Fall 10-10-2022

Software Protection and Secure Authentication for Autonomous Vehicular Cloud Computing

Muhammad Hataba

SOFTWARE PROTECTION AND SECURE AUTHENTICATION FOR AUTONOMOUS VEHICULAR CLOUD COMPUTING

by

Muhammad Hataba

A Dissertation
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Computing Sciences and Computer Engineering
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved by:

Dr. Ahmed Sherif , Committee Chair
Prof. Ras B. Pandey
Prof. Dia Ali
Prof. Chaoyang Zhang
Dr. Amer Dawoud

October 2022

ABSTRACT

Artificial Intelligence (AI) is changing every technology we deal with. Autonomy has been a sought-after goal in vehicles, and now more than ever we are very close to that goal. Vehicles before were dumb mechanical devices, now they are becoming smart, computerized, and connected coined as Autonomous Vehicles (AVs). Moreover, researchers found a way to make more use of these enormous capabilities and introduced Autonomous Vehicles Cloud Computing (AVCC). In these platforms, vehicles can lend their unused resources and their sensory data to join AVCC.

In this dissertation, we investigate security and privacy issues in AVCC. As background, we built our vision of a layer-based approach to thoroughly study state-of-the-art literature in the realm of AVs. Particularly, we examined some cyber-attacks and compared their promising mitigation strategies from our perspective. Then, we focused on two security issues involving AVCC: software protection and authentication.

For the first problem, our concern is protecting client's programs executed on remote AVCC resources. Such a usage scenario is susceptible to information leakage and reverse-engineering. Hence, we proposed compiler-based obfuscation techniques. What distinguishes our technique, is that it's generic and software-based and utilizes the intermediate representation, hence, it's platform agnostic, hardware independent and supports different high level programming languages. Our results demonstrate that the control-flow of obfuscated code versions are more complicated making it unintelligible for timing side-channels.

For the second problem, we focus on protecting AVCC from unauthorized access or intrusions, which may cause misuse or service disruptions. Therefore, we propose a strong privacy-aware authentication technique for users accessing AVCC services or vehicle sharing their resources with the AVCC. Our technique modifies robust function encryption, which protects stakeholder's confidentiality and withstands linkability and "known-ciphertexts" attacks. Thus, we utilize an authentication server to search and match encrypted data by performing dot product operations. Additionally, we developed another lightweight

technique, based on KNN algorithm, to authenticate vehicles at computationally limited charging stations using its owner's encrypted iris data. Our security and privacy analysis proved that our schemes achieved privacy-preservation goals. Our experimental results showed that our schemes have reasonable computation and communications overheads and efficiently scalable.

ACKNOWLEDGMENTS

This is to thank all of those who have assisted me in this effort. I am forever indebted to my advisor, Dr. Ahmed Sherif, who is the source of all wisdom in this wordly life. I am and remain in his awesome, brilliant shadow. I am grateful to Prof. Dia Ali, whom I look up to and from whom I learned a lot. And I would like to thank the rest of my dissertation committee; Prof. Ras Pandey, Prof. Chaoyang Zhang and Dr. Amer Dawoud for their helpful guidance and valuable attention.

I am also thankful to the National Telecommunications Institute (N.T.I) of Egypt, for allowing me the time to pursue my post graduate studies.

Last but not least, great appreciation to my parents for their always sincere care, utmost devotion and endless encouragement during my life. Also, I would like to thank my brother and sister for their advice and help during all of my studies. Finally, I wouldn't forget all of my friends who stood beside during hours of stress and hardship. Honestly, I wouldn't have been able to do anything without the love and support from everyone around me.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGMENTS iv

LIST OF ILLUSTRATIONS vii

LIST OF TABLES ix

LIST OF ABBREVIATIONS x

1 INTRODUCTION **1**

2 SECURITY AND PRIVACY ISSUES IN AUTONOMOUS VEHICLES: A LAYER-BASED SURVEY **6**
2.1 Introduction 6
2.2 Application Layer Security 6
2.3 Operating System Level 27
2.4 Network Level 28
2.5 Physical Level 32
2.6 Related Work 34
2.7 Summary 35

3 A PROPOSED SOFTWARE PROTECTION MECHANISM FOR AUTONOMOUS VEHICULAR CLOUD COMPUTING **37**
3.1 Introduction 37
3.2 Background Information and Related Work 38
3.3 System Description 44
3.4 Proposed Technique 45
3.5 Experiments and Results 46
3.6 Summary 49

4 ENHANCED OBFUSCATION FOR SOFTWARE PROTECTION IN AUTONOMOUS VEHICULAR CLOUD PLATFORMS **50**
4.1 Introduction 50
4.2 System Description 51
4.3 Proposed Technique 51
4.4 Experiments and Results 53
4.5 Related Work 58

4.6 Summary 59

5 A PRIVACY-AWARE SYSTEM FOR AUTHENTICATION IN AUTONOMOUS VEHICLES CLOUD COMPUTING 61
5.1 Introduction 61
5.2 System Models 62
5.3 Proposed Scheme 63
5.4 Privacy and Security Analysis 67
5.5 Experiments and Discussions 67
5.6 Related Work 71
5.7 Summary 72

6 PRIVACY-PRESERVING BIOMETRIC-BASED AUTHENTICATION SCHEME FOR ELECTRIC VEHICLES CHARGING SYSTEM 74
6.1 Introduction 74
6.2 System Models 75
6.3 Proposed Scheme 76
6.4 Privacy Analysis 79
6.5 Experiments and Performance Evaluation 80
6.6 Related Work 83
6.7 Summary 84

7 CONCLUSION AND FUTURE WORK 85

BIBLIOGRAPHY 86

LIST OF ILLUSTRATIONS

Figure

1.1 Autonomous vehicle cloud computing system. 2

2.1 The expanded layered structure of autonomous vehicles system. 7
2.2 A typical traffic management system would control intersections, traffic lights and ramp meters among other things. 9
2.3 An autonomous vehicles platoon and the various sensors they use. 11
2.4 Architecture of an AV carpooling system. 15
2.5 An example of an automated parking system for AVs. 19
2.6 Architecture of IoV system. 23

3.1 The network model of an AVCC platform. 44
3.2 Obfuscated code versions of the Bubble Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 47
3.3 Obfuscated code versions of the Quick Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 47
3.4 Obfuscated code versions of the RealMM program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 48

4.1 Flow chart explaining the steps of producing an obfuscated code version using our proposed system. 51
4.2 LLVM cross-compilation steps. 52
4.3 Obfuscated code versions of the Float program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 54
4.4 Obfuscated code versions of the IntMM program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 55
4.5 Obfuscated code versions of the Perm program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 56
4.6 Obfuscated code versions of the Queen program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 57

4.7 Obfuscated code versions of the Quick Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 58
4.8 Obfuscated code versions of the Puzz program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 59
4.9 Obfuscated code versions of the Oscar program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version. 60

5.1 Our proposed secure AVCC network model. 62
5.2 Encryption time results across different ID sizes for index and trapdoor. 68
5.3 Matching time results across different ID sizes. 68
5.4 Indices encryption time results with different numbers of users for $n = 64$ & $n = 32$. 70
5.5 Search time results with different numbers of users for $n = 64$ & $n = 32$. . . . 70

6.1 The considered network model. 76
6.2 Encryption times across different iris sample sizes. 82
6.3 Comparing our average search times across different iris sample sizes with the work of Rajasekar et al. 82

LIST OF TABLES

Table

2.1 Security & Privacy Attacks On Traffic Flow Optimization Applications. 10
2.2 Security & Privacy Attacks On Platoon Applications. 13
2.3 Security & Privacy Attacks On Carpooling Applications. 17
2.4 Security & Privacy Attacks On Parking Applications. 22
2.5 Security & Privacy Attacks On IoV Applications. 26
2.6 Security & Privacy Attacks On The Operating System Level. 29
2.7 Security Attacks On Vehicular Networks and Their Countermeasures. 31
2.8 Security & Privacy Attacks On The Physical Layer. 35

4.1 Comparison of Range of Normalized Runtime Differences in Both Versions of Our System. 54

5.1 Security & Privacy Attacks On AVCC Applications. 71

6.1 Main Notations We Used in Our Technique. 78
6.2 Communication Overhead. 81
6.3 Computation Overhead. 81

LIST OF ABBREVIATIONS

ACC - Adaptive Cruise Control
AI - Artificial Intelligence
AV - Autonomous Vehicles
AVCC - Autonomous Vehicles Cloud Computing
AVI - Automated Vehicle Identification
CA - Certificate Authority
CAN - Controller Area Network
CACC - Cooperative Adaptive Cruise Control
CC - Charging Company
CCU - Connectivity Control Unit
CFG - Control-flow Guard
CP-ABE - Ciphertext-Policy Attribute-Based Encryption
CPS - Cyber-physical Systems
CS - Charging Station
CSP - Cloud Service Provider
DMV - Department of Motor Vehicles
DL - Deep Learning
DoS - Denial of Service
DDoS - Distributed Denial of Service
DRTM - Dynamic Root Trust Management
DSRC - Dedicated Short Range Communications
ECC - Elliptic Curve Cryptography
ECU - Electronic Control Unit
EV - Electrical Vehciles
FHE - Fully-homomorphic Encryption
HOV - High Occupancy Vehicle
ICMetrics - Integrated Circuit Metric technology
IEE - Isolated Execution Environment
IoV - Internet of Vehicles
IPE - Inner Product Encryption
ISA - Instruction Set Architecture
KNN - k-Nearest Neighbors Algorithm
LIN - Local Interconnect Network
LLVM - Low-Level Virtual Machine
MOST - Media Oriented Systems Transport
MSCC - Multiple Sensor Consistency Check
NRS - Non-transferable Ridesharing Service

OBD	-	On-board Device
OBU	-	On-board Unit
OTA	-	Over-The-Air
OTP	-	One-time Password
P3	-	Privacy-Preserved Pseudonym
PSA	-	Physical Shift Authenticates
PUF	-	Physically Unclonable Function
ROR	-	Real-Or-Random
RSU	-	Roadside Units
SCA	-	Side-channel Attack
SRTM	-	Static Root Trust Management
SWD	-	Serial Wire Debug
TA	-	Trusted Authority
TCP	-	Trusted Computing Base
TLS	-	Transport Layer Security
TMC	-	Traffic Management Center
TOS	-	Trip Organizing Server
TRS	-	Transferable Ridesharing Service
UART	-	Universal Asynchronous Receiver-Transmitter
UAV	-	Unmanned Aerial Vehicles
V2V	-	Vehicle to Vehicle
VANET	-	Vehicluar Ad-hoc Network
VCC	-	Vehicular Cloud Computing
VENTOS	-	Vehicular Network Open Simulator
VIN	-	Vehicle Identification Number
VLC	-	Visible Light Communication
VN	-	Vehicular Network

Chapter 1

INTRODUCTION

The industrial revolution is still evolving, and now we are in the most significant shift of all, eliminating the need for the human factor. Artificial intelligence, machine learning, and intelligent robotics can already replace humans in various fields, such as manufacturing, medicine, economics, education, and public safety. One key field, which long suffered from human mistakes, is transportation. Hundreds of thousands of people die in car accidents every year [1]. The total adoption of autonomous driving systems would significantly decrease human errors and allow for more efficiency in various aspects, such as better fuel utilization, lower accident rates, and of course, passenger welfare while offering a pleasant entertainment-rich experience.

Moreover, the computing power of Autonomous Vehicles (AVs) is quickly increasing. AVs are outfitted with different processing, memory and storage facilities, as well as computer vision technologies [2]. In addition, there is a myriad of sensors and actuators that are connected to each other and to their surroundings via various communications interfaces, allowing them to drive themselves autonomously without the need for human control. Nevertheless, relying solely on the vehicle's sensors, such as proximity sensors, cameras, and light detectors, is not enough for AVs' safe operation. An autonomous driving system would not have thrived without the need for networking. That is because these sensors may have physical limitations, which may result in making erroneous decisions [3]. That is why vehicles need to communicate with each other to make up for these deficiencies by exchanging information on road and traffic conditions, thereby improving the navigation of vehicles. Hence, serious accidents can be mitigated if vehicles communicate continuously, thereby avoiding collisions and improving road safety [4]. Also, the autonomous driving system needs a continuous connection to the car manufacturer's cloud, which monitors the vehicle's condition and provides aid if needed and also send necessary firmware updates [5].

Furthermore, AVs frequently collaborate to collect data from their surroundings and transmit it to distant servers, where it may be processed and analyzed to deliver various services. Such that, in addition to navigation, some collaborative applications were introduced

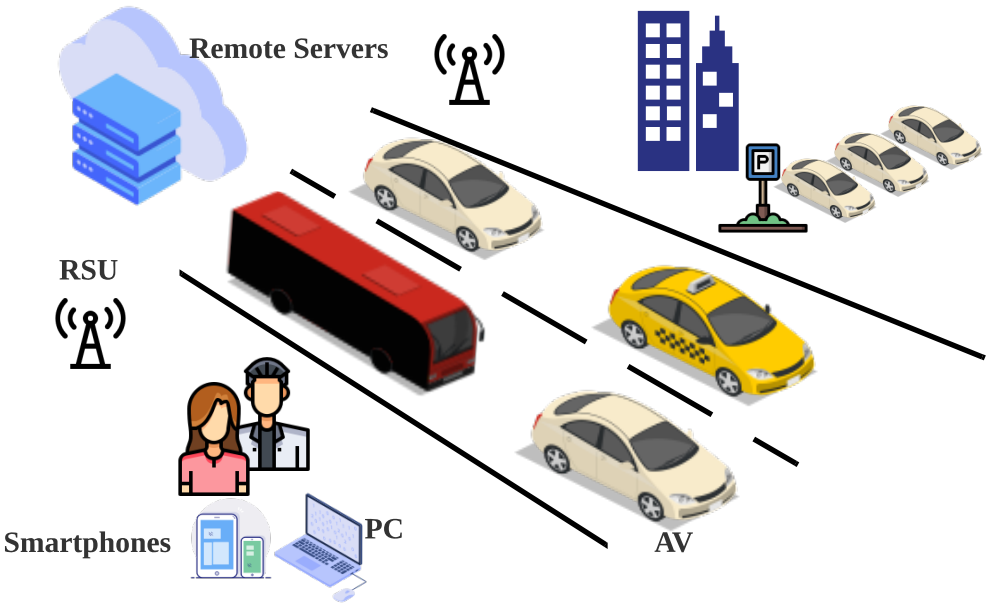

***Figure 1.1*:** Autonomous vehicle cloud computing system.

to monitor the environment and pollution levels or aid in wide-scale traffic management systems. Hence, a paradigm called the Internet of Vehicles (IoV) [6] was born, which is the next level of Wireless Sensor Network, where the cars themselves act as the information hub.

Unfortunately, regardless of the expected benefits of AVs, these systems are still facing plenty of security issues and privacy concerns [7]. Vehicles, which used to be all-in-all mechanical systems, are now inheriting computer systems problems that are susceptible to a wide range of unexpected attacks. This happens more often when these systems are connected via communication networks, thereby opening the system to the outside world or even malicious insiders in the network. Additionally, modern vehicles allow users to connect their smartphone, other portable, or more recently wearable devices via various wired (AUX, USB, etc.) and wireless interfaces (WIFI, Bluetooth, etc.) in a seamless integration [8]. This sort of connection allows for sharing data between the vehicle and the mobile device for playing music, answering phone calls, checking social media notifications, and surfing the web [9]. These devices may make the car system susceptible to new attacks and vulnerabilities since they are inherently prone to hacking and malware programs [10].

Security attacks are an enormous threat to autonomous driving systems. Successful

cyberattacks may cause system failure, which may lead to accidents and thus losing human lives. Moreover, malicious hackers can deliberately target a particular vehicle and disrupt its normal operation to steal it or even harm others or cause any damage. In addition, privacy is a major concern in AVs. The continuous communication between the vehicle and its surroundings puts the user's private data at risk. Imagine an intelligent car equipped with cameras and a microphone, and a variety of sensors that can be used to harness troves of data on the car's passengers [11]. Potential threats include information leakage, identity theft, tracking, and stalking. Users may not trust technology providers because they may collect sensitive information and sell it to interested parties.

That said, researchers took the computing potential found in AVs to the next level. They aim to utilize these smart cars' occasionally inactive computing capabilities to provide computing services as a utility. This model is called autonomous vehicles cloud computing (AVCC) [12]. Cloud computing is a relatively new technology that is currently game-changing in the industry. Users don't need to own powerful computing capabilities at their hands. Instead, they can rent as much power as they need in a pay-as-you-go model. The advancement in communication technologies such as LTE and 5G allows a gradually ubiquitous spreading of this new paradigm. Cloud computing is being offered in different delivery models suiting different user needs. There is Software as a service, Platform as a service, and Infrastructure as a service. In other situations, some companies opted to use cloud computing platforms to allow their employees a more flexible working style. In times of hardship like nowadays, the pandemic forced many people to work from their homes, and they needed to access the company's computing resources seamlessly with the same functionalities. These models have something in common; some computation task is done remotely in a physically out of reach platform that the user cannot control or govern.

Remote code execution is a trending requirement in numerous usage scenarios, such as a case when a user is using a smartphone or a small computer. In other situations, some companies opted to use cloud computing platforms to allow their employees a more flexible working style. In times of hardship like nowadays, the pandemic forced many people to work from their homes, and they needed to access the company's computing resources seamlessly with the same functionalities. On the other hand, all of these usage scenarios suffer from common security threats and privacy concerns. More importantly, remote code execution on shared platforms that are physically inaccessible is inherently risky in terms of trustworthiness. That is to be confidential, integral, and available at time of need.

These requirements become more challenging in the field of AVCC. Although AVCC is

essentially a cloud computing platform, using cars instead of stationary computers residing in some company's building introduced more challenging problems. The first obvious problem is that these cars are moving, which means that the communication interfaces will continually change the cloud formation. Although the organizational problems are addressed from an architectural perspective, they open the system to unknown threats every time a car enters or leaves the cloud [13]. Secondly, AVs are powered by embedded systems, which means they have power limitations and are also limited in processing capabilities, storage, and memory.

AVCCs are indeed a bonanza of a variety of security attacks such as unauthorized access, intrusion, Denial of Service (DoS), jamming, hijacking authentication, racketeering, copyright infringements, stealing data, sabotage, and information leakage via side-channels and reverse engineering. This new breed of attacks is resilient to traditional open security methods, relying on conventional cryptographic approaches. Digital signatures, certificates, and trusted platforms are examples of traditional cryptographic approaches that may not be sufficient. Since the code is running remotely, attackers may be able to view the actual decryption process and get the information [14]. Homomorphic encryption [15] is a fairly new technique to support encrypted execution of instructions. Even so, it is not quite applicable yet, since it requires complicated setup hardware and tremendous practical cost, which may not be suitable in the embedded systems environment. Therefore, we need new practical security approaches to remotely protect code execution from potential attackers who share the same physical hardware in the AVCC especially against side-channel attacks.

