and recognition, linear normalization in the time domain aspect removes the effect of heart rate variability on temporal ECG features(e.g., R–P distance, R–T distance, R–LP distance,R–Q distance, R–S distance, R–TP distance, P width,T width, S–T distance, P–Q distance, P–T distance,LP–Q distance, and S–TP distance)[7].We used the R-wave as a basis for calculating the average heart rate period T of the subjects, and divided all the data by T to obtain the time-domain normalized data.

### 2.3 Model Training

First, the data file names were obtained from the database and the peak data (R, P, Q, S, and T waves) were read for each individual one by one. Next, features were extracted from the read peak data, and these features included the distance between the R wave and the other peaks, and the width of each peak. The feature data from all people were combined into one large data frame, and labeled columns were added to distinguish different individuals.

In order to prevent outliers from adversely affecting the training results of the model, the data is cleaned by a function that removes outliers, a function that retains data that are within the standard deviation interval. The processed data frames are divided into feature matrix and label vectors, and the data set is further divided into training and test sets using the *train_test_split*() function, which ensures that there is a consistent proportion of data in each category.

The model is trained using KNN classifier with the number of neighbors set to 3. The training set data is used to train the model. In the testing process, the peaks in the test data are first read and features are extracted. The processed test data also goes through the outlier removal step. Subsequently, the trained model predicts the test data to obtain the predicted label and the predicted probability.

Ultimately, the predicted frequency of each label is counted and the confidence of the prediction result is judged. If the frequency of a label exceeds Threshold, the person corresponding to that label is considered to be correctly recognized. This process ensures the validity of the data and the accuracy of the model, especially in terms of outlier handling and feature extraction, which enables the model to better recognize and classify ECG data.

For more details, visit our Our Project on Github.

## 3 Experiment

We invited five subjects (three males and two females; no heart disease, age 20 ± 2 years; recent good sleep, no intense exercise, mood in a stable state; no intake of drugs containing neuroleptics). These subjects underwent a 30-second ECG acquisition as training data.

Then, after a period of time (30 minutes), the ECG data of the five subjects were again collected in the same manner as test data, the model predictions were recorded, and key parameters such as true negative rate (TNR) and false positive rate (FPR) were calculated. The K-NN distance between the five subjects are shown in the table below:

With the preset threshold=0.75:

$\overline{TNR} = 10.07\%$ $\overline{FPR} = 2.74\%$

With the preset threshold=0.5:

$\overline{TNR} = 4.38\%$ $\overline{FPR} = 9.15\%$

Based on the experimental results, it can be concluded that our designed ECG system has a high accuracy rate for the

| Sub | YKW | ZZY | LYT | WKF | HYF |
|---|---|---|---|---|---|
| YKW | **0.970** | 0.037 | 0.037 | 0 | 0 |
| ZZY | 0.095 | **0.762** | 0 | 0 | 0.143 |
| LYT | 0.095 | 0 | **0.905** | 0 | 0 |
| WKF | 0 | 0.111 | 0.111 | **0.778** | 0 |
| HYF | 0 | 0.238 | 0 | 0 | **0.762** |

**Table 1.** Normalized K-NN Distance

identification of experimental samples. This is reflected in the more desirable K-NN distance, TNR, and FPR values.

## 4 Application

CardioActive Authentication can be performed on portable devices such as watches, headphones, etc. and can be used flexibly in everyday life. In frequently used low-security authentication systems, cardiac active authentication will directly compare the current result with the last result stored locally. We will use a watch as a detection device and make the following assumptions about an access control system: 1) When a user wears the watch, the watch will automatically perform cardiac active authentication and compare the data with a database to verify the user's identity. If the characteristic data is similar, the difference between the detection data and the authentication data is within a certain error range, and the authentication is successful; if there is no similar data, the watch will identify the user as an "unauthorized user" and alert the watch owner of the security risk through other electronic devices. At the end of a security cycle or after taking off the watch, the watch will update the data using the characteristic data detected during this period and perform the authentication test again. 2)Before the user arrives at the door, they use the watch to transfer the feature data to the door access system (via Bluetooth technology or QR code technology, etc.). 3)The door access system reads the previous data and the current data and compares them. If the difference between the two data is within the allowable error, the system returns a pass signal and updates the data; if the difference between the two data is significant, the system returns a fault signal. In cases where high security is required, due to the larger amount of data and the need to validate more characteristic values, the user's characteristic values are stored in a dedicated database. After the user uploads his/her identity information, the database compares and validates the user's various characteristic data with the detected characteristic values. For a large group of validated objects, a general challenge-response approach is used for validation.According to the proposal of Kushwahaha et al.[8] for challenge response based authentication application, a separate authentication behavior database is established and asynchronous dynamic challenge response method[9] is used to generate the question information at a time to avoid replay attacks. In important security scenarios,

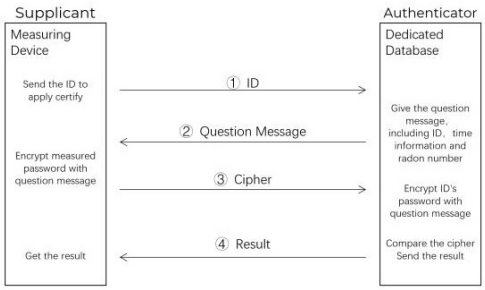



**Figure 4.** challenge-response Chart

for example, at custom checkpoints, the following scenarios may occur: 1) The server receives user identity information read by the device and sends question information 2) The device encrypts the data using the question information and uploads it to a dedicated cardiac activity authentication database 3) The database decrypts the data and compares it to the data in the database. 4) The result of this comparison is returned to the server to complete authentication In order to ensure data security and efficient encryption, data encryption uses stream cipher mode of operation. By utilizing the pseudo-randomness of stream ciphers, we can reduce potential security risks. Even if an attacker intercepts a fragment of a stream sequence, they cannot predict and infer the rest of the sequence. Since the communicating parties know the length of the feature memory, synchronization can be used to increase the efficiency of encryption and decryption and reduce the waiting time.

## 5 Outlook on the Evolution and Adoption

Blockchain provides a decentralized and immutable database system that securely stores and verifies transactions. Unlike public blockchains, private blockchains have restricted network access. Private blockchains require stringent admission and membership policies to establish trust among members. Participation in a private blockchain involves adhering to network rules, similar to the rigorous process of opening a bank account. The network can reject or revoke membership. In ECG authentication, Device Programmers (DPs) manage access keys to IMD devices through a blockchain. This private blockchain securely stores sensitive information in a tamper-proof, distributed manner, ensuring data security and reliability. DPs must provide necessary documentation to join the DP blockchain and act as nodes to provide verifiable, timestamped access to users' IMDs. This access management system benefits from the decentralized network's advantages, including peer-to-peer operations and cross-network data storage, eliminating the risks of centralized architectures and single points of failure, and providing robust protection against severe denial-of-service attacks.

In order to enhance the accuracy and robustness of biometric systems, some scholars in this research area have proposed a multimodal biometric fusion strategy. Specifically, electrocardiogram (ECG) signals are combined with other biometric metrics, such as heart rate variability (HRV), galvanic skin response, and photoplethysmographic volumetric pulsography (PPG), to achieve more comprehensive individual recognition. Related research works[10] have explored a multimodal authentication method that combines ECG and finger vein feature, investigated a biometric identification strategy that integrates ECG and electromyography (EMG) signals[11], and proposed a dual authentication mechanism that fuses ECG features with personal identification numbers (PINs)[12].

## 6 Conclusion

This thesis analyzes the shortcomings of the existing identity verification system and proposes a new type of identity recognition system based on ECG bioelectrical signals, analyzes its feasibility theoretically and separately, completes the complete system architecture of the system, and indicates the possible building method for the subsequent research. And under the existing equipment conditions, the complete construction of ECG identity recognition system with wearable devices as the platform, experimental testing and results demonstration were accomplished. In the experimental evaluation, the accuracy and efficiency of the ECG identification system are very good. According to this study, the identity recognition system based on bioelectric signals is fully feasible. In the future, the researcher's work will mainly focus on continuing to improve the accuracy and speed of identification using techniques such as machine learning, and considering how to integrate the identification system into portable devices that are as small as possible and consume as little energy as possible. Cardiac active certification is difficult to change, easy to collect, and can address some of the certification issues in the lives of people with disabilities and in extreme cases. According to the latest data from the World Health Organization, the number of people suffering from visual impairment and blindness is 2.2 billion globally. For the visually impaired, the widely used method of input verification is still the broadcast input method, where a machine will broadcast the input digits in real time for them to verify. This method is very inconvenient and insecure for the visually impaired. For the partially blind, who have some eye pathology, they prefer to wear sunglasses, but the traditional biometric-facial recognition has to be removed from their glasses, which is very disconcerting for them. Cardiac Active Authentication will solve this problem and provide a safer, privacy-friendly authentication method for the visually impaired.

In addition to being very effective at protecting privacy, the flexibility of the Heart Active Authentication device is one of the reasons we believe it will be successful in future applications. Some cameras are set to a fixed angle and fingerprint capture machines cannot be moved, but cardiac active authentication can successfully collect data through a watch or headset. Cardiac Active Authentication can help many people with physical disabilities and no longer need to rely on the help of others, leading to a more independent and convenient life.

## References


[1] Gi-Chul Yang. 2020. Next-generation personal authentication scheme based on EEG signal and deep learning. *Journal of Information Processing Systems* 16, 5 (2020), 1034–1047.

[2] Ying Zeng, Qunjian Wu, Kai Yang, Li Tong, Bin Yan, Jun Shu, and Dezhong Yao. 2018. EEG-based identity authentication framework using face rapid serial visual presentation with optimized channels. *Sensors* 19, 1 (2018), 6.

[3] 杨亿栋, 叶汪洋, 丁柯军, 张晨曦, 陶恒屹, and 陈萌. 2022. 心电信号在个人身份识别中的稳定性研究. *Computer Science and Application* 12 (2022), 1553.