On the other hand, implementing a secure and trustworthy authentication system is an eventuality for AVCC to reduce the misuse of the system and increase their spread. A lightweight and efficient encryption mechanism is needed to provide a privacy-preserving robust encrypted access scheme. A limitation of existing searchable encryption techniques [16–18] is that anyone who receives the encrypted ids, for example, by overhearing AVCC communications, can compute the similarity score and deduce side information. Hence, it's required that the cloud server would be able search over encrypted data using its secret key. Additionally, the system should have a low overhead in terms of communications and computations. This will allow for a fast response such that user identification and verification are made quickly and efficiently with minimum wait time for the user. Also, we need a more lightweight, yet secure, authentication mechanism for services related to the AVCC operation such as the electrical charging systems.

In this dissertation, we aim to tackle some of the aforementioned problems, in particular

software protection and secure authentication for the AVCC platform. The remainder of this dissertation is organized as follows. Chapter 2 shows an overview of the literature related to the major security and privacy challenges in the realm of AVs, and some of their mitigation strategies, presented in model that examines each layer of the AV system architecture. In Chapter 3, we focus on AVCC application security and we present an obfuscation-based software protection mechanism. In Chapter 4, we propose an enhancement to the former technique, that address some of its shortcomings and more applicable to some usage scenarios. In Chapter 5, we developed a secure authentication mechanism for AVCC platforms the protects both the cloud users and vehicle participants. In Chapter 6, we developed another, more simple, authentication mechanism for charging systems that power electrical vehicles. Finally, in Chapter 7, we conclude the dissertation and suggest directions for future work.

Chapter 2

SECURITY AND PRIVACY ISSUES IN AUTONOMOUS VEHICLES: A LAYER-BASED SURVEY

2.1 Introduction

Autonomy has long been a sought-after goal in vehicles, and now more than ever we are very close to that goal. But this new paradigm shift comes with newly introduced privacy issues and security concerns.

In this chapter, we present a survey on security and privacy issues in autonomous driving systems. We classify these issues from a layer-based perspective inspired by the TCP/IP network model [19], which was based on the OSI reference model of computer networks [20]. Figure 2.1 shows the layered structure of the AVs system that we are focusing on; application layer, operating system layer, network layer, and physical layer. We are not concerned with software or hardware faults arising within the system itself. Instead, we focus on the dangers of opening the system to its surroundings and the outside world via various wired or wireless communication channels.

The remainder of this chapter is organized as follows. Section 2.2 shows an overview of the literature related to a variety of application layer programs that are quite popular in the realm of AVs, their major security and privacy challenges. The vulnerabilities of the AV operating system layer are investigated in Section 2.3. Section 2.4 focuses on the security issues regarding the network layer. In Section 2.5, we discuss the hardware attacks on AVs. Additionally, in Section 2.6, we show some related works that surveyed AV security and privacy issues. Finally, in Section 2.7, we summarize the chapter and suggest directions for future work.

2.2 Application Layer Security

Autonomous driving systems allowed for a new generation of applications, some of which may have existed before, but with the vehicles being able to drive themselves, researchers had to revolutionize these applications to make use of the new possibilities. And with the new possibilities comes new security risks that must be addressed before the total adoption

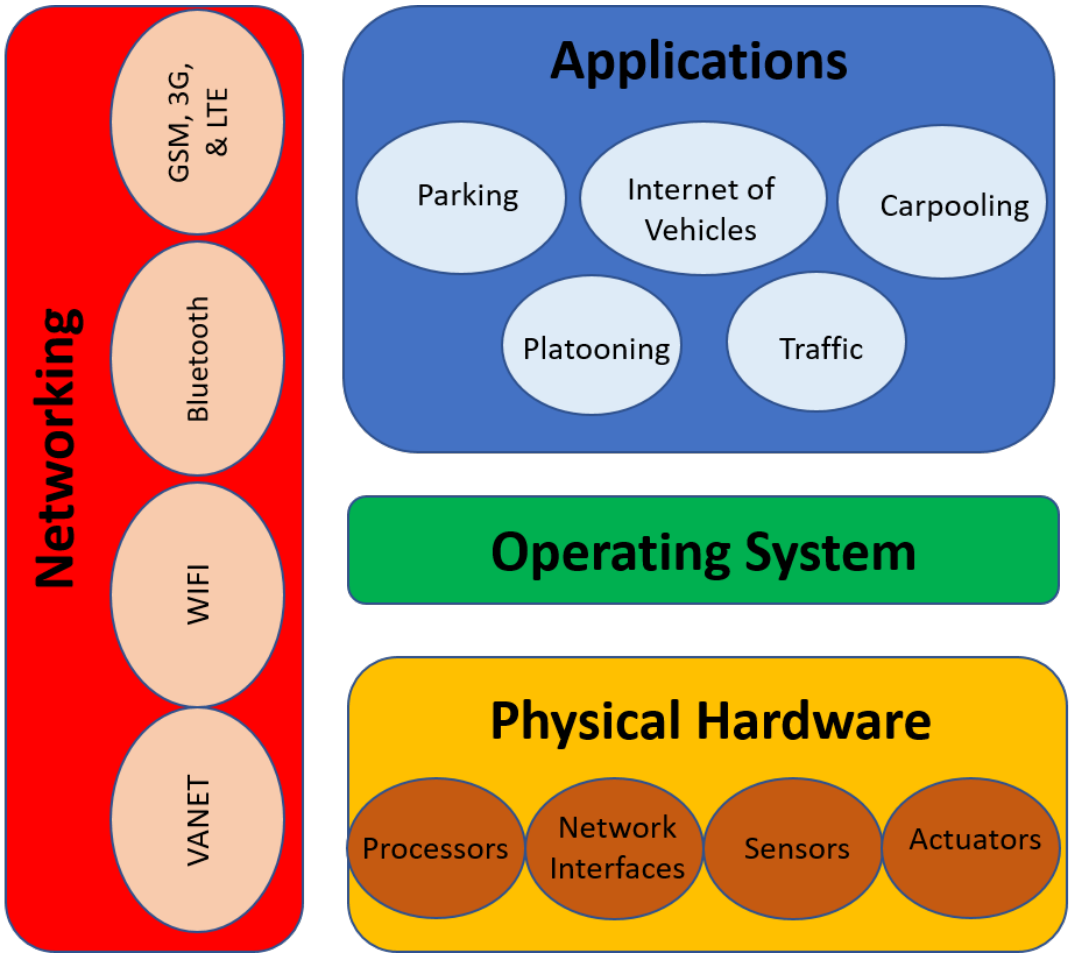

***Figure 2.1*:** The expanded layered structure of autonomous vehicles system.

of these applications and creating room for much more. Some example application includes; but not limited to, carpooling, automated valet parking, automated electrical charging (since most modern AVs will be electrical), sensor data gathering, forensics, platoon stability, safe navigation and crash prevention applications; video upload (e.g., an accident scene, Pic-on-wheels, remote drive, ...etc.), news, entertainment, location-relevant info download, driver behavior study, traffic crowdsource, mitigating congestions/pollution by efficient routing and intelligent transport [21]. Here we shed some light on primary application areas and an overview of the work being done to secure them.

Traffic Flow Optimization

In traffic flow optimization, traffic is directed through a road network in order to minimize travel times, eliminate traffic congestion, and maximize overall road infrastructure utilization. Currently, new approaches seek to influence vehicles through the adaptation of traffic signal schedules [22] [23] and digital signage, as illustrated in Figure 2.2. When AVs are present, the method can take an entirely new form.

Many researchers tried to simulate and study the traffic management systems, such as [24,25]. Some researchers focused on ramp metering as in [26], while others concentrated on intersection management on a larger scale such as [27] and [28]. That said, besides the AV traffic management algorithm itself, which is sometimes quite complicated, there are side problems that need to be handled [29] such as the coexistence of AV with human-driven cars, pedestrians, motorcyclists, etc.; dealing with complicated situations such as severe weather conditions or natural accidents. There are also data handling and communication problems.

Here we focus on traffic messages being communicated by the vehicles. These are vulnerable to a variety of network-level security attacks that we will further investigate in Section 2.4. Moreover, stakeholders' privacy is of primary concern, and there has been a considerable effort to protect it.

In [30], the authors proposed a model based on data quality evaluation metrics to augment trust and reputation. Their proposed model can detect agents that supply incorrect or fraudulent data; thereby, they can be removed, and hence the overall system accuracy can be enhanced.

Another piece of work that we studied is [31], where the authors proposed a scheme to protect the privacy of vehicles sharing their routes for the purposes of traffic-flow optimization. Knowing that these systems need only to learn the number of vehicles traveling in the same road segment, anticipating possible congestion situations, the authors developed a segment-based route reporting system that sends data encrypted using homomorphic encryption to roadside units (RSUs). All segment data from different vehicles are then collected at RSUs. It computes the encryption of the number of anticipated cars occupying every road segment without knowing the actual routes of vehicles. Then, this information is shared with a traffic management center (TMC), that computes the decryption and extracts the same number while individual vehicles' routes are hidden for privacy preservation. After analyzing this information, the TMC can send directions to traveling cars about traffic conditions and congestion.

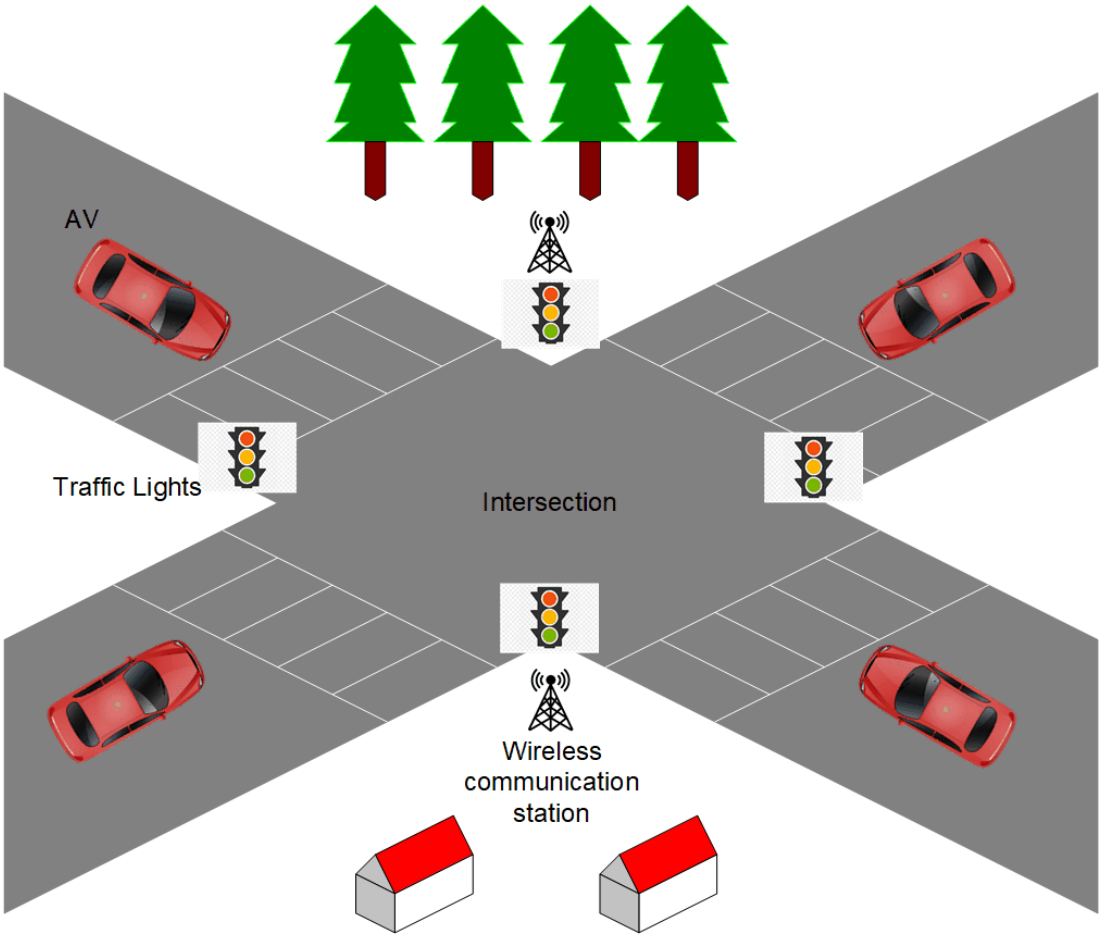

***Figure 2.2*:** A typical traffic management system would control intersections, traffic lights and ramp meters among other things.

Another significant issue affecting traffic systems is Sybil attacks. In such a type of attack, individual attackers can imitate multiple vehicles that submit a bogus incident down the road. Generally, a traffic system requires information regarding events in order to notify vehicles of unanticipated road hazards and ensure safe driving. The authors of [32] presented a secure event-reporting mechanism to thwart Sybil attacks while maintaining users' privacy. The proposed scheme distributes a set of pseudonyms/keys to enable reporting of incidents without disclosing private vehicle information. Additionally, the authors proposed a method for identifying the vehicles that conduct Sybil attacks using their pool of pseudonyms. The proposed scheme classifies vehicles into groups, and the RSUs only know the group numbers of the vehicles, not their identities. If the pseudonyms keys were used to report an incident related to the same group, the RSUs would suspect a singular Sybil attack. In this situation,

***Table 2.1*:** Security & Privacy Attacks On Traffic Flow Optimization Applications.

Ref. No.	Threat	Proposed Method	Critique
[30]	Data integrity	Data quality evaluation metrics	Susceptible to Sybil, man-in-the-middle, and repudiation attacks.
[31]	Privacy of vehicle routes	Segment-based route reporting and homomorphic encryption	Incurs performance cost in terms of computation overhead, communication cost and power consumption.
[32]	Sybil attacks	Pseudonyms/keys and vehicle grouping with RSUs	Management of groups may become an overhead. Also, there is considerable communication overhead.
[33]	Data Integrity	Road Monitoring using UAVs	Requires huge set up cost, and maintenance of the UAVs. Also, they become honeypot for attacks.

the RSUs transmit the messages to the department of motor vehicle (DMV). Sybil attacks are recognized when the DMV determines that the suspicious pseudonyms are associated with a particular vehicle. Even if numerous attackers collaborate, the RSU will detect it when it compares the beacon packet signals from the vehicles to the reported events.

More interestingly, the authors in [33] went as far as using unmanned aerial vehicles (UAV) to assist with road monitoring. They proposed a so-called security situational aware intelligent traffic monitoring, which can re-route the traffic, offer guidance instructions for shortest routes with the help of a traffic management security control center. They combine information from graph theory representation of the road map with sensory networks. UAVs' usage enables real time and rapid response to different security cases, since it shouldn't encounter any hindrances to wireless communications in comparison with traditional traffic monitoring systems.

Although the discussed techniques may be promising, we think that in the future, some of these techniques can be integrated together into a more comprehensive intelligent vehicular transportation system, which is trained against different security attack scenarios and protect the stakeholders' data. Table 2.1 summarizes these findings and our critique of them.

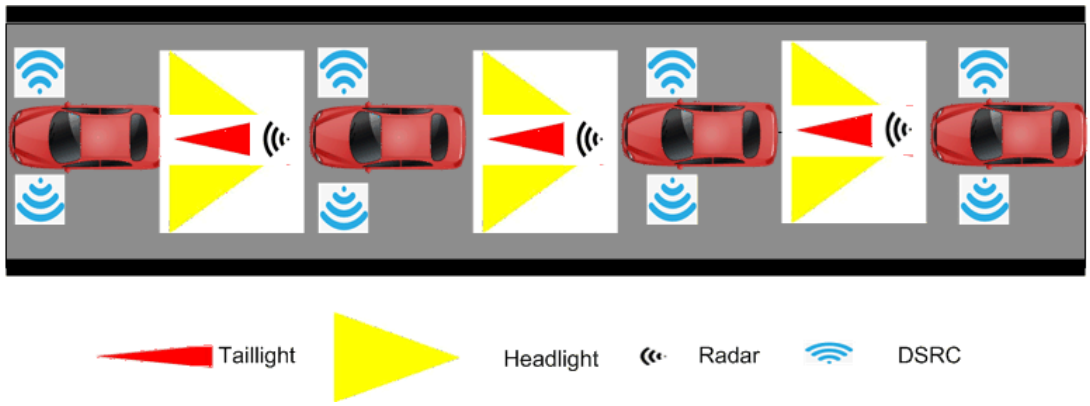

***Figure 2.3*:** An autonomous vehicles platoon and the various sensors they use.

Platooning

AVs platoon, as depicted in Figure 2.3, is an advancement of autonomous conduct in which AVs are assembled into clusters of cars in close range and communicate wirelessly [34]. Cooperative adaptive cruise control (CACC) can be thought of as an improved version of adaptive cruise control (ACC) that is used in this group of vehicles [35]. This system enables vehicles to keep a proper distance from one another and make cooperative navigation decisions. Since the vehicles in a platoon work together to plan ahead and drive closer together, the platoon improves traffic flow. In addition, the ability to respond to events more quickly than drivers improves transportation safety. Moreover, lessening the amount of time spent accelerating and decelerating on the road helps save fuel and reduce emissions.

Most platoon members communicate via IEEE 802.11p, the most common vehicular RF technology. However, this technology has vulnerabilities that various malicious attackers can exploit. In applications that involve collaborative driving, an unexpected surfacing of a security threat may endanger the following: (i) the integrity of traffic flow messages on the AV network by submitting fake data that change the platoon organization and coordinated movements; and (ii) the stability of platoon applications by affecting communication capabilities in the AV network .

AVs in a platoon are more tightly coupled than ordinary cars, making them more vulnerable to attacks targeting their platoon system. In [36] the authors showed that a single malicious AV could destabilize an entire platoon and cause catastrophic events. This malicious car combines some changes to the gains of the control law with vehicle movements, thereby forcing the platoon to oscillate at a resonant frequency and violating the platoon string stability features, resulting in fatal accidents. In [37], the authors managed

to mimic a high-speed collision induction attack by overtaking the platoon controller. They manipulated the Dedicated Short Range Communications (DSRC) to cause an unexpected behavior from a platoon member, thereby causing collisions. In [38] the authors showed that a malicious AV could introduce a precise effect on the mobility of vehicles in its vicinity, thereby raising the energy expenditure of neighbouring vehicles by 20% to 300%.

In [39,40] the authors provided an in-depth study into the control laws of AV in a platoon. They showed that one malicious AV could introduce erroneous traffic messages that other vehicles can then amplify, causing traffic jams or accidents. They studied the conditions under which the attacker can disrupt the AV stream and the string stability and proved that such disruption will self-perpetuate as one of the vulnerabilities of relying on AV compared to human-driven cars unless additional inputs are provided.