[4] Philip Langley, Luigi Yuri Di Marco, Susan King, David Duncan, Costanzo Di Maria, Wenfeng Duan, Marjan Bojarnejad, Dingchang Zheng, John Allen, and Alan Murray. 2011. An algorithm for assessment of quality of ECGs acquired via mobile telephones. In *2011 Computing in Cardiology*. IEEE, 281–284.

[5] Gari D Clifford, Francisco Azuaje, Patrick McSharry, et al. 2006. *Advanced methods and tools for ECG data analysis*. Vol. 10. Artech house Boston.

[6] Narender P Reddy. 2002. Book review: biomedical signal analysis: a case-study approach, by Rangaraja M. Rangayyan. *Annals of Biomedical Engineering* 30 (2002), 983–983.

[7] Steven A Israel, John M Irvine, Andrew Cheng, Mark D Wiederhold, and Brenda K Wiederhold. 2005. ECG to identify individuals. *Pattern recognition* 38, 1 (2005), 133–142.

[8] Prashant Kushwaha, Harshita Sonkar, Fahiem Altaf, and Soumyadev Maity. 2021. A brief survey of challenge–response authentication mechanisms. *ICT Analysis and Applications: Proceedings of ICT4SD 2020, Volume 2* (2021), 573–581.

[9] Guifen Zhao, Ying Li, Liping Du, and Xin Zhao. 2015. Asynchronous challenge-response authentication solution based on smart card in cloud environment. In *2015 2nd International Conference on Information Science and Control Engineering*. IEEE, 156–159.

[10] Basma Abd El-Rahiem, Fathi E Abd El-Samie, and Mohamed Amin. 2022. Multimodal biometric authentication based on deep fusion of electrocardiogram (ECG) and finger vein. *Multimedia Systems* 28, 4 (2022), 1325–1337.

[11] Noureddine Belgacem, Régis Fournier, Amine Nait-Ali, and Fethi Bereksi-Reguig. 2015. A novel biometric authentication approach using ECG and EMG signals. *Journal of medical engineering & technology* 39, 4 (2015), 226–238.

[12] Sairul I Safie, John J Soraghan, and Lykourgos Petropoulakis. 2011. ECG based biometric for doubly secure authentication. In *2011 19th European signal processing conference*. IEEE, 2274–2278.

# Dynamic Password Assessment and Protection Based on Attacker Strategies

Zheng Changqian
Beijing Institute of Technology
changqian.zheng@gmail.com

Zhang Shuwei
Southern University of Science and Technology
12212912@mail.susetch.edu.cn

Hu Yijing
University of Nottingham, Ningbo China
yijinghu975@gmail.com

Zhao Zewen
Beijing University of Posts and Telecommunications
zhao16682299199@163.com

Zhang Yijun
South China University of Technology
zhangyijun1913@outlook.com

## ABSTRACT

This study pioneers an innovative approach to password security by examining attacker-generated passwords, an area often overlooked in current research. We introduce a system for extracting and analyzing malicious passwords to bolster defenses against online brute-force attacks, diverging from conventional methods focused on user password habits. Furthermore, our research emphasizes the importance of understanding attackers password construction strategies to tailor defenses effectively. By analyzing patterns and semantic components of passwords used by attackers, we enhance the robustness of password security measures.

Additionally, our approach includes a dual methodology for capturing and storing password characteristics: firstly, storing user-specific patterns during registration or modification, which are independent of attackers (e.g., password patterns); and secondly, dynamically updating and analyzing the current attacker database during each login, focusing on attacker-related characteristics such as frequency of use of each word in attacker password attempts. This enables proactive user warnings based on current attacker trends.

This comprehensive approach not only addresses evolving cyber threats but also aligns defenses with dynamic cybersecurity landscapes. Moving forward, our focus will be on refining theoretical frameworks for password strength evaluation and exploring advanced pattern and word extraction techniques. The scarcity of research on attacker password patterns underscores the need for further exploration in this critical cybersecurity domain.

In conclusion, this research contributes novel insights into password security by shifting focus to attacker perspectives, paving the way for more effective defense strategies against malicious activities targeting password systems.

## 1. INTRODUCTION

In the realm of password security, a significant gap exists between the ideal standards for robust passwords and the practical preferences of users, who often prioritize ease of memorization over security strength. Current guidelines advocate for passwords of at least 11 characters in length, comprising a mix of uppercase letters, lowercase letters, numbers, and special characters, recognized for their resilience against attacks[7].

However, due to the challenges users face in remembering such complex passwords [7], they frequently opt for word-based passwords, i.e. passwords constructed from a list of words or meaningful strings, which, despite being easier to recall, are more vulnerable to exploitation by attackers [7]. These passwords which are typically derived from common words or phrases, lack the security strength afforded by more complex and randomly generated combinations.

This study focuses on the development of a password strength assessment model tailored specifically for word-based passwords, leveraging dynamically collected malicious password data. In addition to storing user-specific patterns during registration or modification, which are independent of attackers (e.g., password patterns), we also dynamically update and analyze the current attacker database during each login. This dual methodology aims to provide proactive user warnings based on current attacker trends.

By establishing a phishing site designed to capture passwords attempted by malicious actors, our research aims to enhance understanding of password vulnerabilities and improve overall password security measures.

This paper addresses the challenge of evaluating the security of word-based passwords, despite their prevalent usage and susceptibility to attacks. Users often favor word-based passwords due to their ease of recall, yet these passwords lack the

robustness provided by more complex combinations. The primary focus lies in developing effective methods to assess and improve the security of such passwords amidst evolving cyber threats.

Our study makes several key contributions to the field of password security:

- Basic characteristics of the attackers' password: We identify and analyze fundamental characteristics and distribution patterns of attacker passwords by constructing a simulated server to collect malicious password data.
- Words extraction and Patterns: We propose mechanisms for extracting words from passwords and analyzing patterns, which enhance the detection of weak password structures.
- Password strength evaluation: We introduce a novel framework for evaluating password security based on empirical analysis of attacker methods, providing insights into the vulnerability of user passwords and guiding improvements in password security practices.
- Password management system design: We integrated theoretical research with practical application and designed a system that enables dynamic evaluation of user password security and defense against online password attacks

This sets the stage for understanding the critical need to address weaknesses in word-based passwords and outlines our approach to enhancing password security through analysis and proactive defense strategies.

# 2. RELATED WORK

## 2.1 Literature review

Current strategies aimed at safeguarding user passwords primarily employ a user-centric approach. This approach involves analyzing password compositions, identifying frequently used words or phrases, and implementing targeted protections based on these findings [7]. While effective to a degree, this strategy requires ongoing adaptation to counter evolving attack methodologies and user behaviors.

From an offensive standpoint, comprehensive statistical analyses using tools like CAUDIT shed light on significant insights into password security. CAUDIT is an instance of meticulously logged 11 billion SSH brute-force attacks directed at NCSA's systems from February 2017 to November 2019, revealing enduring and methodical efforts by malicious actors [9]. Attackers employed sophisticated strategies, including the use of stolen SSH keys and evasion techniques like randomized client versions, underscoring their adaptability and resourcefulness.These findings underscore the inherent vulnerabilities of multi-user systems, where even a single weak password can compromise entire infrastructures. Attackers often exploit generic password habits derived from publicly accessible datasets, perpetuating risks despite advancements in computational capabilities and password policies [9]. Even though Owen et al have done some research [6] on brute force passwords based on fake ssh collection, however, the research protection based on attacker's passwords strategies is still nearly in a blank.

These findings underscore the inherent vulnerabilities of multi-user systems, where even a single weak password can compromise entire infrastructures. Attackers often exploit generic password habits derived from publicly accessible datasets, perpetuating risks despite advancements in computational capabilities and password policies.

To mitigate these threats effectively, proactive security measures tailored to address evolving attack vectors are essential. Furthermore, despite advancements in hacking techniques, dictionary-based attacks remain prevalent. These attacks are often enhanced with password mangling rules that mimic common user behaviors, such as character substitutions and alterations in capitalization and numerical placement within passwords [1]. Consequently, reinforcing strong password policies and educating users on secure password practices are crucial in mitigating the effectiveness of such attacks.Although strong, complex passwords provide substantial security benefits, their widespread adoption is impeded by usability concerns. Therefore, it is imperative and pragmatic for us to examine password security from the perspective of attackers and their methods.

## 2.2 Comparison with Existing Methods

Analyzing password security from the attacker's perspective offers several advantages compared to traditional user-centric assessments:

- By focusing on how attackers construct passwords, defenses can be tailored specifically to thwart known attack methods. This approach directly addresses the techniques and tools malicious actors use, enhancing the likelihood of successful mitigation.
- Analyzing the user's password composition for protection usually involves privacy issues, and obtaining the data set requires certain permissions[2]. The existing research is basically based on the accidental leaked password library for analysis, and there is a problem that the data may be outdated. Attackers' techniques evolve rapidly, necessitating up-to-date data and analysis methods to effectively counter emerging threats.
- By adopting this perspective, security professionals can implement proactive measures that anticipate and mitigate potential vulnerabilities before they are exploited. This approach not only enhances the robustness of password security but also aligns defenses with the dynamic landscape of cyber security threats.

# 3. HYPOTHESIS

Our research proposes several hypotheses for further exploration and optimization:

- Exclusion of Weak Passwords: Weak passwords are systematically excluded from secure systems due to their vulnerability to attacks. This hypothesis underscores the importance of stringent password policies in mitigating security risks.

- Challenges with Strong Password Adoption: Despite their enhanced security, strong passwords face adoption challenges due to user memorization difficulties. Consequently, users often resort to word-based passwords, which are easier to remember but more vulnerable to attacks.
- Exploitation of Word-based Password Vulnerabilities: Attackers leverage knowledge of user habits to exploit vulnerabilities in word-based passwords. They employ sophisticated brute-force techniques tailored to mimic common user behaviors, thereby compromising security.
- Scale and Persistence of Attackers: The volume and persistence of password attempts by attackers exceed those typical of legitimate user behavior. This observation highlights the scale and intensity of malicious activities targeting password systems. This can be proved by our statistic based on collected malicious passwords.