By utilizing light's directivity and impermeability, visible light communication (VLC) can mitigate these vulnerabilities. On the other hand, using just VLC in a platoon might affect its safety due to VLC's sensitivity to environmental effects. SP-VLC [34] is a VLC and IEEE 802.11p based protocol for securing communications in platoons. This protocol ensures stability of platoons and security of their motions under different types of attacks such as jamming, channel overhearing and injection of data packets. They describe these maneuver attacks by defining various circumstances in which a malicious party sends a forged maneuver packet. SP-VLC contains techniques for establishing an encryption key, authentication of messages, communications over both VLC and IEEE 802.11p, detection of jamming attacks and response to switching to VLC-only transmission, and movement safety depending on both VLC and IEEE 802.11p. Additionally, they develop a simulation platform that incorporates a realistic vehicle mobility model, realistic VLC and IEEE 802.11p channel models, and platoon management for vehicles.

In [41], the authors investigate how to secure AV platooning when an unknown vehicle is attacked and bounded system uncertainties occur. A malicious attacker can arbitrarily modify the attacked vehicle's GPS position and speed measurements. In the beginning, two detectors were proposed to determine which car is being attacked based on relative measurements (camera or radar) and local information gathered from measurements of surrounding vehicles. They next create a local state observer for each vehicle based on the detectors' data by using a saturation method to the measurement innovation. Additionally, based on the observer's neighbor state estimates, a distributed controller is presented to

***Table 2.2*:** Security & Privacy Attacks On Platoon Applications.

Ref. No.	Threat	Proposed Method	Critique
[34]	Jamming, channel overhearing and injection of data packets	SP-VLC	Attacker with enough knowledge about the switching scheme can exploit VLC vulnerabilities, if coupled with other attack vectors.
[41]	Unknown vehicle is attacked and bounded system uncertainties occur	Relative measurement and local state observer	Susceptible to collusion, the observer can be a honeypot for attacks.
[42]	Message falsification and spoofing attacks	Cooperative control technique	Voting algorithm can be exploited by colluding vehicles.
[43]	Attacks on Message authentication and security	VPKIbrID	Public key infrastructure and Attribute Based Encryption require large computation cost, communication overhead and power consumption.

establish vehicle speed consensus and maintain a stable desired distance between two neighboring vehicles. Under certain conditions, it was demonstrated that the observer's estimate error and the controller's platooning error are asymptotically upper-bounded.

In [42], the authors focus on several important forms of attacks which impact security of platoons. They examined Burst Transmission and DoS attacks, which affect the network layer. They also studied message falsification and spoofing attacks, that target the application layer. The paper proposed a new closed-loop collaborative control technique for strengthening autonomous platoon security. They also implemented their system in PLEXE [44] and analytically proved its claimed stability.

In [43] the authors studied the security of underlying vehicular network which supports platoon management protocol. The proposed model uses Public Key Infrastructure and Attribute Based Encryption with Identity Manager Hybrid (VPKIbrID). This robust encryption scheme would ensure message communication authentication and security. They simulated their system using various sizes of platoons and different modes of VPKIbrID propagating different numbers of multicast/broadcast messages.

As we can see from this investigation, platooning is a complex application that requires security precautions at several layers. The interactions between the AV, the Connectivity Control Unit (CCU) should be strongly authenticated and defined in a lightweight key management scheme. A secure design for a DSRC communication protocol robust to different attack types, including packet falsification, replay, jamming, membership falsification, and hijacking, is still needed. In Table 2.2 we summarize this discussion and our comments.

Carpooling

Carpooling or ride-sharing has emerged in the last decade. Carpooling is a system which requires different individuals having interchangeable journeys to ride together in one car at the same time, instead of having different cars [45]. By decreasing the quantity of cars on the roads, carpooling will lessen air pollution and traffic congestion. It can also share the cost of the trip between several people.

Companies like Uber and Lyft certainly revalorized the transportation industry and made a seismic shift in the employment market. In many countries now, and as a part-time job, many users utilize their vehicles as ride-hailing vessels through these apps, thereby significantly enhancing their income. On the other hand, from the passenger side, these apps allow for a much more convenient customer experience when dealing with an easy-to-use app, precisely calculated fare in advance, and an easy-to-use customer complaint and dispute facility through the same apps. Eliminating the driver from the whole equation is a perfect opportunity for every car owner who had been toying with the idea to use his car in carpooling. He no longer has to sacrifice time and effort to make some extra money. Service-providing companies may not worry anymore about customers having uncomfortable situations with drivers. We have to mention that some companies may opt to abandon this decentralized business model altogether and build their fleet of autonomous vehicles, but this may not be economically attractive as the first model.

The use of ride-sharing has risen considerably [46]. As of 2010, North America had at least 613 platforms for ride-sharing organizations primarily based on the internet [47–49]. Additionally, many government programs have already been made to allow people to share trips. One such strategy is to make high occupancy vehicle (HOV) lanes. This requires a reserved lane to be used for cars with more than two passengers on board [50, 51]. Another tactic for promoting ride sharing is to include toll reductions, and refunds [52].

Figure 2.4 shows a typical architecture of an AV carpooling system. The organization

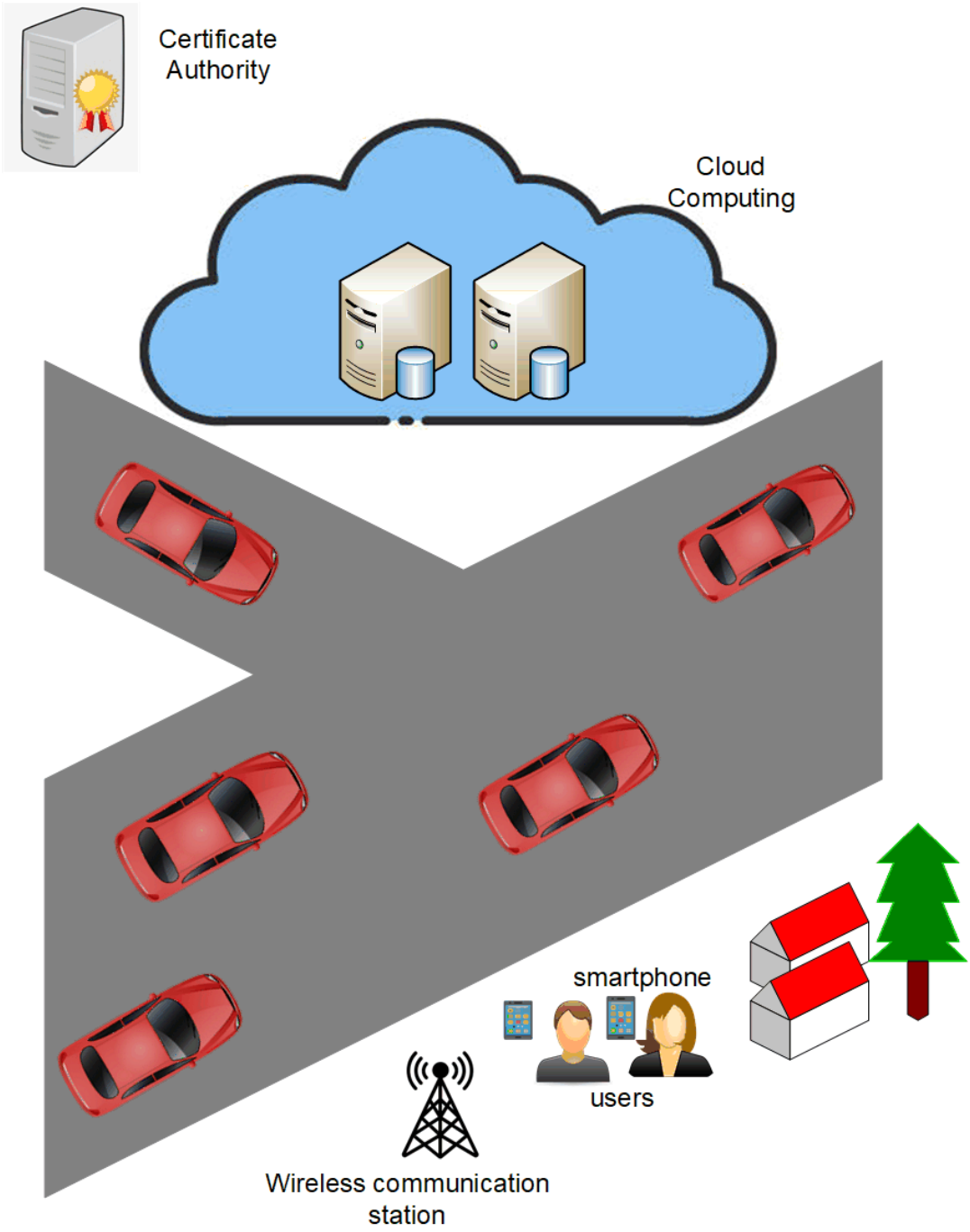

***Figure 2.4*:** Architecture of an AV carpooling system.

of shared rides can be significantly improved by the use of Internet access, GPS systems and smartphones. Customer registration is expected to be done through a web portal

which manages shared trips, and afterwords submit a ridesharing offer to a Trip Organizing Server (TOS) via a car owner (operator) and wanting to share a trip with other customers (riders). This offer would provide details on the travel, such as the location, destination, time of the trip, and direction. Moreover, passengers requesting mutual transportation can first send requests for ridesharing with specific details to the TOS. Then, TOS compares offers from drivers with requests from riders, thus allocating each driver to one or more passengers. Nevertheless, the TOS is owned and managed by a private corporation that may gather information about the locations and behaviors of the customers and may launch user impersonation, forgery, and replay attacks.

But the catch is how to make the customer trust these indigent cars with his/her private data such as the locations they visit, their trips, their usage patterns, and use the service with a level of anonymity. On the other hand, car owners should trust that their cars will not be stolen, vandalized, or used in some malicious endeavors. The service provider should worry about both the car owner and the customer, and at the same time, protect itself from untrusted users who try to hack the system for their gain. Some users may report false locations to match certain passengers/cars or attract traffic to some area for malicious purposes.

Although most of the work on carpooling applications focused on human-driven cars, similar ideas can be applied to AVs. Here we investigate some of these ideas and discuss how they relate to AVs.

In particular, secure and privacy-preserving schemes for ride-sharing is now an essential need to flourish the usage of AVs [53, 54] In [45], the authors proposed effective privacy-preserving ride-sharing management techniques for transferable and non-transferable ride-sharing systems. In a non-transferable ridesharing service (NRS), a TOS would execute matching over encrypted data to associate a single driver to each rider. Nevertheless, TOS utilizes trip data from the drivers in the transferable ridesharing service (TRS) to construct an encrypted directed graph for the management of carpooling. Preferences of the riders are used to determine the weights of the graph's edges. Nevertheless, TRS provides an attractive service that can expand ridesharing. At the same time, NRS offers a valuable and convenient service for the aged and disabled, who do not want to switch between multiple drivers.

One promising piece of work [55], entirely suited to this seemingly decentralized model, is the use of blockchain technology. The authors proposed to build a private blockchain ledger to store all carpooling records. They also proposed storing locations grid into a tree and achieving drop-off locations matching by a range query scheme. They also adopted a privatized proximity check to attain one-to-many proximity matching and expand it to

***Table 2.3*:** Security & Privacy Attacks On Carpooling Applications.

Ref. No.	Threat	Proposed Method	Critique
[45]	Privacy of user and AV data	TOS executes matching over encrypted data	TOS is the central point of failure in the system, also, it doesn't account for collusion and double booking attacks.
[55]	Compromising privacy of user and AV data	Blockchain	Susceptible to fraudulent submissions. Also, the heave computation overhead.
[56]	Compromising Location privacy	Central cloud server, anonymous authentication	The central cloud is a honeypot for attacks. Also, there is more communication overhead and setup costs.
[57]	Compromising privacy of user and AV data	Blockchain and Time-locked deposit	The system is too complicated and difficult to be scalable.

effectively set up a secret communication key between a rider and a driver. In [56] the authors went into more details about users' privacy location tags, range queries, and anonymous authentication. But the difference here is that instead of using fog computing, they relied on a central cloud server. They were working under the assumption that this server might be honest-but-curios. That is why they suggested using blockchain technology to record all the hashes and trip records.

Additionally, the authors of [57] proposed B-Ride, a decentralized ride-sharing service built on public Blockchain. They examined a scenario in which fraudulent users could take advantage of the anonymity given by the public Blockchain to submit many bogus ride requests or offers while remaining uncommitted to any of them to secure a better deal or render the system unstable. As a result, the paper developed a time-locked deposit mechanism based on smart contracts and zero-knowledge set membership proof. The driver and the rider must demonstrate exemplary conduct by making deposits to the Blockchain. This will be considered when determining the fair in a pay-as-you-drive system based on the driver and rider's elapsed distance. Additionally, they implement a reputation model to assess drivers based on their prior behavior without involving third parties, allowing riders to choose drivers based on their history on the system.

In summary, the introduction of new prospects from AVs will bring forth a massive transition into carpooling applications. In this context, ongoing research efforts should address a situation where adversaries compromise the RSUs. Tamper-proof Blockchains are not enough since we need a mechanism to authenticate data feed from users in case of disputes [58]. Crowdsourcing mechanisms [59] can be integrated to be a witness in claims and proofs. Anonymous, yet secure payment mechanism for carpooling services is also an open area of research. Table 2.3 gives a summary of the literature discussed above and our thoughts of them.

Parking

Parking is one of the most significant issues in many major cities worldwide. Due to the increase in population density and the number of cars in the streets, it is not always easy to find a spot to park your vehicle near your home, the place you work at, or even in front of the shop or the venue you are visiting. During special events and holidays, this could be a nightmare. That's why many cities built big parking lots in every neighborhood, but sometimes they may not be close to your desired destination. Moreover, in many cases, you may have to visit multiple places to find an empty spot, thereby losing more time, effort, fuel, and at the same time, causing unnecessary traffic load to the already crowded cities.

It is worth mentioning that big tech giants, which may not seem directly involved with car manufacturing, have numerous patents dealing with various aspects of parking in AVs. Such as [60] which is owned by IBM and provides an algorithm to optimize the delay in the parking process, which detects the last embarking passenger of the car and then initiates the parking process automatically. It deals with the steering system, transmission controls, and brakes to cause the vehicle to park in and out of parking spots from/into the roadway. On the other hand, car manufacturers are still developing ways to deal with physical control issues of parking systems. For example, in [61], Ford developed an electric braking system tailored for parking AVs. Also, in [62] Volkswagen developed a system for data processing for obtaining the operational state of AVs.

Having an AV park itself is an excellent idea, and it will save much-needed time and effort. But there must be coordination between the parking lots and the AV to guide through the process. Figure 2.5 depicts an example of an automated parking system for AVs. An efficient system would have an app that tells you where to find empty parking spots nearby the destination you want. Then your car would drop you off and go there to park itself.

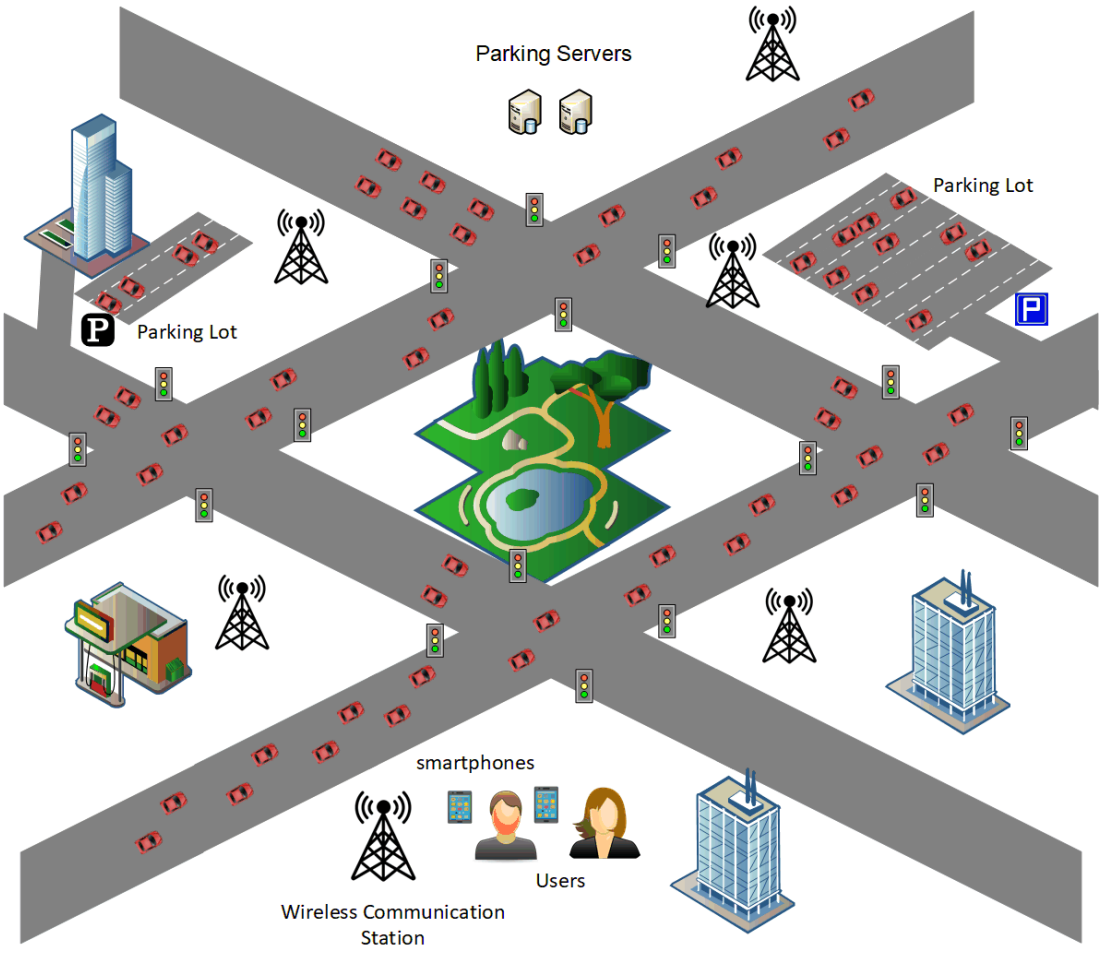

***Figure 2.5*:** An example of an automated parking system for AVs.

When you finish your business there, you will order the car to come to pick you up. All payment and handling fees would be automatically charged to your account in a seamless manner. Sounds perfect, but here is the catch. How do you trust that your car will not be stolen or vandalized. An inside attacker may thieve other parked AVs by launching attacks such as man-in-the-middle, forging, replay, and impersonation attacks. Theses attacks also can be launched remotely by an outside attacker. Their objective is to compromise the authentication between AVs and smartphones that control them, hence, he could start the motor and run away with the AV.