Based on these assumptions, our research endeavors to explore and optimize password security strategies, aiming to enhance protection against evolving cyber threats from the perspective of investigating attackers.

# 4. METHODOLOGY

## 4.1 The basic research for the malicious passwords

Because the lack of the search for analyzing the attackers password patterns. To proof our assumption, we did a basic research for malicious passwords.

we set a open-sourced fake ssh project [3] to collected malicious passwords and use some naive methods to sort the passwords into three parts: Weak passwords, word-based passwords and the strong passwords. Here, we define these three types of passwords as follows:

- Weak passwords: The password that is too simple that is too easy to find, which contains some week acknowledged passwords like "123456" or some information about the information of website, server and users like "root" , "linux"
- Word-based passwords: The password here contains the words that is used in passwords, include the language words and the other strings that frequently used in passwords. The words can be extracted and recorded its difference by these type of password in limited time for calculation like "P@ssword" , "root123"
- strong passwords : The password that is with high randomness of characters or can not extract the words in a limited time.

Based on our classify of the passwords, we acccordingly analysis the passwords collected. From 109,904 instances, weak passwords constitute approximately 60% of the dataset, while word-based passwords account for 39%, and strong passwords represent about 1%.

## 4.2 Extract the pattern of the password

Recently there have been a few researches on the structure and semantic analysis of password [10] [4] [8]. However, these methods still have some limitations such as Lack of consideration for common special characters in passwords or insufficient segmentation algorithm.

In our research, we apply dynamic programming method and NLP techniques to extract semantic pattern of common passwords. In the following part, we will discuss the three critical parts of our approach: pattern definition, corpus generation and algorithm workflow.

### *4.2.1 Pattern definition*

To record and analyze the structure of a password, we introduce the concept of pattern: "Pattern" is a succinct symbol sequence for expressing the password structure. In the following part of our paper, we use these characters to record the passwords structure:

- **W** : Represents the part consists of the words or the other word like phrases that frequently found in passwords , like "password" , "linux" , "admin" , "123456"
- **A** : Represents the part consists of the characters in alphabet , like "a" , "Z" and so on.
- **N** : Represents the part consists of the characters in numbers like "1" , "9" and so on.
- **S**: Represents the part consists of the characters in special words like "@" , "!" and so on.

**Example** 4.1.

- ***admin@123456xyz*** $\rightarrow W_5S_1N_6A_3$

In our definition, all meaningful strings are classified as **W**, and the meaningless characters are categorized in alphabet, numbers, special words. For security consideration, we have not classified word any further, for example POS. More details will be discussed in 5.1.

### *4.2.2 Corpus generation*

Our algorithm relies on two main corpus sources: existing English corpora and special strings derived from public password datasets. For English corpora, we utilize data from the Google Ngram project, obtained from [4], which includes frequency information for the most common 10,000 1-grams, 5,000 2-grams, 3,000 3-grams, 1,000 4-grams, and 1,000 5-grams. Additionally, we augment this with lists of names, surnames, countries, and cities to encompass potential password constructs.

Regarding special strings found in passwords, a relevant area of research involves "word detection" or "word discovery," extensively explored in Chinese text processing. Despite similarities in the absence of word spaces in both passwords and Chinese text, existing algorithms for word detection in Chinese texts often consider grammatical characteristics, which may not directly translate to password generation rules. Due

to these reasons, we employ a straightforward algorithm that counts substring occurrences across all passwords, with subsequent noise reduction.

These corpora serve as references for exact or approximate matches and aid in probability calculations for different segmentations. Given the inherent uncertainties in our string detection algorithm, priority is given to existing corpora, with the generated corpus utilized when no direct matches are found.

#### *4.2.3 Algorithm workflow*

The core process of our pattern extraction involves segmentation, where the goal is to identify the optimal segmentation that accurately reflects the password's structure. Initially, common strategies treated English letters and other characters separately, viewing numbers and special symbols as potential "gaps" [6]. However, this approach may fall short due to users' tendencies to manipulate passwords through character insertion, deletion, and substitution, often including special characters within words.

To address this issue, our algorithm explores all potential segments and employs a heuristic function to assess how closely a substring resembles a word. Prior approaches such as [3][6] utilized a heuristic function that assigns a length value if a segment matches any word in the corpus; alternatively, our algorithm employs a quadratic function for heuristic calculations. This approach transforms the problem into finding the segmentation with the highest score, where the score represents the sum of heuristic values for all sub-segments, calculated using dynamic programming techniques.

Given that multiple segmentations may achieve the maximum score, and that the segmentation with the highest score may not always be the most suitable due to heuristic function imperfections, we maintain records of segmentations with the top k highest scores. Subsequently, we select the segmentation with the highest probability based on word frequency information within the corpus. In our experiments, we typically set $k = 10$.

To estimate the probability of a segmentation candidate, we utilize the frequency of both single words and n-grams (2-grams to 5-grams) in our calculations. Specifically, we estimate the probability using segments tagged as **W** and use this word list's probability to represent the segmentation's overall probability. For words without a direct match in the corpus (identified via fuzzy matching), we also consider the probability of edits applied to them.