Another major problem is how you would trust that your location privacy will not be compromised. Here, the attacker may try to identify a specific user's location. Moreover, a nefarious attacker could cooperate with other adversaries and penetrate the system and

automated parking computers to predict a victim user's location based on the position of AVs.

Additionally, an adversary may actually follow some particular AV and intercept every messages it receives and obtain access to the local server's and automated parking system server's secret keys. If the attacker is not capable of knowing the AV owner in advance, the attacker cannot compromise the user's location privacy. Another concern is slandering, in which an attacker may pretend that an honest user scratched his vehicle. The attacker could be a registered but malicious user who provides the judge with enough location information to classify an honest user as a scratching perpetrator. Another instance of defamation is when a legitimate but wicked user slanders the parking lots, claiming that his/her AV was lost, when he/she already picked it up before. Additionally, certain involved parties, such as clients, parking lots, or thieves, might possess distinct motives to conceal their misbehavior. First, the adversary, a legitimate client, attempts to send an untraceable message to pickup the AV. As a result, he could use defame the parking lot by saying his AV was lost. A legitimate user could potentially remain anonymous in order to avoid being compensated if a traffic accident happens, during which his AV was implicated. Second, the parking lot is more inclined to remain inconspicuous throughout any accident inquiry to minimize complications. Finally, an attacker who steals an AV may prohibit the authorities and the client from locating the vehicle.

That is why the researchers proposed authentication schemes to address these concerns and facilitate the implementation of automated valet parking protocols. [63] proposes a parking protocol that is supposedly automated and secure to protect against vehicle hacking or theft. They proposed two-factor authentication using a one-time password (OTP) and a smartphone was introduced to protect remote control of AVs and defend against malicious access. Hence, an adversary using a smartphone alone or one-time password will not be able to control the AV or steal it. Using the BBS+ signature [64] and the Cuckoo filter [65], they were able to ensure anonymous authentication between clients and automated parking systems, as well as user location privacy preservation. The proposed scheme enhances automated parking services by assisting users in securely connecting to their AVs and securing all transcripts without compromising clients' privacy. Additionally, a judge is responsible for tracing anonymous clients to prevent misdeeds during vehicle pickup or relocating AVs to assist authorities in locating missing AVs with the permission of their owners. Moreover, that judge has authority to retrieve the parking lot, in which some AV is parked for its owner if his/her smartphone was stolen or lost. However, the paper needs to

develop strong intrusion detection techniques and strengthen the AV's control systems to secure the vehicle against security threats and attacks.

On the other hand, the researchers in [66] decided to tackle the "Double-Reservation Attack" while maintaining the user's privacy. Their proposed system allows the client to reserve only a single parking space at a time and prohibits one user from making more than one booking and hold multiple parking spots simultaneously. Moreover, they aimed to ensure pseudonymity and unlinkability, that is a painless but effective technique to protect privacy of users. Pseudonymity means that the automated parking system will not have knowledge of the user's unique id that creates a certain booking/parking request. That is except for the registration phase, during which a client have to divulge his/her real id to the automated parking system to validate himself/herself as a legitimate user. Unlinkability means that the automated parking system would not be able to match up a client's two parking bookings, even when knowing the credentials of these tow sessions. Besides, they aimed to ensure geo-indistinguishability by using a mechanism to obfuscate the tracing data of the users to protect him/her from attacks that analyze location statistics in the automated parking system. And finally, they sought to study the system's efficiency, as well as communication cost and computational performance.

The problem of parking lot payment is also being examined. The users must pay to park their cars using the smart payment system [67]. In the beginning, the currency is collected using cash counters, but they are difficult to maintain. After that, a variety of methods are employed to collect the money. Payment is made using the Automated Vehicle Identification (AVI) tag, based on RFID technology. RFID and mobile devices are contactless technologies, whereas smart cards, debit cards, and credit cards are contact methods.

Additionally, there is the problem of fleet management in AVs. Often, a company would have more than one vehicle running around the city for various errands. They all have to park somewhere, not necessarily all in the company's same parking lot. In that case, a plurality of parking spaces is needed to be assigned, keeping in mind the current location of the vehicles and their tasks for the next day. Therefore, a fleet parking system is being investigated by researchers, such as in [68], where they developed an algorithm to produce a total cost function and cost value to the decision-maker to help choose the best fleet parking scenarios.

On the other hand, the security of ultrasonic sensors heavily involved in the parking process was investigated in [69]. These sensors detect hurdles by releasing ultrasounds and examining their reflections. Some attackers may exploit their operation to introduce

***Table 2.4*:** Security & Privacy Attacks On Parking Applications.

Ref. No.	Threat	Proposed Method	Critique
[63]	Vehicle hacking or theft	Two-factor authentication, OTP and a smartphone	Lacks strong intrusion prevention techniques and needs to strengthen the AV's control systems.
[66]	Double Reservation Attack	One parking spot Per user, pseudonymity and unlinkability and geo-indistinguishabili by data obfuscation	Doesn't account for theft, sabotage or slandering.
[67]	Fraudulent payment	Smart payment system using AVI and RFID	Contactless payments can be hackable without any physical trace.
[69]	Spoofing and jamming the ultrasonic sensors	PSA and MSCC	Susceptible to wide-band jamming attacks and attackers collusion.

spoofing or jamming attacks, thereby causing the AV to stop when it should be moving or the other way around, which may lead to collisions. Therefore, they proposed two protection methods. The first one is a single-sensor-based, which provides physical shift authenticates (PSA). The second method uses multiple sensor consistency check (MSCC), which verifies various signals on the system level. They tested their work on the autopilot system of a Tesla Model S car.

The following sections will investigate further security attacks on the system and physical levels. But in short, the previous ideas are just a sample of how one application may have numerous ideas being developed, and hence, multiple privacy and security issues will arise accordingly, which will need to be handled in more innovative ways. Systematic design strategies on different levels are needed to enhance the overall systems' security and reliability against future threats. Striking a balance between location privacy and the optimal utilization of parking lots is a crucial problem. In Table 2.4 we show a brief digest of the aforementioned ideas and our corresponding comments.

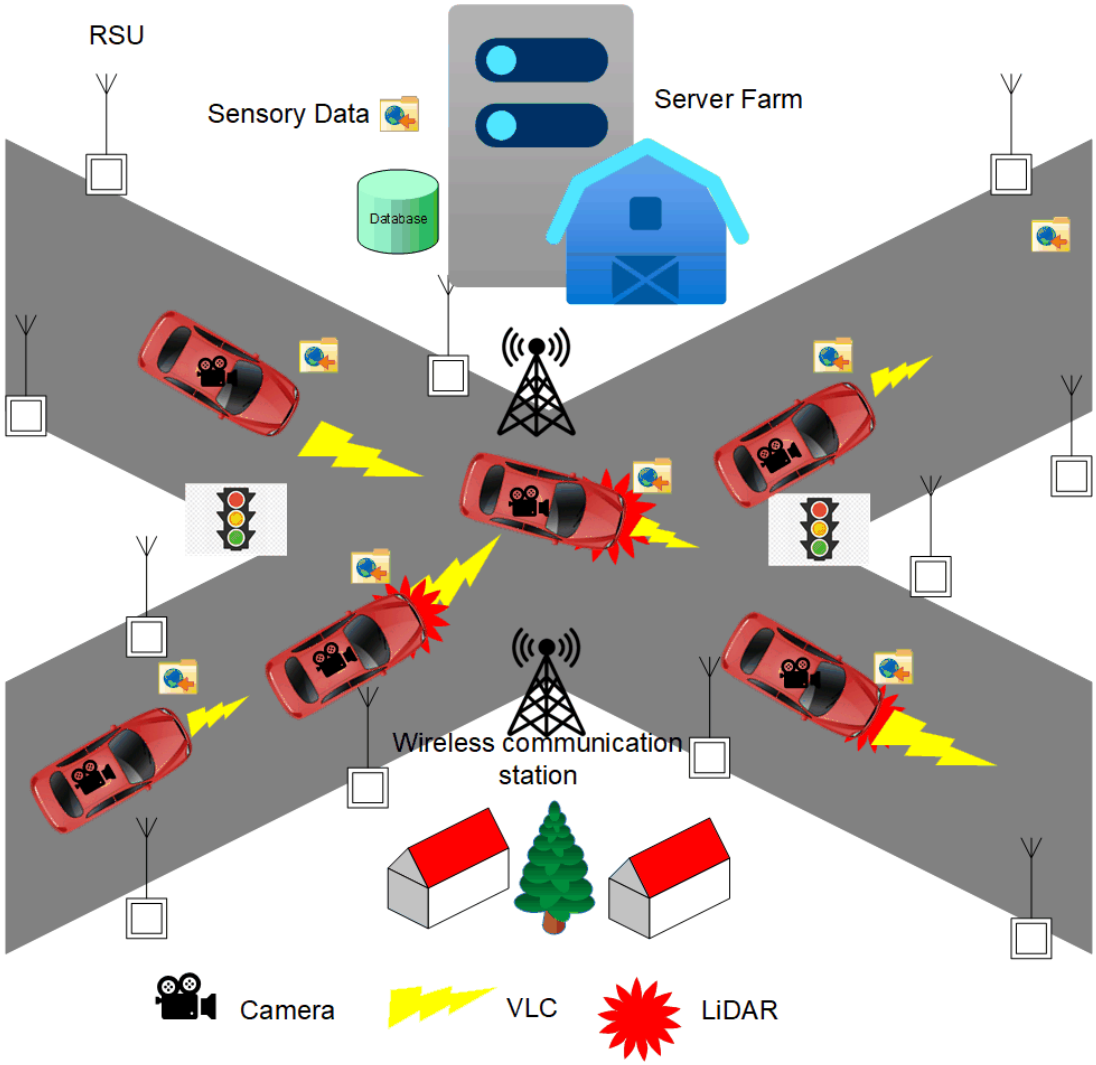

***Figure 2.6*:** Architecture of IoV system.

Internet of Vehicles

Another application worth investigating is using the vehicle itself equipped with all sorts of sensors as a sensor platform, which collects information from the environments and neighboring vehicles and feeds this information to some system to assist in various smart city data repositories such as traffic management and pollution control. This so-called Vehicle Grid fundamentally evolved into an Internet of Things (IoT), which is conveniently called the Internet of Vehicles (IoV).

As shown in Figure in 2.6, IoV incorporates many so-called "things", for example:

- *Vehicle's beacons.* Alarming devices which monitor the AV state; such as location,

internal parameters, potential hazards, etc.

- *Driver's messages.* Such as social media posts and other crowdsourced info.
- *Internal cockpit sensors.* Such as the driver's state of tone of voice, alertness, seat position, health, and other propriety sensors such as Ford heart Monitor, etc.
- *Internal automotive sensors and actuators.* Such as accelerator, steering wheel, brakes, etc.
- *External sensors.* Such as LiDARs, cameras and GPS, etc.

The differences between IoV and other IoTs are the following characteristics: I. Sensors are mobile, which may cause a wireless communication bottleneck while guaranteeing motion privacy. II. Some information will be used in safety-critical applications, which requires small latency. Moreover, IoV doesn't transfer data to the Internet using the internet connection only. Additionally, It utilizes vehicle-to-vehicle communications to complement onboard sensor data and bring forth safe and orderly navigation. However, such continued information collection may create privacy and security violations, which need to be addressed. More specifically, researchers should focus on guaranteeing location privacy and offer privacy-keeping methods to anonymously upload sensor data from AVs.

In [70], the authors studied the IoV system. An attacker may use the size of the database to determine whether a certain targeted user is included. As a result, the size of the database or the total number of users shouldn't be publicized. To ensure the confidentiality of the shares, data owners would establish an individual Transport Layer Security (TLS) connection to each aggregator. The TLS connection is designed to be long-lived in order to compensate for connection initialization costs. While aggregation delegates may attempt to conspire, it was considered in this scheme that at the minimum one truthful aggregate delegate wouldn't conspire. Additionally, data contributors can attempt to conspire with aggregators; thus, they presume that at minimum two trustful data contributors do not conspire with aggregators. The more trustworthy data contributors there are, the greater privacy protections for data contributors. Aggregation servers are anticipated to be online and continually available, and they did not account for DoS attacks, in which data owners' responses are unable to be transmitted. Additionally, the paper assumes aggregators do not corrupt data.

In [71] the researchers provided an authentication technique to secure AV users' privacy. Their proposed system addressed previous work [72] shortcomings in terms of location

spoofing, offline identity guessing attacks, and reply attacks. Their work presented a defense against impersonation and DoS attacks using a lightweight encryption and hashing scheme. In [73], the authors design a secure authenticated key management protocol for IoV (AKMIoV) based on fog computing. They use road units equipped with fog computing platforms and cloud servers to provide secure communications for the vehicles using secure session keys among all these parties. They performed a security analysis of their system using the "Real-Or-Random (ROR)" model, as well as the Automated Validation of Internet Security Protocols and Applications (AVISPA) model. They showed that this new method supersedes comparable techniques in terms of enhanced computation cost, network throughput, packet loss rate and end-to-end delay.

Fog computing was also used in the work of [74]. The paper presents a model called F-IoV, which stands for fog computing supported IoV. This model aims to effectively manage networked resources in the IoV, utilizing the roadside infrastructure. In addition, they proposed a hierarchical privacy-preserved pseudonym (P3) scheme, which provides a context-aware pseudonym changing game. Also, their analysis showed effectively enhanced location privacy with reduced pseudonym management and communication overhead.

The use of blockchain technology was also investigated in the field of IoV. In [75], the authors introduced a blockchain-enabled IoV (BIoV) to ensure the security and traceability of the shared data. Their focus was to defend against voting collusion between candidate data miners by securing the selection process by a reputation-based voting scheme that examines historical interactions and recommendations from other vehicles. In addition, the paper introduced a block verification scheme to defend against internal collusion among active miners. This verification is audited by standby miners, who will be incentivized to participate using a contract theory model.

In [76], the authors took a new direction and investigated the problem of digital forensics investigations in IoV. In AV, this problem becomes more difficult due to AVs' distributed and dynamic nature in an IoV; hence collecting and analyzing evidence may be more difficult. The authors presented the TrustIoV framework to collect and store trustworthy evidence from the decentralized infrastructure of IoV while maintaining the integrity of data and its provenance with minimal overhead.

In summary, unlike some of the previous applications, in the context of AVs, the field of IoV is relatively new. Hence it is ripe with an open area for research and development [77]. Security requirements should be balanced with energy efficiency, communication overhead, and safety requirements [78]. In addition, ongoing research should focus on migrating from

***Table 2.5*:** Security & Privacy Attacks On IoV Applications.

Ref. No.	Threat	Proposed Method	Critique
[70]	Compromising user privacy	TLS	Doesn't count for DoS attacks and data corruption.
[71]	Location spoofing, identity guessing attacks and reply attacks	lightweight encryption and hashing	They didn't account for intrusion. Storage cost is quite considerable. They didn't consider communication overhead.
[73]	Attacks on user authentication	AKMIoV and fog computing	Entails more setup cost to equip RSUs with powerful computing devices and assumes there would be continuous communication channles between all parties. Also, scalability and management becomes an issue.
[74]	Compromising location privacy	Fog computing supported IoV, P3	Entails more setup cost to equip RSUs with powerful computing devices. They didn't consider intrusion attacks on the RSUs. Scalabity is also an issue with sparse number of AVs.
[75]	Voting collusion	Blockchain- enabled IoV	Susceptible to fraudulent submissions which may compromise voting process. Also, the heave computation overhead.

traditional operating systems models to new middleware platforms, which could enable proper and secure handling, processing, and analytics of data generated in IoV, which may grow to the level of so-called big data [79]. A synopsis of the work discussed here and our related critique is shown in Table 2.5.

2.3 Operating System Level

Due to the continual progress and enhancement in deep learning (DL) technology, AVs have achieved remarkable advances in recent years. On several standard autonomous driving systems, X86-based software performs sophisticated functions and contributes significantly to driving safety. However, software vulnerabilities in autonomous driving might compromise vehicle components and systems, which impair the AVs' performance.

In [80], the authors studied malware attacks on AV systems. They investigated an attack that exploits the onboard device (OBD) and thereafter targets electronic control units (ECUs) and its connected Controller Area Network (CAN) bus. They also exploit mobile apps to access the DSRC. The target of these attacks can be any of the AV system's various software components, such as OBD ports, software Over-The-Air (OTA) updates, ECU firmware, etc. The paper also discussed different approaches to defend against these attacks, such as offering OTA updates, but these techniques may also open the system to a new attack surface; therefore, these updates need a great deal of work to protect it against tampering and/or cloning. In [81], the authors focused on remote OTA updates and their security issues. They also studied various scenarios and regulations of road safety in different nations. Some of these security techniques relied on cryptographic authentication and encryption keys such as [82–85] others use blockchain such as [86].

Other techniques to defend against malware attacks incorporate a much more capable cloud-based protection. In [87], the authors present a cloud-assisted vehicle malware defense framework. The paper proposed a lightweight malware defense mechanism that resides on the vehicle. At the same time, the heavy and much-advanced processing is done over a cloud with an up-to-date malware information database. However, adding a continuous communication link between the vehicle and the cloud might be a bottleneck that could open the system to a whole new network-level breed of attacks, which we will delve into with more details in the following section.

Additionally, numerous researchers have developed static and dynamic analysis techniques to uncover malware on embedded system components based on the X86 architecture. Static methods examine an input program from a semantics and syntax perspective without executing the software. These techniques offer the advantages of rapid detection and a small percentage of false positives. Nonetheless, a typical limitation of static analysis techniques is their impotence to detect malware that employ sophisticated mechanisms like cryptography, compression, packing and obfuscation.

Dynamic analysis-based techniques can resolve the issue by executing the subject program and observing process creation, memory access, file writing/reading, network utilization and system calls during the runtime of the program. Nevertheless, the dynamic analysis method is resource-intensive, and parallel processing is inefficient in actual applications.

In [88], the authors presented a hybrid malware detection technique that combines machine learning with fusion features from three distinct levels of static analysis. The purpose is augment static analysis's low accuracy and to address dynamic analysis's high resource cost. They developed a model for feature extraction depending on the level of operation. Additionally, to improve the accuracy of static analysis, the authors built a set of tools to unpack codes with various packing methods. Also, they present a novel model using the control flow graph and information entropy for extracting OpCode features. Their technique was established on Extreme Gradient Boosting (XGBoost). Which in comparison with other popular classification algorithms, achieves a greater accuracy in recognition.

On the other hand, despite their enormous potential, AVs supported by DL technology continue to confront several system-level security concerns. The authors of [89] focused on traffic sign recognition system and they leveraged particle swarm optimization to undertake attacks that target their deep learning algorithms. The paper first exploits the "poisoning attack with particle swarm optimization" (PAPSO) attack, that targets the deep learning algorithms during the training phase. During which malicius samples are implanted in the input data by the attacker, reducing the traffic sign recognition system's classification accuracy. Additionally, the authors investigate the "evasion attack with particle swarm optimization" (EAPSO), that target the deep learning algorithms' interference process. Attackers introduce certain barely noticeable changes to the selected test sample, resulting in misclassification. Table 2.6 depicts an abridgment of the literature shown in this subsection as well as our critique to them.

2.4 Network Level

Vehicular networks (VNs) are gaining huge attention as a research area due to the importance of aiding traffic management and providing road safety. Nowadays, many communication interfaces are integrated in Vehicles, this demands installing processing hardware, storage elements and larger power sources. In addition, to aid with traffic management system, road side unites are deployed, to communicate with vehicles wirelessly.

These systems can support taking preventative safety actions for the sake of vehicle

***Table 2.6*:** Security & Privacy Attacks On The Operating System Level.

Ref. No.	Threat	Proposed Method	Critique
[80]	Malware	OTA updates	Susceptible to tampering and cloning.
[81]	Attacks on remote OTA updates	Encryption keys [82–85] or blockchain [86]	Each technique has it's own performance cost in terms of computation overhead, communication cost and power consumption.
[87]	Malware	Cloud-based protection	Requires a continuous communication link between the vehicle and the cloud (a bottleneck for Network attacks).
[88]	Malware	Machine learning with fusion features	Requires huge computations, and doesn't support dynamic analysis of threats.

commuters and drivers, as well as aid traffic authorities. Several methods have been proposed for discussing the security and privacy issues for VNs and vehicular cloud computing (VCC).

It is essential to fulfill the security requirements in a vehicular network, which we explain in more detail as follows:

- *Availability.* Means that the system can withstand malicious or erroneous circumstances and still ensures it's functionality [90]. That's why this requirement is susceptible to wider attack types in comparison to other security requirements. [91] describes some countermeasures such as encryption techniques and trust mechanisms to defend against some of these attacks.

- *Confidentiality.* Means that only a specified/delegated user can access his/her data that he has legal authority/rights to do so. Others who don't, cannot access said data.

- *Authentication.* Means that only pre-identified and authorized users can access the network and pass messages. Other malevolent parities or intruders are denied from access [92].

- *Data Integrity.* Means that we ensure the correctness of the messages being passed from unintended alterations or modifications while being passes throughout the net-

work. There are some techniques to ensure data integrity such cryptography revocation and public key encryption [91].

- *Nonrepudiation.* Means that in case of a dispute, both communicating parties: the sender and the recepient cannot repudiate their action [93, 94].

A survey of AV communication layers and its security challenges is presented in [95]. The paper shows how conventional vehicular ad hoc network (VANETS) has metamorphosed to a contemporary paradigm coined the Internet of Vehicles (IoV) - aligned with the advancement in Internet of Things (IoT) systems and will soon evolve into the Internet of Autonomous Vehicles (IoAV).

Table 2.7 shows an illustration of different attacks on these communication aspects of the VN and exemplary countermeasures to tackle some of these attacks [96–99].

In [100], the authors studied the effects of security vulnerabilities and risks associated with deploying VANET communication and tampering of automated sensing and control of CACC, which describes a stream of vehicle connected together and cooperating together. The authors showed how an attacker can compromise CACC systems. They used an open-source software “Vehicular Network Open Simulator" (VENTOS), to implement a CACC car-following model. The model was utilizing IEEE 802.11p to simulate one vehicle look-ahead communication in VENTOS. Hence, they were able to provide a quantitative analysis of a variety of what-if cases that describe security onslaughts on the CACC system. The paper also discussed some important security consideration that needs to be contemplated in design to warrant the system’s safety by performing a detailed packet-level analysis.

Another major issue that targets traffic systems is Sybil attacks, during which an attacker could impersonate a number of cars that pass on an untruthful incident down the road. Typically a traffic system needs information about any events to alert vehicle operators of unforeseen dangers on the roads to provide safe driving.

The authors in [101] proposed a security model to protect VNs based on Integrated Circuit Metric technology (ICMetrics). This ICMetrics is utilized to provide a robust authentication scheme based on distinctive features extracted from vehicle sensors and its behaviour. In particular, they studied a simulated VANET and examined the resulting trace file, looking for infrared sensors’ bias values. Thereby their system can identify abnormal behavior that could mount to be a malicious attack.

***Table 2.7*:** Security Attacks On Vehicular Networks and Their Countermeasures.

Service	Attacks	Countermeasures
Availability	Malware	Digital Signature of Software, Reliable Hardware
	Spamming	Digital Signature of Software, Reliable Hardware
	Greedy Behavior	Intrusion Detection Systems
	Blackhole and Grayhole	Digital Signature of Software, Reliable Hardware
	Broadcast tampering	Cryptographic Primitives
	Jamming	Frequency Hopping, Direct Sequence Spread Spectrum
	Denial of service	Signature Based Authentication
Confidentiality	Man In the Middle	Digital Certificates
	Traffic Analysis	Encryption Techniques
	Social	Digital Signature
	Eavesdropping	Physical Layer security Protocols
Authenticity	Sybil	Central Validation Authority, location and Position Verification
	Tunneling	Digital Signature of Software, Reliable Hardware
	GPS Spoofing	Signature authentication, trusted positioning systems, bit commitment
	Free-riding	Strong Authentication
	Key and/or Certificate Replication	Certified Keys, Validate Certificates in Real Time
Integrity	Masquerading	Digital Signature of Software, Reliable Hardware
	Reply	Digital Signature
	Illusion	Digital Signature of Software, Plausible Validation Network
	Message Tampering	Zero-Knowledge Schemes
Non-Repudiation	Repudiation	Identity Based Signature, Digital Certificates

On the other hand, the researchers in [102] studied intrusion attacks and their countermeasures in AVs. They classified VN into two categories: Intra-vehicle networks and Inter-vehicle networks. The first category, networks that operate on the "intra-vehicle" level, connect cars to the outside world, including all external devices, usually through wireless interfaces such as Wi-Fi, GSM, LTE, etc. ECU fabricated in the car supports these com-

munication interfaces using technologies that support different types of sub-networks. The most used sub-networks are FlexRay, CAN, Local Interconnect Networks (LIN) and Media Oriented Systems Transport (MOST) [103]. In the intra-vehicle networks, some of the most common intrusion attacks targeted web browsers on the cars system, the telematics units, or the broadcast messages [104, 105]. In order to tackle these attacks, the authors suggested the use of cryptographic techniques as in [106, 107], secure hardware implementations for the ECU such as [108], the infrastructure of the AV as in [109], and finally employing mechanisms for anomaly detection as in [110–112].

On the other hand, Inter-vehicle networks are famously known as VANET, where communication happens between the vehicles (V2V). Here the attacks can be launched from inside or outside of the network, and they could be active or passive attacks. They may cause a denial of service, inject bogus information, steal the identity of some user or part of his information like his speed or position, and last but not least, racketeering and digital piracy. More details about these attacks and their countermeasures can be found in [113–115].

2.5 Physical Level

AVs are equipped with various sensors that enable them to monitor the surrounding environments and navigate safely. One of the most direct and upfront physical level security threats are attacks against these sensors. Attackers may produce incorrect information messages or ultimately hinder sensor operation to disrupt autonomous driving without accessing the AV's driving system. There are many attack surface vectors according to the sensor type [116]. For instance, AVs are commonly fitted out with numerous cameras with a variety of lenses for object detection and tracking. Here, attackers plant fake traffic signs or lights, or even some kind of objects, to spoof AVs and drive them into wrong decisions [117]. The attacker may also use a very bright laser beam to blind the camera, thus preventing it from taking any images [116, 117]. In addition to that, AVs use global navigation satellite systems and inertial navigation system sensors to learn their location in real-time. The attacker may use noise signals to interfere with the sensor's receivers to jam it. Furthermore, they may spoof location messages and tamper with them [116, 117].

On a related issue, the continued expansion of electronic device cloning poses a severe threat to any essential infrastructure that relies on the Internet, such as AVs, because cloned devices might send secret data and create security issues. Also, cloned devices could be untrustworthy since they might be built using substandard materials and may have numerous

flaws due to insufficient testing. Thus, it is critical to secure these electronic devices against cloning. Preventing a device from being cloned is as simple as avoiding copying the firmware. In the absence of the correct firmware, the machine would not operate in the same manner as the original. As conventional cloning protection solutions, such as: integrity checking and firmware encryption, are prohibitively expensive in development costs. Also, during execution, they consume a lot of resources on oftenly resource-constrained devices. That is why, more than ever, we urgently need low-cost solutions.

The authors of [118] examined two types of security attacks. In the first level (Level-I), a microcontroller serves as an interface device programmed using a serial wire debug (SWD) interface's driver and the Universal Asynchronous Receiver-Transmitter (UART) module. The interface communicates with the subject device through the SWD and controls its reset and power connections. While the subject device completes cyclic redundancy check, the interface device reads the SRAM, controls the subject's d power, and resets if needed. A script (implemented in Python) runs on the computer communicates with the interface via UART, and snapshots from the SRAM of the interface device can be transmitted to the laptop for firmware extraction. While this Level-I protection can be disabled, the flash memory is erased, thus hindering the extraction of the firmware by an attacker. The second level of protection (Level-II) considers attacks that target the microcontroller invasively. Following decapsulation, memory protection bits are reprogrammed by using some specific ultraviolet light [93]. After reprogramming, the Level-I attack mentioned above can be launched and the firmware can be extracted from the victim. Level-II locks the debug interface completely and irreversibly, allowing just processor's cores access to the flash memory.

Additionally, the authors introduced a new firmware obfuscation technique that has a low cost for detecting cloned systems successfully. Obfuscation of the firmware is accomplished by reordering a number of chosen instructions. The native instruction flow is disrupted by an efficient technique to obfuscate the firmware's original execution sequence. Their approach begins with a single instruction and searches for a collection of swappable instructions to ensure that no faults are encountered. Their approach will not stop an attacker from stealing the firmware, in lieu making it operational ultimately as well as providing reasonable cloning protection. Additionally, they built a mechanism for tamper resistance. Specifically, they explain how it is impractical to rebuild the original firmware by an attacker, taken into account the available computing resources, making their mechanism a good candidate in securing "cyber-physical systems (CPS)" and IoT applications such as AVs.

Another interesting piece of work is [119], where the authors presented a six-step model

that utilizes System-Theoretic Process Analysis to ensure both the safety of the vehicle and its passengers as well as the security requirements. They claimed that their approach is compliant SAE J3016, SAE J3061, and ISO 26262 all of which are international standards.

In [108] the authors focused on securing vehcile ECUs and their communications. They proposed a holistic hardware security model (HSM) and offered details on a prototype implementation. Their model aim is to act as a trusted security anchor which can generate, store, and process information protected from malicious software. It should also resist hardware tampering attacks and support cryptographic hardware operations all of which presented in a highly optimized special hardware prototype.

The authors in [120] presented a survey and classification of intrusion attacks on vehicles and their detection methods. They studied intrusion detection systems (IDS) in different types of cyber-physical systems such as AVs, UAV and autonomous ships. They also investigated on-board physical attacks and their countermeasures as well as remote and network-based intrusion attacks and also their mitigation strategies along with their advantages and disadvantages. Their eventual goal was to shed some light on the shortcoming of some of the current literature and show directions for future research.

In Table 2.8 we summarize the literature presented in this section and mention some brief comments about them.

2.6 Related Work

There has been a great deal of literature surveying many security aspects of AVs. Some earlier work only focused on the network part, such as [121–123]. Others focused on one of the security issues related to one of the applications discussed in Section 2.2. For example, the authors in [70] investigated numerous of applications that utilize V2I and V2V platforms and connections. They also describe location privacy issues related to users' mobility. They showed the necessity of building openly and publicly accessible, "smart city" data repositories available for all researchers as well as offer privacy-keeping schemes to enable vehicles to upload urban sensor data anonymously. In addition, the authors in [124] studied the viability of guaranteeing preserving privacy along with safety and integrity in the new hyper-connected vehicular platforms. To prove that point, they studied a scenario where vehicles upload their map updates while keeping their privacy as well as the integrity of their messages.

***Table 2.8*:** Security & Privacy Attacks On The Physical Layer.

Ref. No.	Threat	Proposed Method	Critique
[118]	Level-I and Level-II attacks on the microcontroller to extract firmware	Firmware obfuscation	Will not prevent stealing hardware, it's hard but not impossible for an attacker with sufficient dynamic analysis resources to reverse engineer the code.
[119]	System Failures and compromising AV cyber-security lifecycle.	STPA method and the Six-Step Model	System is too complicated and Needs more refinement to be applicable. Also, need to consider scalabity and collaborative scenarios.
[108]	Attacks on the ECU	Holistic HSM	Although the model might be tailored to guarantee in-vehicle specific requirements, this might hinder its scalability and wide, general purpose industrial applicability.
[120]	Intrusion attacks	Various IDS according to vehicle type and access methods	Each technique has its pros and cons. As such, some might need physical audit features, others may require more generalized hardware design considerations.

That said, the contribution of this chapter is to survey AV privacy issues and security challenges from a multi-layer perspective. We tackled these problems within each layer and showed some state-of-the-art work to solve them. We hope that this big picture view would help develop solutions that would address several issues simultaneously and support the spread of AV technologies.

2.7 Summary

In this chapter, we surveyed AV's security and privacy issues in a layer-based model. We vision the AVs system as consisting of four layers, and we investigated some of the attacks on each layer and some of the most promising corresponding countermeasures. Our goal is to help researchers from a different backgrounds identify where they can contribute in this

vivid field.

Chapter 3

A PROPOSED SOFTWARE PROTECTION MECHANISM FOR AUTONOMOUS VEHICULAR CLOUD COMPUTING

3.1 Introduction

AVs are rapidly becoming more computationally powerful. AVs are equipped with advanced processing and storage units, computer vision technologies, aided with a variety of sensing and communication capabilities to enable the vehicles to autonomously drive themselves without any human intervention [2]. Such huge computing power allowed for a myriad of applications and services to run along with the vehicle's autonomous navigation system. Their huge sensing and data collection capabilities can be used to support traffic management systems or environmental monitoring and pollution control systems using a model called IoV [6]. Nevertheless, sometimes this power is under-utilized, hence comes the need for autonomous vehicles' cloud computing [12]. Consequently, remote code execution on a physically unreachable shared platforms is inherently challenging in terms of security predicaments. When it comes to embedded systems, such as the ones supporting AVs, theses issues become more challenging due to the limited capabilities of these new systems, in terms of processing power, memory and storage.

In addition to that, in the case of AVs, the moving and continued change of location of the car, along with the different communication interfaces they may use, and the different perpetrators they may encounter, may open AVs to an unknown pool of attack scenarios [13], such as intrusion, Denial of Service (DoS), information leakage via side-channels and reverse engineering, which we will focus on.

Traditional cryptographic techniques such as digital signatures, certificates or trusted platforms, may not be adequate. Because, the decryption is done remotely, therefore, attackers may observe the actual running process and expose the code [14]. That's why we propose closed security, by increasing the logical complexity of programs to make it more unintelligible for attackers, this process is called obfuscation.

This chapter presents a study of improvement and extension to the dynamic obfuscation via compilation system to make it suitable for work on AVCCs. In particular, this work has the following contributions:

- We present an enhanced software-based security technique to protect remote code execution on AVCC platforms against timing side-channel attacks;
- We show the utility of these various new improvements within a simulated embedded system environment running a set of standard benchmarks and study the performance cost of our method.

The remainder of this chapter is organized as follows. Section 3.2, lays down some basic concepts behind the ideas we utilized in our proposed technique and provides a summary of the literature relevant to our study, as well as their efficacy in contrast to our suggested approach. Furthermore, we describe our network and threat models in Section 3.3. In Section 3.4, we discuss the implementation of our system. The experiments and the analysis of their results are depicted in Section 3.5. Finally, in Section 3.6, we summarize the chapter and suggest directions for future work.

3.2 Background Information and Related Work

Before moving forward with laying out our proposed system, we have to build a firm background about the information that we used as well as a knowledge base of the related literature and how it connects to our work.

3.2.1 Preliminaries

In this section we delve into more details about the most important constructs that we used in our system.

The LLVM Compiler

A compiler is a tool that converts high-level language programs into machine-level instructions. Compilers learn a lot of program's semantic information, which allows them to optimize output programs' performance.

LLVM [125] is a robust, open-source compilation infrastructure for building compilers. Historically, LLVM began as a research project known as (Low-Level Virtual Machine) developed by Vikram Adve and Chris Lattner at the University of Illinois [125, 126] to provide a static/dynamic compiler applicable for an arbitrary wide range of programming languages. Now LLVM is the official compiler for Apple products, including MAC OS X

and iOS. The compiler is based on the famous Static Single Assignment (SSA) form [126] significantly simplifies developing compiler optimizations.

Moreover, LLVM is an extensible compiler suite that supports customization through modules and plugins. It allows developers to create plugins to customize the functionality of the compiler. For instance, a custom LLVM pass exists to implement the Control Flow Guard (CFG) exploit mitigation and things like binary instrumentation for fuzzing/code coverage purposes. The LLVM compiler suite, functions as a backend portion of a compiler that handles machine code generation from the LLVM IR (Intermediate Representation).

Compiler suites such as the Clang C/C++ compiler and other programming languages like the Swift and Rust compilers use the LLVM project as a backend. These compilers output LLVM IR code, which is then passed to LLVM to generate compiled binaries from the LLVM IR. Any compiler targeting LLVM as a backend automatically supports code generation for any architecture supported by LLVM, such as Intel X86 or ARM. The LLVM project also includes a linker (LLD) and other valuable utilities when developing compilers.

Due to its wide adoption and modular extensible architecture, LLVM is an excellent choice when writing plugins for code obfuscation purposes. By performing obfuscation at the LLVM IR level, it is possible to develop compiler passes for code obfuscation purposes that support multiple programming languages and instruction set architectures.

Conditional Branches

Control-flow optimizations, such as branch optimizations, relax the control dependences in the program in order to facilitate parallelization and pipelining. Conditional branches are the main instructions that affect control-flow, as their outcome is generally not known until the runtime. Typically, these branches add a significant cost to the runtime of a program. As a result, conditional branches impede parallelism or pipelining attempts [127].

Historically, numerous scholars experimented with various ways to convert or resolve branches to increase program speed [128]. As a result, speculation was developed to reduce the impact of such control dependency while maintaining data correctness. Branch prediction is used in speculative execution to figure out the location of the next instruction beforehand. Then we can fetch, decode, and execute the next instruction as though the branch forecasts were consistently right [129]. The entire pipeline is flushed if a mistake is found during execution, the proper instruction is retrieved, and all operations completed are thrown away. By using predicates, which are a form of guarded instructions, the authors in [130] attempted to relieve such control dependences by using guard instructions called

predicates. An instruction's execution is determined by the evaluation result of some guard expression.