Ultimately, the segmentation with the highest score and probability is selected as the password's pattern.

### 4.3 Evaluating user's password security

Building upon the methodologies described earlier, we extract patterns and associated words from user passwords, enabling us to gauge the likelihood of attackers using these passwords based on pattern and word frequency analysis. The specific calculation is in Appendix B. By calculating the probability, we can make the estimation of the password strength : When the probability in a scale of time is bigger than the preset threshold value $\theta$ , then we can judge it that this password is not secure recently.

## 5. SYSTEM DESIGN

To bridge theoretical research with practical applications, we develop a basic framework for a dynamic password protection system.

### 5.1 Design principle

Traditional password management system displays demonstrate good resilience against offline password cracks, i.e. the resilience to password file leakage and corresponding offline hash cracking. However, such system have poor performance against online user-targeted attacks, for they cannot identify whether a user is under attack and how closely the password is to be cracked.

On the other hand, if we more or less record some characteristics of users' passwords, it is possible that we can detect such targeted attacks by conducting similar analysis on the attackers' passwords and performing characteristic comparisons and hence evaluate the strength of user password. However, while recording more information can make strength analysis and attack detection more accurate, attackers can more easily engage in offline cracking in the case of file leakage. Thus, there is a trade-off between defensive capabilities against online attacks and offline attacks.

### 5.2 Goal and framework

Our system is designed to defend against two types of attacks: indiscriminate weak password brute force attacks and targeted password brute force attacks.

Here are explanations for the two concepts:

- Indiscriminate attack: "Indiscriminate attack" refers to a method where Attackers systematically attempt to gain unauthorized access to multiple accounts or systems by trying common weak passwords. This method exploits the widespread use of easily guessable passwords, leveraging automated tools to maximize the chances of breaching a large number of accounts.
- Targeted password brute force attack: "Targeted attack" refers to a method where attackers gather specific information about users' passwords, such as patterns or other details, to generate a large number of password guesses. This approach focuses on exploiting known patterns and user-specific information to crack passwords.

The following diagram depicts the core framework of our system.

### 5.3 Main components

Our system consists of the following components:

- Server handling login requests
- Database of user hashed passwords

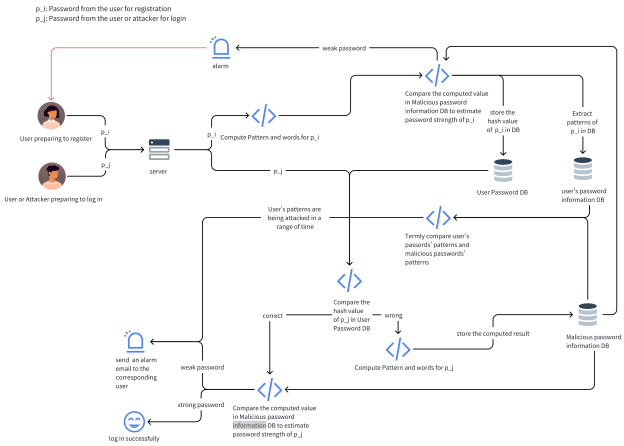



**Figure 1: Overall framework**

- Database of user password characteristics
- Database of attacker password characteristics
- Scripts for automated characteristic extraction and password strength analysis.
- Scripts for regular password strength checking and user alerting.

Here the we use "pattern" and relevant probability to represent password characteristic, which allows for dynamic password analysis without recording excessive information based on the principle mentioned above.

Since there might be demand for maintaining a dynamic weak password dictionary for common weak password detection, we may also need to set up a phishing server to collect weak password data.

## 5.4 Data storage and maintenance

For database of users' hashed password, it only stores userID and their corresponding hashed password values for the purpose of final correctness verification.

For database of users' password characteristic, it stores userID, password pattern, cracked probability estimation (as stated in 4.3). As shown in 4.3, the probability calculation involves pattern frequency and word frequencies, while the later is not store in the database for security consideration. To deal with the problem, we adopt a semi-dynamic method for probability calculation: each time the user register / login, we update the probability based on latest attacker statistics instead of explicitly store the speific words contained in user's password. Under the mechanism, users who log in more frequently can obtain a more accurate assessment of password strength.

For databases of attacker password characteristics, one stores attackers' passwords pattern and corresponding frequencies. Another stores specific words used in attackers' passwords and corresponding frequencies. Since we only consider the attacking trend over a recent period of time, for example, a month, we should also maintain a database/file to record each attack attempts in chronological order to remove statistics beyond the time range.

## 5.5 Defence mechanism

### *5.5.1 Defence of indiscriminate attack*

To defend the indiscriminate attack , we design a dynamically password judging mechanism focusing on two key aspects: identifying and storing potentially malicious passwords, and evaluating the security of user-submitted passwords through pattern and word analysis.

The system dynamically tracks and records potentially malicious login attempts by analyzing failed passwords. These passwords are broken down into their constituent patterns and words, and the resulting data is stored in a malicious password information database.