Conditional branches can be classified into the following branch types [131]:

Forward Branch Changes the control-flow to a target after the branch but, generally, in the same loop nesting level.

Backward Branch Changes the control-flow to a target before the branch but, generally, in the same loop nesting level. A backward branch can be thought of as a loop; thus, the loop optimizations can be applied to it.

Exit Branch Transfers the control out of the loop nest, which terminates one or more loops.

Consequently, there are various techniques involved in optimizing branches within a piece of code [132]. They are listed below:

1. Straightening: put the target code instead of the branch, replaces some branches with the target code to get larger basic blocks.
2. Tail merging: unify tail branches of basic blocks to a single branch, which replaces identical tails with one tail and branch from the others.
3. Branch-to-Branch optimization: replaces a branch by a simpler one, which usually occurs when a branch target is another branch.
4. If simplification: eliminating the empty or constant-valued condition arms of an if-construct; it also deletes the empty if-constructs that may be found in the automatically generated or optimized code.
5. If-conversion: converts conditional branches into predicated instructions, supported by the underlying processor architecture.

Later on, we discuss in details if-conversion optimizations. We utilized these branch optimization techniques for security through obscurity by varying their time cost to thwart probabilistic analysis of code execution.

3.2.2 Related Work

Almost all of the concepts we used in our system have been studied before in some form or another; the originality is in combining the appropriate components in the proper order to better fulfill our security needs without making the system too complicated. Some of the relevant work is discussed in this section.

Code Obfuscation

First of all, we have to state that obfuscation has been around for some time [133]. It was introduced to manage the privacy of sensitive data in cloud computing platforms [134] and for program protection as in [135]. In [136], the authors present a study of Obfuscation, and deobfuscation tools in Android platforms. They investigated automated tools such as R8, ReDex, Obfuscapk, and DeGuard. Additionally, [137, 138] also investigated some other techniques to obfuscate android apps. Some of the most famous techniques are: Debug Information Removal, Function Call Indirection, Goto Instruction Insertion, Reordering, Arithmetic Branch Insertion, Nop Insertion, and Medhods Overload. Also, there several commercial Java obfuscators such as Zelix Klassmaster, Stringer, Allatori, DashO, DexGuard, ClassGuard, and Smoke.

However, the more prevalent use of obfuscation is in hiding malware and other foist software to evade scanning or analysis. An example work of such context is of [139]. They proposed a technique for obfuscating trigger-based malware code based on some conditions at the static compiler level. This scheme allows for evading malware analysis tools. They used the LLVM compiler to transform the input program into an obfuscated binary. The system captures a conditional input trigger that starts the malware; it derives an encryption key from the input, encrypts the code, and removes the key from the generated code. Thus analyzer programs cannot easily detect the start or execution of malware code. This system generates static obfuscated code essentially for malware triggering.

On the other hand, there has been extensive research as well on deobfuscation and automated analysis tools [140–142]. For example, in [143], the authors present a technique for extracting an executable program from the packed and obfuscated binary code. They developed a hardware-assisted software for import tables reconstruction and rebuilding obfuscated API names.

That said, our proposed obfuscation system generates dynamically, and every changing obfuscated binaries, which make our is generic in that sense ad suitable for application in the realm of AVCC platforms.

Cloud Security

There have been many attempts in the literature to solve the security problem in the remote execution platforms in general such as the cloud computing. Many recent papers surveyed the latest state of art work in that endeavor, such as [144–147]. However, to the best of our knowledge, none of them has considered using dynamic compilation technology to secure remote code execution. Here, we shed some light on some of these attempts.

Twin Clouds: [148] propose securing the cloud by utilizing two clouds: a trusted private cloud (where the cloud is under the user's control) and a commodity public cloud. The private cloud is used for encrypting critical data and algorithms (setup phase). The commodity cloud is used for computing time-critical computations (trusted cloud queries) in parallel under encryption (the query phase). A user first sends his/her request to the trusted private cloud, which authenticates and encrypts the algorithm/data using a trust mechanism that is based on Yao's garbled circuits [149]. This process is based on two-party encryption to realize what is called verifiable computing [150]. That is computing the value of a function with minimal knowledge from participating parties. The system exposes the twin cloud architecture to programmers, increasing the cost of the software. Moreover, it incurs the extra cost of garbled circuit execution and communication between the clouds. Though, this work presented a practically efficient approach for secure computations as opposed to Fully Homomorphic Encryption (FHE), which aims to allow calculations on encrypted data without using additional helper information [151, 152].

Hypervisor Security: [153] discussed the issue of how to trust a hypervisor. They present root trust static and dynamic management concepts. They suggest having a third-party certificate authority that provides certificates that can be used for remote attestation of a given platform; by extending the Trusted Computing Base (TCB) as per the Orange Book [154]. The difference between Static Root Trust Management (SRTM) and Dynamic Root Trust Management (DRTM) is that the latter can start a program in an Isolated Execution Environment (IEE) at any time, not just at boot time, which is a new root for trust chained from the initial state of the machine (a clean CPU state). Hence, a client can be assured that its virtual machine is integral since it has started from a trustworthy state and has not been modified or replaced by a malicious one. The system incurs a costly start overhead due to the chained trust mechanism. Moreover, the system is still susceptible to side-channels attacks from other virtual machines. Also, a downside of the system is that the technique relies only on verifying that the hash belongs to a list of trusted hashes, but that does not necessarily guarantee that it represents a trustworthy module. The certificate authority can be deceived by a fraudulent certificate issued by a malicious insider since the system relies only on a key for security in the launch process. Moreover, after the launch process, there is no way to guarantee the integrity or privacy of our computations on the cloud during runtime. There is also the risk of sabotage attacks via buffer and memory overflow exploits.

Secure Virtual Architecture (SVA): [155] present a new compiler-based virtual instruction set for executing code on a given system, including kernel and application code. The architecture provides instructions for object-level memory safety, control-flow integrity, and type safety, allowing it to monitor all privileged operations and control physical resources. They also provide custom instructions to control memory layouts, such as allocation and explicit de-allocation instructions. Thus, this work only protects the system from sabotage attacks such as memory or buffer overflow attacks on a physical resource. However, the system is still susceptible to eavesdropping attacks, especially at the OS level. Moreover, the SVA sandboxing mechanism focuses only on the instruction set beyond the Intermediate Representation (IR) level and a code-generation phase.

That said, there are hardware-based commercial solution offered by major vendors in

the cloud market. For example there is Intel TXT [156], ARM TrustZone [157], AMD SEV [158], and Intel SGX [159] technologies. The researchers in [160] studied these technologies and compared them with each other. However, the found that each technology can offer certain security guarantees that the other technologies do not provide under some particular settings. Therefore, it is up to IT security managers to choose which hardware to adopt according to their needs.

On the other hand, our work is a protection mechanism that is totally software based, which is easier and less expensive to set-up and more applicable to a wide range of usage scenarios. Moreover, it's independent of input programming language and architecture agnostic. That's because we focus on the intermediate representation level. This makes our proposed mechanism suitable for a heterogeneous platform such as the AVCC.

Side Channels

Side Channel Attacks (SCA) have been posing a great threat against different platforms and architectures. That's why there has been a great deal of research on how to launch them and how to thwart them equally such as [161–163].

For example, the authors in [164] present a software-based side channel attack that monitors power consumption statistics. They exploited the Intel Running Average Power Limit (RAPL) interface to gain unprivileged access, which allowed them to correlate and infer which instructions being executed and distinguish hamming weights of operands and memory access operation. This breach allowed them to leak information about the control flow of running program and to leak data and cryptographic keys as well. They also presented some non-trivial mitigation techniques to this attack.

In addition, the authors in [165], examined the vulnerabilities of ARM processor's instruction set architecture (ISA) and their susceptibility to SCAs. They also, surveyed different countermeasures to thwart these attacks. However, most of these attacks targeted cryptographic algorithms in order to leak information about encryption keys, but here, we aim to protect the execution trace of the running program using obfuscation. Thereby, we mangle with the correlation between a program's runtime behavior and the input data. Hence, we make it difficult for attackers to leak information about the program and/or reverse engineer it.

On the other hand, the researchers in [166] introduced a software tool that can generate a polymorphic version of a function of a program. They used different techniques like instruction reordering, changing addresses of registers, and dummy code insertion. Nonetheless, our techniques provides more diversification to the resulting technique since we employ a technique that randomly and dynamically changes the control-flow of the program and hence its execution time. Moreover, our scheme is simple to apply and already integrated within the compiler infrastructure which makes it a relatively light-weight technique. However, some of the aforementioned ideas, can be incorporated in our system to offer more protection. These, along with other similarly integrable methods will be investigated in future work.

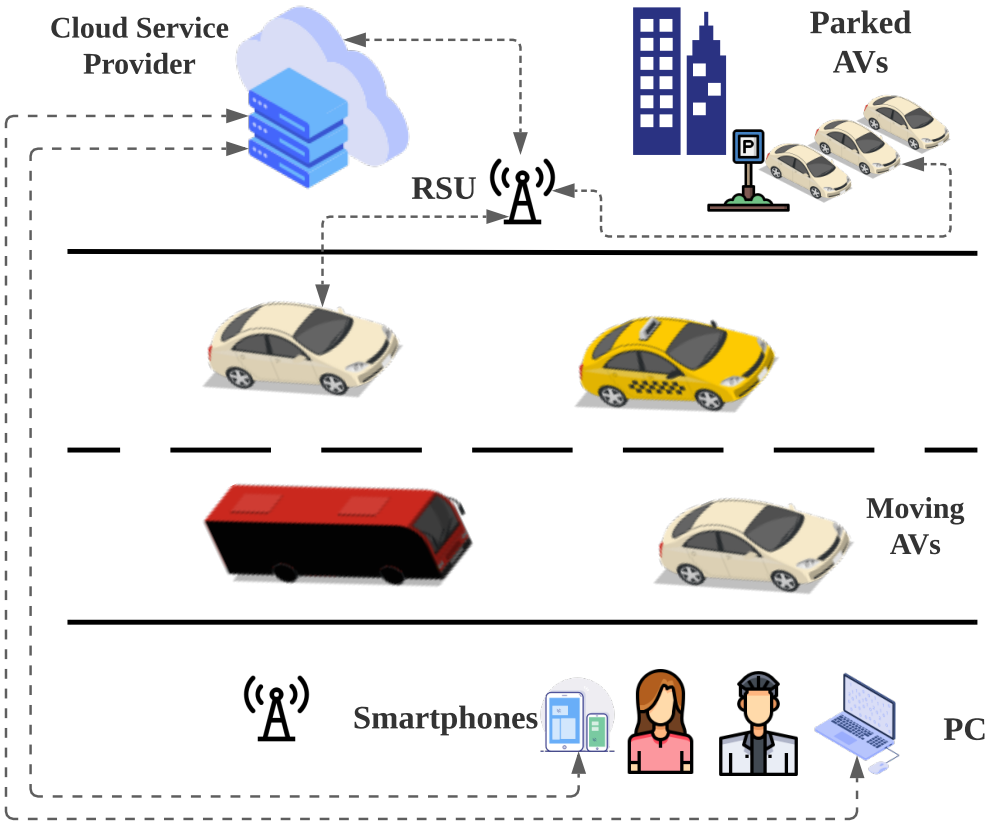

***Figure 3.1*:** The network model of an AVCC platform.

3.3 System Description

3.3.1 Network Model

Cloud computing has existed for a while now, hence the idea of utilizing AVs computing capabilities to create a similar on-demand computing platform coined Autonomous Vehicular cloud computing (AVCC). As illustrated in Figure 3.1, AVs communicate together and to the outside world using Road Side Units (RSUs) and many communication interfaces. Thereby, they can share their resources such as processing, storage, sensing and collaborate together to perform computation tasks which may need powerful processing. These AVs can be traveling along the road or parked in some place. Often, they may connect with remote servers to provide some service. The management and synchronization of such collaborative efforts and sharing of physical resources will be done by a designated AVCC controller, often one of the parked AVs, in order to balance the demand with the availability. The service provider will sell and manage the computing offerings through some kind of an interface with perspective clients who might be using their personal computer or even smartphones in any place around the world. A client should be able to execute a code or/and store data on some physically hardware selected by the AVCC controller securely and without any

governance from the client side.

3.3.2 Threat Model

Platforms that are utilize shared resources such as AVCC, often suffer from security attacks. That's because, attackers try to advantage of some loopholes to breach the system and obtain access to user data.

Here, we focus on non-invasive attacks that aim to leak information about running programs. These attacks are often easy to implement and quite scalabele and if executed on a large scale can threaten the entire platform. They are called side-channel attacks (SCA), which are based on monitoring of some physical phenomenon associated with the execution of a target program, such as it's power consumption, acoustic emission, resource access patters or running time. Then, the attacker might try to analyze this data in order to deduce some correlations about the behavior of the program being executed. Based on that, he might be able to learn which instructions were issued and back trace the execution of the program in order to reverse engineer it. His target might be stealing a copyrighted program or later tamper with it.

We assume that the attackers can be unauthorized intruders or other users of the cloud platform. We also assume that the service provider is honest but curious, meaning that they would not affect the integrity of our system and the correctness of the running program, but may try to leak information about the code in the same manner mentioned above. Either way, our proposed closed security-by-design technique should try to thwart side-channels from any adversary by obscurity and increasing logical complexity.

3.4 Proposed Technique

In this chapter, we built on an earlier implementation of obfuscated compilation [167]. Here, we used the same basic concepts and developed a system tailored to suit the embedded system's architecture supporting the AVCC platform. In this section, we will detail the components and operations of our system.

Compilers have access to a vast amount of program semantic information, which can be leveraged for security purposes. A quite powerful infrastructure for building compilers is LLVM the open source framework [125]. Using compilers allowed us to dynamically generate apparently different executable versions of any source program that are architecture dependent, yet functionally equivalent. This process is called obfuscation. This process aims to change and complicate the execution sequence of the program and perhaps its running time. Such changes if properly applied, would hinder timing side-channel attacks. There are many techniques to apply obfuscation [168, 169], but here we focus on control flow-obfuscation. In particular, these transformations target the code behavior itself, by changing control sequences in a program. There are many techniques to achieve that, such as function in-lining and outlining loop switching and unrolling or even deletion, dummy tasks, opaque predicates and branch conversion.

Here, we focus on the last technique, that is we override the normal behavior of LLVM compiler transformation passes, to implement a dynamically randomized conversion of con-

ditional branches targeting. When our system is applied to a source program, it discovers the branch conversion opportunities within each function. Then, a random *guard* is dynamically generated at compile-time as a bitmask to decide whether or not convert a branch, within each compilation phase. These phases correspond to each compilation unit in the input program and can be manipulated independently to allow for focusing on hot-spot functions in the programs, where any branch change would have greater effect on the overall execution time. Moreover, this code diversification process happens dynamically such that we can generates dissimilar versions of the same code pieces having different control-flow paths, which would eventually make it harder for attackers to leakage information about the actual code behavior.

The next step was to cross compiled the obfuscated LLVM compiler to suit the AVCC. Our target platform is based on ARM architecture, which are quite popular in embedded systems and more specifically AVs.

In the ARM architecture, the original 32-bit instruction set provides a conditional execution that allows most instructions to be predicated by one of 13 predicates based on some combination of the four condition codes set by the previous instruction. ARM's Thumb instruction set (1994) dropped conditional execution to reduce the size of instructions so they could fit in 16 bits, but its successor, Thumb-2 (2003), overcame this problem by using a special instruction which has no effect other than to supply predicates for the following four instructions. Thumb-2 introduces a logical if-then-else function that you can apply to the following instructions to make them conditional.

In our work, our randomly generated guards control which instruction to be predicated or not, depending on the bit's value (either 0 or 1). We made sure to introduce this prediction such that it does not interfere with the correctness of the compiler's optimizations or the correctness of the running program itself. That's why the best way to insert the obfuscation step was to use the compiler itself to do it.

Keeping these conditions in mind, some programs offered a wider range of predication opportunities compared to other programs. Meaning that the number of conditional branches that would be converted without threatening these conditions expectedly varies from one program to another. The reflection of such variation would be apparent in the timing results.

3.5 Experiments and Results

Software running on ARM-based platforms exist in two delivery models, the first one the more powerful general purpose platforms, running a Linux distribution (albeit specifically tailored distro for embedded platforms, such as Linaro OS). This one is more easier to deal with from a software prospective, but it obviously has more hardware requirements to run the OS and the code on top of it. The other model is a bare-metal platform, which you need to have all the program libraries and associated frameworks compiled and linked when generating the machine code. These differences needed to be taken care of when compiling our code, since it greatly affects the resulting executable file and hence its execution time. But in both cases, our obfuscation mechanism will be working essentially in the same way.

For this experiment, the host machine ran Ubuntu 20.04.1 LTS x86_64 version, and the target was a Linux-based 32bit ARM platform. As a preliminary showcase of our ideas' va-

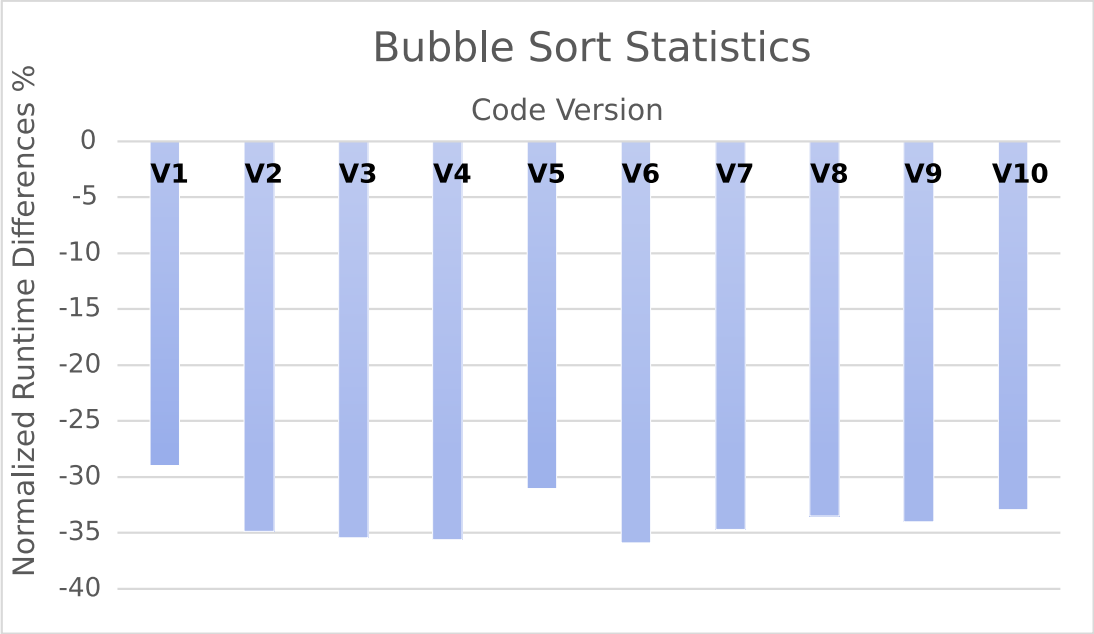

***Figure 3.2*:** Obfuscated code versions of the Bubble Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

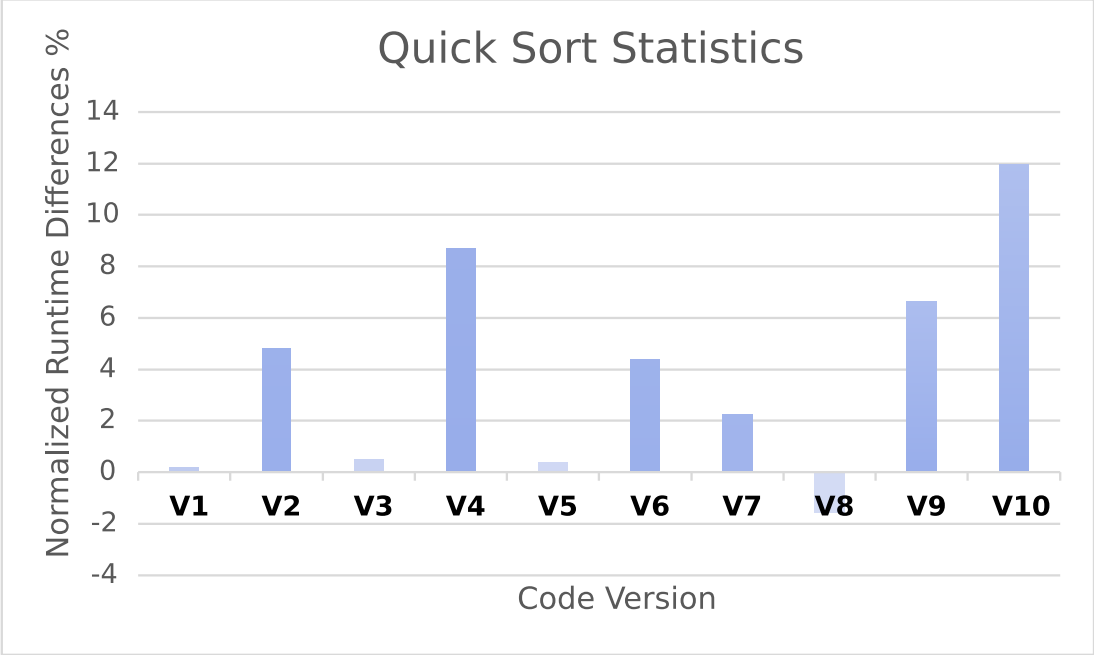

***Figure 3.3*:** Obfuscated code versions of the Quick Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

lidity, we used an ARM Emulator (QEMU) [170]. It encompasses system-level architecture as well as processor micro-architecture simulation. We plan to test our implementation on actual hardware instead of the simulator in our future work.

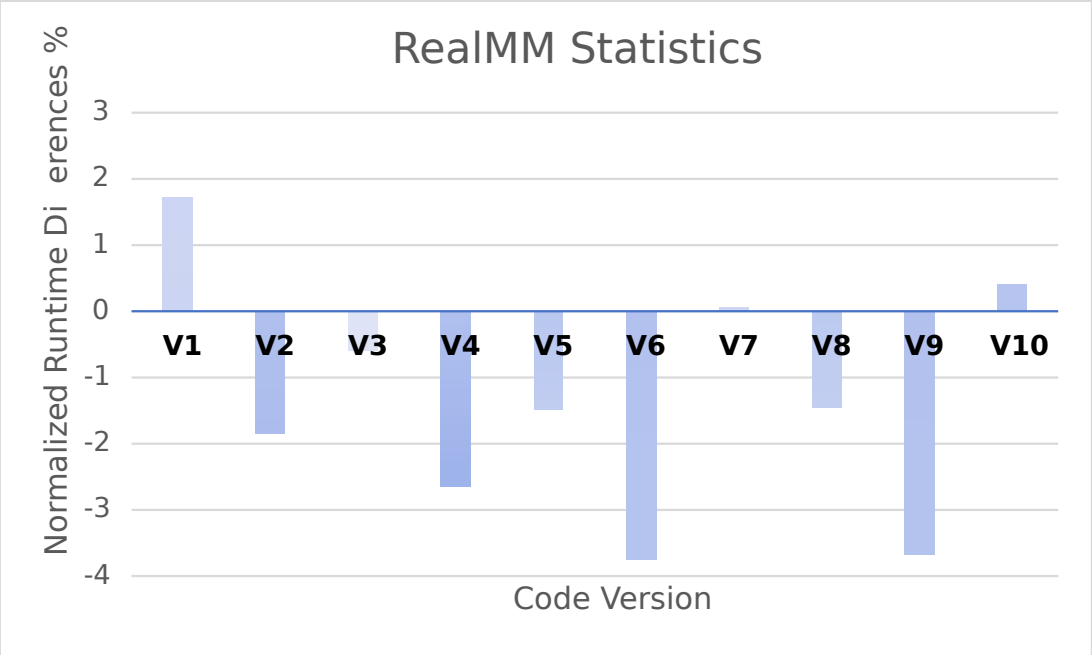

***Figure 3.4*:** Obfuscated code versions of the RealMM program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

As a simple showcase of our work, we experimented on some simple standard benchmark programs with predefined workloads. These are some very well-known algorithms implemented in the package provided by the Gem5 simulator [171].

Figure 3.2 shows the bubble sort program, where we show the normalized execution time difference percentage across the various program versions that we produced. Each of these different program versions corresponds to a different bitmask, hence a different predicated instruction series. But all the programs have the same functionality, albeit with different running times. The running times were normalized in comparison to the original unmodified code version, and the results were plotted in Figure 3.2. We can see that these normalized differences varied between -29% and -36%, which means they ran faster than the original code, with different speedups for each code version.

On the other hand, Figure 3.3 presents the normalized runtime differences for the quicksort program and shows different behavior. Here, the differences varied between -2% to 12%. The cases where it's nearly 0% would mean that no instructions were predicated, according to the bitmask. This case would happen if the branch conversion opportunities were minimal. Removing exit-branches from loops may cause large execution time overhead while removing backward and fall through branches impact on the program control-flow is difficult to predict at compile time [172]. Since the quicksort program algorithm has many recursive calls, it would be expected that it's more challenging to perform the if-conversion transformation. Hence, the opportunities for predication are limited, and therefore the time changes are also limited.

Figure 3.4 tells a different story. The normalized runtime differences for the REALMM benchmark varied between +2% and -4%, which may be limited, but interesting to see that

some modification may slow down the overall runtime. In contrast, others may speed it up with respect to the same program.

In summary, we can see from these different results that we need to produce more runtime changes adapting to various subject programs, which may lack predication opportunities. We intend to address that in our future work.

3.6 Summary

AVCC platforms are still in their very early stages of development and adoption. There are many issues to be handled with most cloud computing delivery models, but security and privacy issues take more precedence. In this chapter, we present an investigation on a proposed technique to protect software running on AVCC. Our approach extends on early work to cover code running on embedded system platforms, which is the case in hand in AVCC. Obfuscation is quite popular in implanting surreptitious software on remote embedded platforms, here we utilize the same idea for software protection. Our LLVM-based obfuscation system proved to be architecture agnostic, and we saw the promising result for simple benchmark programs. Future investigation will examine performance, latency and other aspects to assess the applicability of our work.

Chapter 4

ENHANCED OBFUSCATION FOR SOFTWARE PROTECTION IN AUTONOMOUS VEHICULAR CLOUD PLATFORMS

4.1 Introduction

In this chapter, we propose an enhancement to the technique discussed in Chapter 3, which aims to hide the actual behavior of a running program by increasing its logical complexity [173].In particular, obfuscation makes the code difficult to understand by attackers; hence they cannot leak information about it. That's why this closed technique is called security by obscurity or security by design. In particular, we disrupt the normal control-flow of a running program, by introducing a compiler-based branch instruction transformation algorithm, which is applied dynamically and randomly to the the input program. Therefore, we complicate the behaviour of the control flow of the program, which will be reflected in its running time and other physical manifestations such as power consumption, electromagnetic and sound emissions. A side-channel attacker would need consistency in these manifestation to infer correlations about a running program and hence leak information about the running program's behaviour. By using our technique we would disrupt these correlations and hence thwart side-channel attacks.

In particular, the contributions of this chapter are:

- ***first***, we enhance our obfuscation mechanism to protect programs running on AVCC platforms against information leakage via side-channel attacks which use timing analysis;
- ***second***, we address the limitations of our initial implementation in some AVCC applications, specifically in the cases where there was a small number of opportunities for branch conversions, hence it limited control-flow obfuscation transformations.;
- ***third,*** we kept the advantages of our compiler-based software system, that's being input language agnostic and platform independent, which makes our system generic and easily applicable in AVCC platforms;
- ***finally***, present an analytical study for our system and their effectiveness with regard to different input programs.

The remainder of this chapter is organized as follows. In Section 4.2, we discuss our network and threat models as they apply to our system. In Section 4.3, we talk about how we're going to put our system in place. Section 4.4 depicts the experiments and the analysis of the data. Section 4.5 shows an overview of the literature related to our work. Finally, in Section 4.6, we wrap up the chapter and make some recommendations for further research.

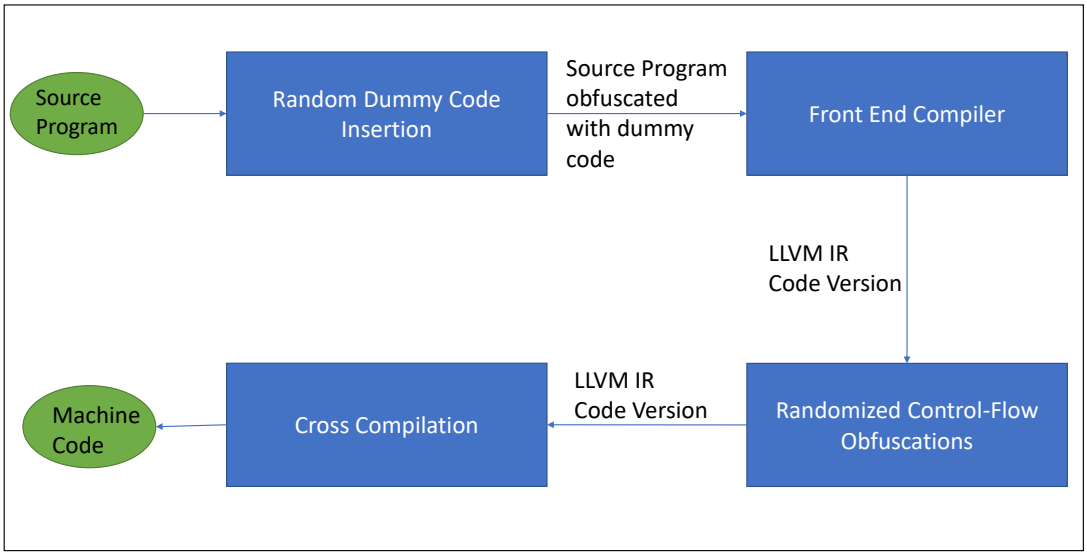

***Figure 4.1*:** Flow chart explaining the steps of producing an obfuscated code version using our proposed system.

4.2 System Description

Since this technique is basically an improvement to the one introduced in Chapter 3, we used the same network model as described in Section 3.3, and that same threat model.

4.3 Proposed Technique

In this chapter, we extend our earlier system that use obfuscated compilation, to work in the realm of AVCC [173]. In particular, We used the same idea and developed an enhancement to that system designed to better serve the ARM based architecture supporting the AVCC platform. In this section, we explain how our system was designed and how it works. We also addressed some architectural issues with the obfuscation transformations to suit the new platform.

ARM processors have a technology called predication. A predicate is a logical concept that adds a control flow decision point or an if-then-else functionality to the next group of instructions, thereby they are said to be conditional. In the original 32-bit architecture, there was a combination of four conditional codes that controlled 13 predicated instruction. Later in 1994 the Thumb instruction set was introduced and the inventors sought to eliminate this sort of conditional execution in order to reduce the size of instruction for a size of 16 bits. Then again in the modified version coined Thumb-2, they overcome the size problem by introducing special instruction which only provides predicates to the following instructions.

In our system, we used a similar concept but with a dynamically changing *guard* bit that controls the branch conversion process based on the bit's random value (either True or False), which is the basis of our control-flow obfuscation technique. We aimed to integrate

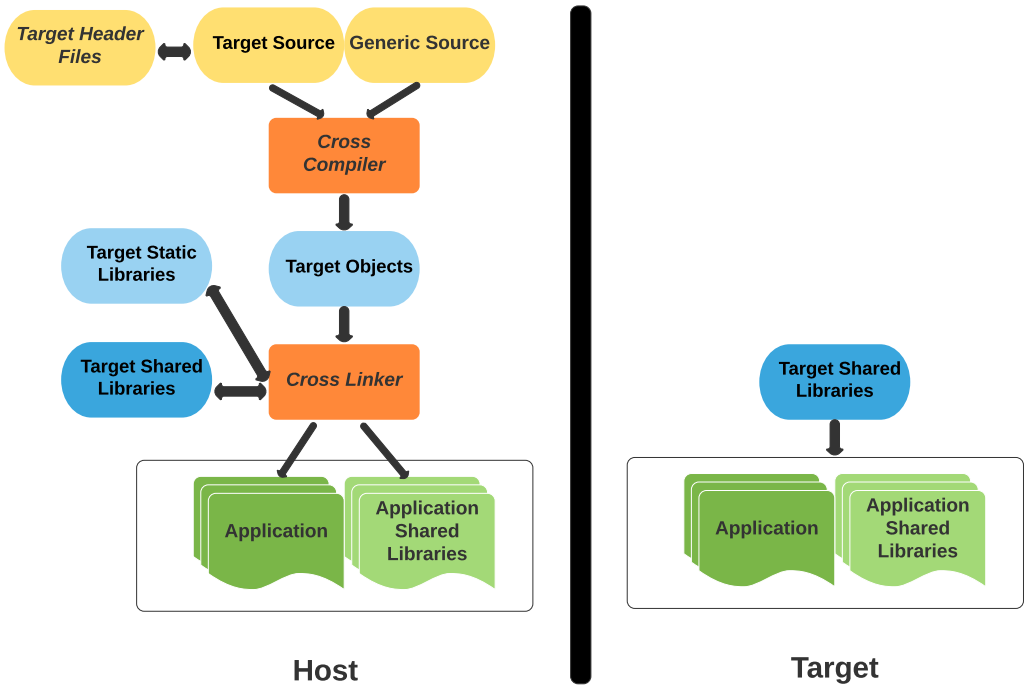

***Figure 4.2*:** LLVM cross-compilation steps.

this process within the LLVM infrastructure, without affecting its integrity or the correctness of the compiled program. Therefore, we modified LLVM compiler framework to implement a dynamic obfuscation scheme using randomized transformation of conditional branches. When a source program is fed into the compilation framework, our system first examines each code block (typically a function) looking for these branch conversion opportunities. Then, these convertible branches are flagged by a bit as a sort of a *predicate* to decide whether to transform that branch or not. This flag bit can be dynamically changed with each compilation phase in order to change the overall control flow of the program. Moreover, each flag can be modified independently which allows us to add more disruption to the input program at certain areas of the code (say hotspot function) which in turn will manifest in significantly added unpredictability of the timing behaviour of that program. This would allow us to create sort of diversified code versions for the same input program, having different control flow behaviour, while maintaining its correctness and functionality. Thereby, making it quite complicated for attackers launching timing SCA to build correlations about the behaviour of the running program, hence he will not be able to leak information about it.

Figure 4.1 shows the steps of creating diversified machine code versions from an input source program. These code version are tailored to the ARM platform target using LLVM's cross-compiler. Figure 4.2 shows how we ported our LLVM based system to suit the the embedded system architecture supporting the AVCC platform. The figure detail how the cross-compilation process works by utilizing different static and shared libraries according to the target object. Programs running on ARM-based systems can be one of two things,

bare-metal infrastructures and OS controlled platforms. In the bare-metal model, you would need to compile all program libraries and every associated framework or toolchain and link them together in the resulting executable machine code. While in the latter model, often a Linux based OS (such as Linaor OS) is the one governing the system. Therefore, you wouldn't need to compile and link every little piece of code needed in your program. But on the downside, the ARM processor itself need to be powerful enough to support the OS and other programs to be executed on it. All of these conditions should be kept in mind, since they would greatly affect the resulting code size and hence its execution time. Nevertheless, in both scenarios, our obfuscation scheme will work essentially in the same manner.

That said, during the experiments performed in [173], we noticed that some a limited number of branch conversion opportunities due to its underlying control-flow structure. It means that the number of conditional branches converted without threatening these conditions expectedly varies from one program to another. That's why, in the current implementation, we introduced an LLVM plugin (Transformation) to randomly insert junk code into the input program. This inserted code could be just random instructions or an entire dummy program running within the actual input program without compromising its functionality. The purpose of this junk code is to maximize the branch conversion opportunities in the input program by introducing new lines of code, which will further hinder reverse engineering and analysis of the compiled code.

In particular, the junk code insertion is a ModulePass, meaning that the pass gets invoked on every module (source code file) during compilation. However, LLVM supports other modules such as FunctionPass that runs on every function, and BasicBlockPass that runs on every basic block within the program, which can be further leveraged in future work.

Note that this pass begins by creating a global variable that is referenced by the inserted junk code. This is because some LLVM optimizations attempt to remove dead code from the compiled program for optimization purposes. This optimization also has the unfortunate side effect of removing the junk code that we have inserted to evade signature-based detection. If the junk code references a global variable, it is not marked as dead code and deleted.

After creating the global variable, the pass uses a loop to iterate through each function, its corresponding basic blocks, and each of the instructions within those basic blocks. It then chooses and inserts random lines of code within the input program. These instructions should be simple enough to make sure not to overly complicate the input program or significantly affect its overall performance. Nevertheless, the manifestation of these modification would be noticed in the runtime results, which serves our goal in mitigating side-channel attacks.

4.4 Experiments and Results

We set up our environment as follows. An HP Notebook model 15-DY1023DX acting as the host machine running WSL2 Ubuntu 20.04.1 LTS x86_64 version, and the target machine was a Linux-based 32bit ARM platform. For the sake of proving our system's validity, we simulated the target platform using an ARM Emulator (QEMU) [170]. This software simulates an Thumb-2 Arm microprocessor along with its system-level architecture. Nevertheless, we plan to test the system with real hardware kits in our future work.