To enhance password security on it, Our system employs the following evaluation and alert mechanisms during registration and post-login scenarios.

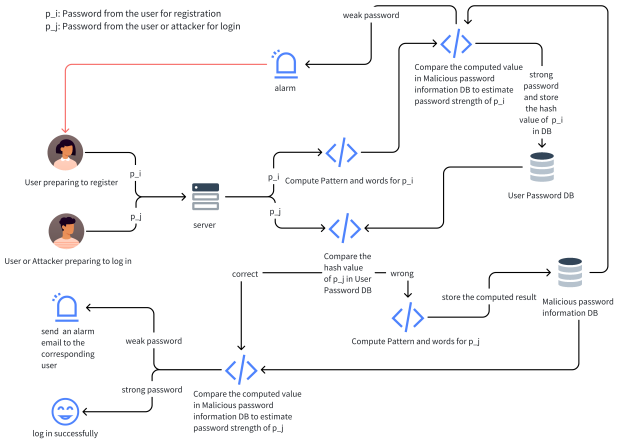



**Figure 2: defence on registration and post-login scenarios**

- Registration Password Evaluation: Upon user submission of a password during registration, the system analyzes its patterns and words. This analysis includes comparison against data stored in the malicious password information database. If the submitted password meets security criteria, it is accepted, and its hashed value is stored in the user password database.
- Post-Login Password Evaluation: For successfully logged-in users, the system continuously evaluates their passwords by comparing patterns and words against the malicious password information database. If a password is deemed insecure, the system alerts the user via email, recommending a password change to enhance security.

### *5.5.2 Defence of targeted attack based on patterns*

To defend targeted attack of user's password, we designed a termly password intensity checking system based on the password intensity evaluating system above. We record user's

passwords' patterns and the probability termly compare and modify the probability of user's passwords.

To enhance this type of attacks, Our system implements the following security evaluation and alert mechanisms throughout the entire lifecycle.

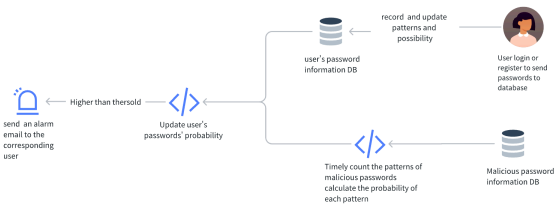



**Figure 3: defence throughout the entire lifecycle**

- Patterns' recording: When a password is recognized as malicious passwords. They are extracted into words and patterns stored in the malicious passwords information database. We termly statistic the patterns and calculate probability of each pattern's occurrence.When a targeted attack on patterns occurred, a large number of passwords with the same pattern will flow into the malicious passwords , which will make that pattern's probability rapidly ascend.

- Probability re-estimation based on changed patterns: In our probability estimation system, the probability calculated is highly relevant to the patterns' probability used by attackers. So when pattern's occurrence changed, we can also update user's password's intensity. If a password's probability of being guessed is highly raised and up to the threshold, then it will be recognized as being targeted attacked and the system will give users alarm.

# 6. CONCLUSION AND REFLECTION

## 6.1 Research conclusion

Our research introduces a pioneering approach to password security by analyzing attacker-generated passwords,which is an under-explored area in current literature. We have developed a password information extraction system to evaluate the strength of malicious passwords, aiming to enhance defense against online brute-force attacks. This method diverges from traditional approaches that primarily focus on understanding user password construction habits.

## 6.2 Limitations of methods

The evaluation of user password strength in our study is overly simplistic. We rely on basic frequency estimation probability methods to assess the usage of words and patterns in passwords. Additionally, our password calculation approach merely multiplies the probabilities of pattern, word, and character insertions, assuming low correlation among these elements. However, this method is vulnerable to manipulation by attackers who can distort statistical insights by flooding the system with large volumes of meaningless passwords.

Furthermore, our current methodology lacks extensive experimentation. While we have been conducting a demonstration to estimate password strength using collected malicious passwords, our system is designed as a dynamic password evaluation system. Future iterations of our research should involve building a more sophisticated processing system and conducting longer-term experiments to assess its effectiveness and availability over time.

## 6.3 Possible Improvements

To enhance the accuracy of weak password assessments, we propose integrating additional parameters of password formation into our probability calculations. Techniques such as deep learning could significantly improve assessment precision. Furthermore, exploring recommender system techniques could refine our approach by analyzing attacker password preferences. Additionally, employing Markov models [5] offers a potential avenue to enhance accuracy in predicting attacker password strategies.

## 6.4 Future exploration

Our method has practical applications for websites, web servers, and systems relying on password-based authentication. By providing users with real-time evaluations of password strength, administrators can actively monitor and mitigate brute-force attacks. Analyzing attacker passwords also provides insights into ongoing threats to password security systems, empowering informed decision-making by administrators.

Future efforts will focus on refining our theoretical framework and conducting experiments to validate the effectiveness of our methods. We aim to transition our system from a static demo to one capable of dynamic password assessment in real network environments. Additionally, we will explore advanced pattern and word extraction techniques to further enhance password strength assessment. The current lack of research on attacker password composition and distribution underscores the need for continued exploration in this critical cybersecurity domain.

# 7. REFERENCES


[1] A novel dictionary generation methodology for contextual-based password cracking. Accessed: 2024-07-18.

[2] L. Arezina. Password statistics for 2020. `https://dataprot.net/statistics/password-statistics/`, Nov 2019. Accessed: 2024-07-18.

[3] fffaraz. fakessh. `https://github.com/fffaraz/fakessh`, 2024.

[4] M. Jakobsson, M. Jakobsson, and M. Dhiman. The benefits of understanding passwords. *Mobile Authentication: Problems and Solutions*, pages 5–24, 2013.

[5] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, pages 689–704, 2014.

[6] J. Owens and J. Matthews. A study of passwords and methods used in brute-force ssh attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, page 8, 2008.

[7] R. B. Varne and R. V. Mane. Captcha: A robust approach to resist online password guessing attacks. In *2014 International Conference on Advances in Communication and Computing Technologies (ICACACT 2014)*, pages 1–6, Mumbai, India, 2014.

[8] R. Veras, C. Collins, and J. Thorpe. On semantic patterns of passwords and their security impact. In *NDSS*. Citeseer, 2014.

[9] Y. W. Mining threat intelligence from billion-scale ssh brute-force attacks. Accessed: 2024-07-18.

[10] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE symposium on security and privacy*, pages 391–405. IEEE, 2009.

# APPENDIX

## A. SEGMENTATION ALGORITHM

**function** MAX-K-SEGMENTATION($password, k$)
  $L \leftarrow \text{len}(password)$
  $dp[0].\text{push}(0)$ ▷ $dp[u]$ is a min-heap for $0 \le u \le L$
  **for** $i \in 1...L$ **do**
    **for** $j \in 0...i-1$ **do**
      $value \leftarrow \text{getValue}(password[j+1:i])$
      **for** $sum \in dp[j]$ **do**
        $dp[i].\text{push}(sum + value)$
      **end for**
    **end for**
    **while** $dp[i].\text{size}() > k$ **do**
      $dp[i].\text{pop}()$
    **end while**
  **end for**
**end function**

## B. CALCULATING OF PROBABILTY

### B.1 Calculate the pattern space

Here we define the addition space D, which means for the same pattern, we can make different passwords via inserting different characters into non-word parts. The number of different insertion of characters can be represented by $D$ , which can be calculated by the number of each pattern's parts apart from the word part: alphabet insertion part$n_{a_i}$, number insertion$n_{n_i}$ , special character insertion part $n_{s_i}$ . Each digit, depending on the data type, will have a different number of insertion possibilities : $k_{a_i}$ , $k_{n_i}$ , $k_{s_i}$ . Then we can calculate the pattern space:

$$S = \prod_{a_i} n_{a_i}^{k_{a_i}} \prod_{n_i} n_{n_i}^{k_{n_i}} \prod_{s_i} n_{s_i}^{k_{s_i}} \tag{1}$$

Then the probability of inserting the correspond characters is:

$$p_1 = \frac{1}{S} = \frac{1}{\prod_{a_i} n_{a_i}^{k_{a_i}} \prod_{n_i} n_{n_i}^{k_{n_i}} \prod_{s_i} n_{s_i}^{k_{s_i}}} \tag{2}$$

### B.2 Calculate the probability of insert corresponding words

For the password, apart from the word insertion, we need to evaluate the probability of using correspond word's. This is evaluated by the corresponding elements $n_{w_i}$ occurrence in words with same length $n_{l_i}$ , then the words probability can be calculate as:

$$p_2 = \prod_{w_i} p_{w_i} = \prod_{w_i} \frac{n_{w_i}}{n_{l_i}} \tag{3}$$

### B.3 Calculate the probability of this password appearing within the expected time frame.

After evaluating the password's editing space and the word's probability , we can evaluate the probability of a user's password used by attackers:

$$p = p_1 p_2 = \frac{\prod_{w_i} \frac{n_{w_i}}{n_{l_i}}}{\prod_{a_i} n_{a_i}^{k_{a_i}} \prod_{n_i} n_{n_i}^{k_{n_i}} \prod_{s_i} n_{s_i}^{k_{s_i}}} \tag{4}$$

Then to calculate probability of the passwords used maliciously within the expected time frame. We also need to calculate number of the malicious passwords sent to server $N$ in expected time $T$ , which can be predicted by the number of malicious passwords $n_0$ in collected time $t_0$ :

$$N = T\frac{n_0}{t_0} \tag{5}$$

Then the expected probability in time range t for a user's password used by attackers is:

$$P(t = T) = 1 - (1 - \frac{\prod_{w_i} \frac{n_{w_i}}{n_{l_i}}}{\prod_{a_i} n_{a_i}^{k_{a_i}} \prod_{n_i} n_{n_i}^{k_{n_i}} \prod_{s_i} n_{s_i}^{k_{s_i}}})^{T\frac{n_0}{t_0}} \tag{6}$$