We added our obfuscation extension to LLVM framework version 12, and used it to

***Table 4.1*:** Comparison of Range of Normalized Runtime Differences in Both Versions of Our System.

Benchmark Name	Old System	New System
Float	-0.3 : 2 %	2.4 : 4.9 %
IntMM	-16 : 5 %	24 : 49 %
Perm	0.95 : 3.9 %	2 : 10 %
Queen	-5.8: 3.8 %	3.6 : 10.8 %
Quick Sort	-4 : 3.2 %	3.8 : 9.9 %
Puzz	0.7 : 3.9 %	9.4 : 12.6 %
Oscar	-1.8 : 4.7 %	1.5 : 6.6 %

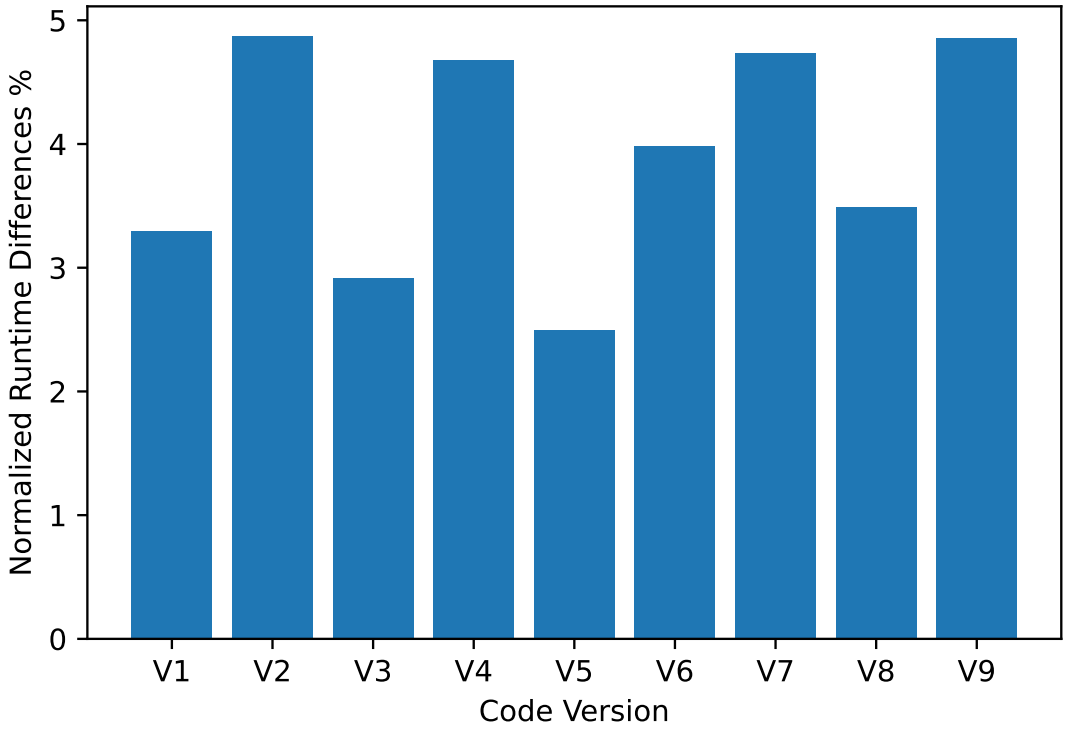

***Figure 4.3*:** Obfuscated code versions of the Float program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

cross-compile the source of the input programs and produce machine code targeting the ARM platform. Our technique is easily integratable with any other version of the LLVM compile that supports the "If Conversion" transformation.

As a proof of concept, we choose some simple yet standard benchmark programs as the subject for our experiments. These benchmarks are well-known algorithms borrowed from the test set of the Gem5 software [171]. We fed these programs to our system, which first added some dummy code that contains branch instructions. This dummy code is randomly selected from a pool of code base programs. We did that to avoid using the same code

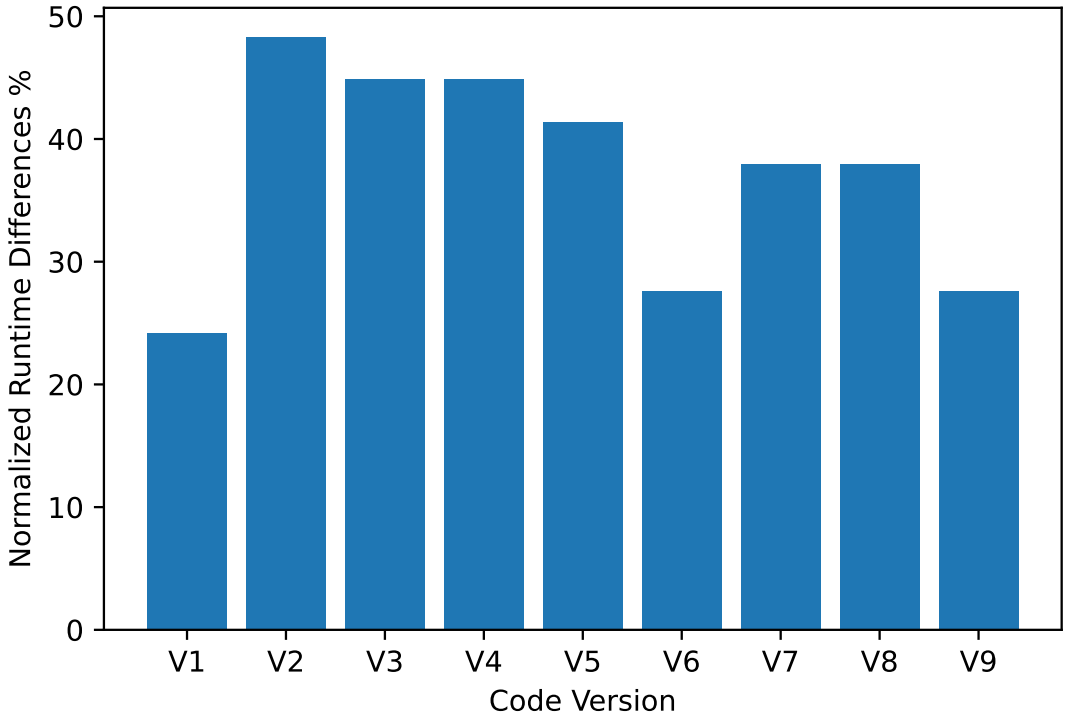

***Figure 4.4*:** Obfuscated code versions of the IntMM program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

multiple time, hence this could present a vulnerability in our system. Then the modified code would be inserted into the control-flow obfuscation pass. Which in turn does random If conversion transformation to the program. Finally, a machine code tailored for the ARM platform is produced using the LLVM cross compiler, as explained before in Figure 4.2. The previous steps were summarized in Figure 4.1.

We did the same process multiple times for each input program, in order to produce different code version for the same program. Then, we compared the running time of each code version. This study of time measurement is the basis for thwarting timing side-channel attack because the greater and the more unpredictable the runtime, the more it will be difficult for an attacker to make statistical correlations about the behavior of the program, hence he can't leak information about it or reverse engineer it.

Figures 4.3 through 4.9 show the normalized runtime difference percentages across the various versions that we produced for each individual benchmark. Each of these different program versions corresponds to a different bitmask, hence another predicated instruction series and consequently different running times, although the programs have exactly the same functionality. The running times were normalized in comparison to the runtime of the original unmodified code version, and the results were plotted in these figures respectively. The equation for the normalized runtime difference can be expressed as follows:

$$\Delta T\% = ((T_{Vi} - T_O)/T_O) \times 100 \qquad (1)$$

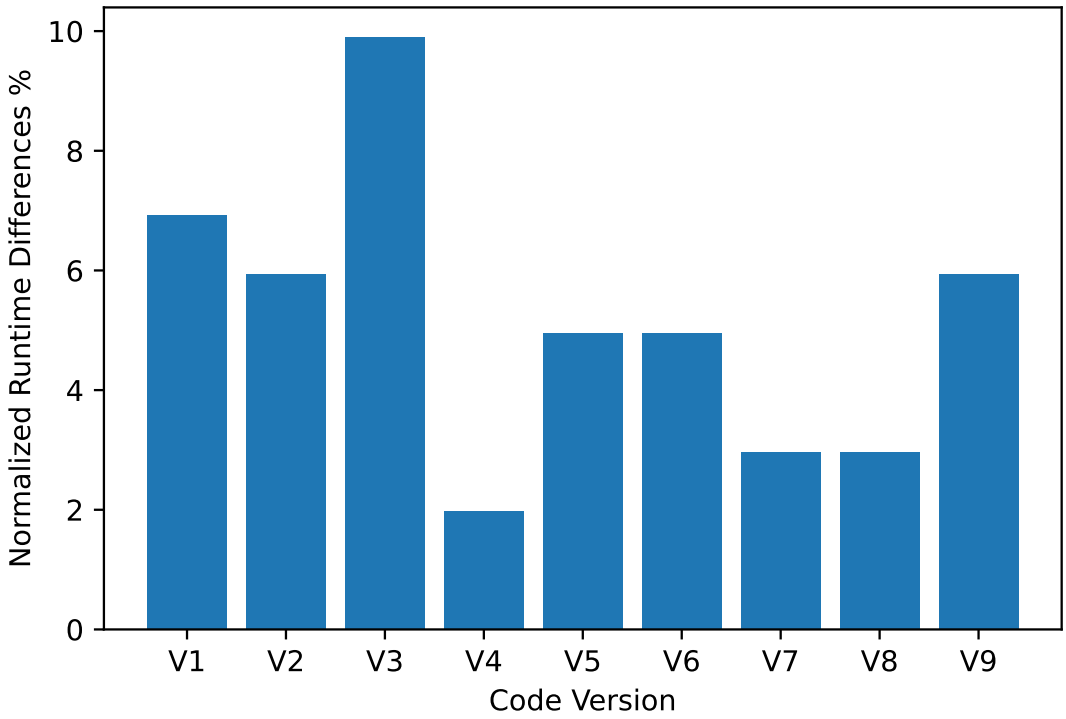

***Figure 4.5*:** Obfuscated code versions of the Perm program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

where T_{Vi} is the runtime for that specific obfuscated code version (which in turn corresponds to s specific bitmask), T_O is the runtime for the original unmodified version. This measure service as both a performance metric and indicator of the robustness of our security proposal.

As we can see that these different code version all have one thing in common, they were expectedly slower than the original code version, due to our current system enhancement that is the insertion of the randomly selected dummy code with more branch opportunities. It also went under the same control-flow obfuscation transformation and produced varying code versions with different runtimes.

To prove on how this enhancement affected the overall system, we experimented on the same benchmarks using the old and the new techniques on the same platform and collected the overall range of normalized runtime differences. We summarized the comparison of both techniques in Table 4.1.

For example in Figure 4.3 we can see that these normalized differences varied between 2.4% and 4.9%, which means they ran a little slower than the original code, with different percentages for each code version. On the other hand from Table 4.1, we know that this range was -0.3% to 2%, which was quite small, proving that we needed this sort of enhancement.

The same goes for the IntMM benchmark plotted in Figure 4.4, showing a range of 24% to 49% as compare to the -16% to 5% with the old technique as per Table 4.1. Also the Perm program in Figure 4.5 had a range of 2% to 10% as opposed to 0.95% to 3.9%. In

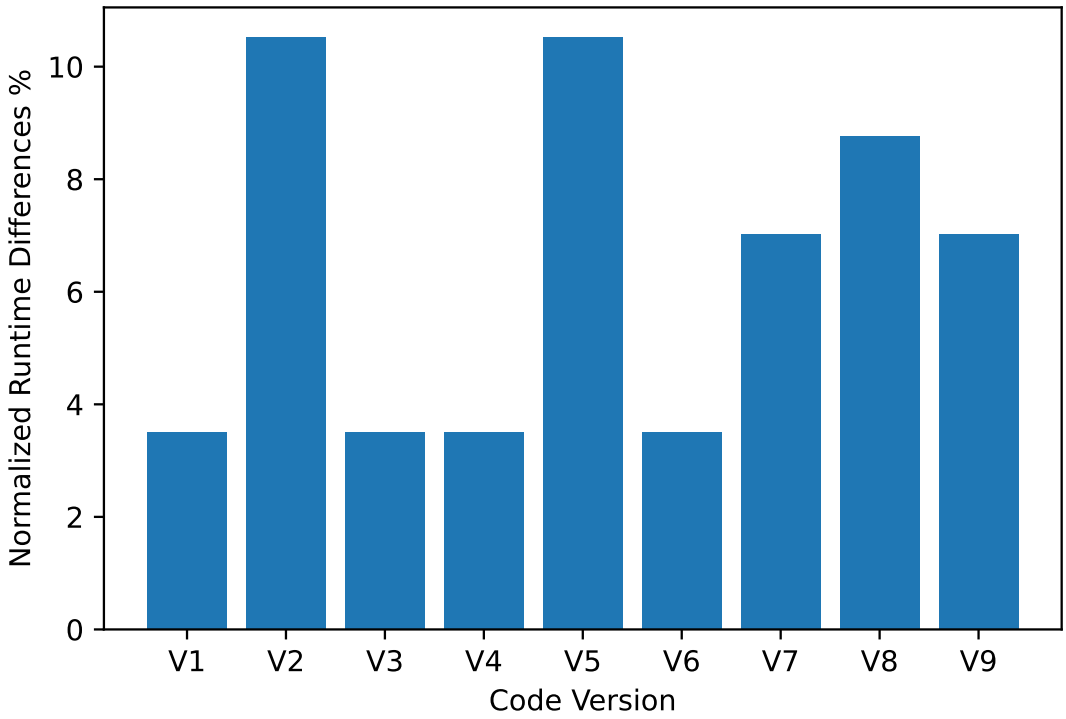

***Figure 4.6*:** Obfuscated code versions of the Queen program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

addition, the Queen program shown in Figure 4.6 showed similar results, ranging between 3.6% to 10.8% as opposed to -4% to 3.2%. As for the Quick Sort program, which we know from our previous work [173], has many recursive calls, thus it was more challenging to perform the if-conversion transformation. Having the opportunities for predication are limited [172], the time changes were also minor -4% to 3.2% as per Table 4.1. But with our new technique we achieved a range of 3.8% to 9.9%, and the statistics from its code versions were plotted in Figure 4.7.

On the other hand for the Puzz program shown in Figure 4.8, although the range is between 9.4% to 12% which is comparably larger than 0.7% to 3.9%, we have to note the width of the newly introduced range (the difference between the maximum and minimum values) is somewhat smaller than some of previous cases, hence the difference between the code versions themselves was not as big as it was in other programs. This could have happened because the inserted random dummy code didn't have many branch conversion opportunities, hence we need a more concise way to choose dummy code instead of just randomly inserting it. This needs to be addressed in future work, because as more as we have significant time changes, we can disrupt timing side-channel attacks.

Also, for the Oscar program, shown in Figure 4.9, the old technique achieved a range of -1.8% to 4.7%, and now with the new technique we have a range between 1.5% to 6.6% which some would say a smaller performance gain. Also, they could have happened because of a poor choice for the random dummy code. Though, unlike the Puzz program, here the difference between the code versions is quite noticeable, meaning that the results could be

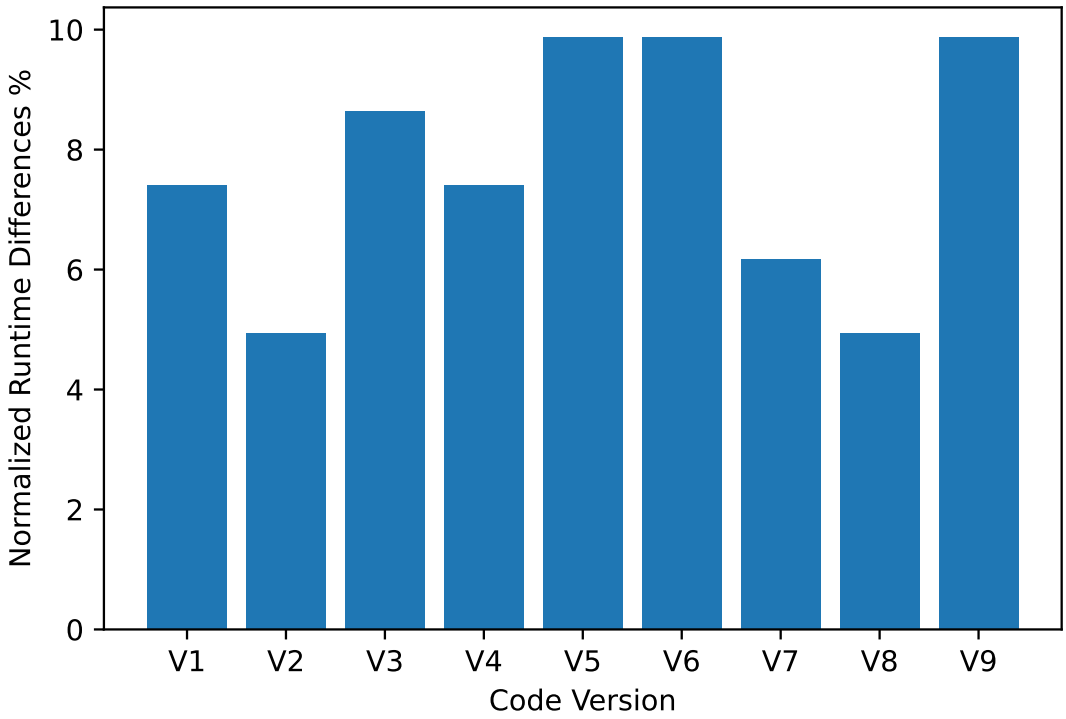

***Figure 4.7*:** Obfuscated code versions of the Quick Sort program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

acceptable in mitigating the attacks.

In summary, we can see from these different results that in most programs, we achieved considerable enhancement over the old technique as per the resulting normalized time differences. That said, in some cases we need some way to efficiently insert random dummy code that would produce more runtime changes. We intend to further investigate that in our future work.

4.5 Related Work

Mitigating side-channel attacks on different architectures and platforms, have been studied extensively over the past years [161] [162]. For example, in [165], the authors focused on the ISA of an ARM processor and compared different SCA and their countermeasures. But most of these techniques tried to protect cryptographic algorithms from encryption key leakage via SCA, but here, we try to obfuscate programs control-flow. Thereby, we disrupt the sole dependence between a program's execution behavior and the input data. Hence, we can defend against SCA aiming to analyze the code and perhaps reverse engineer it.

The authors in [166] proposed an automatic software tool to generate polymorphic version of a functions of a program. They used register shuffling, instruction reordering and dummy code insertion among other techniques. But here, we use a rather lightweight technique, which provides the adequate functionality with minor modifications to the resulting code size, making our technique simple to apply. However, it can be combined

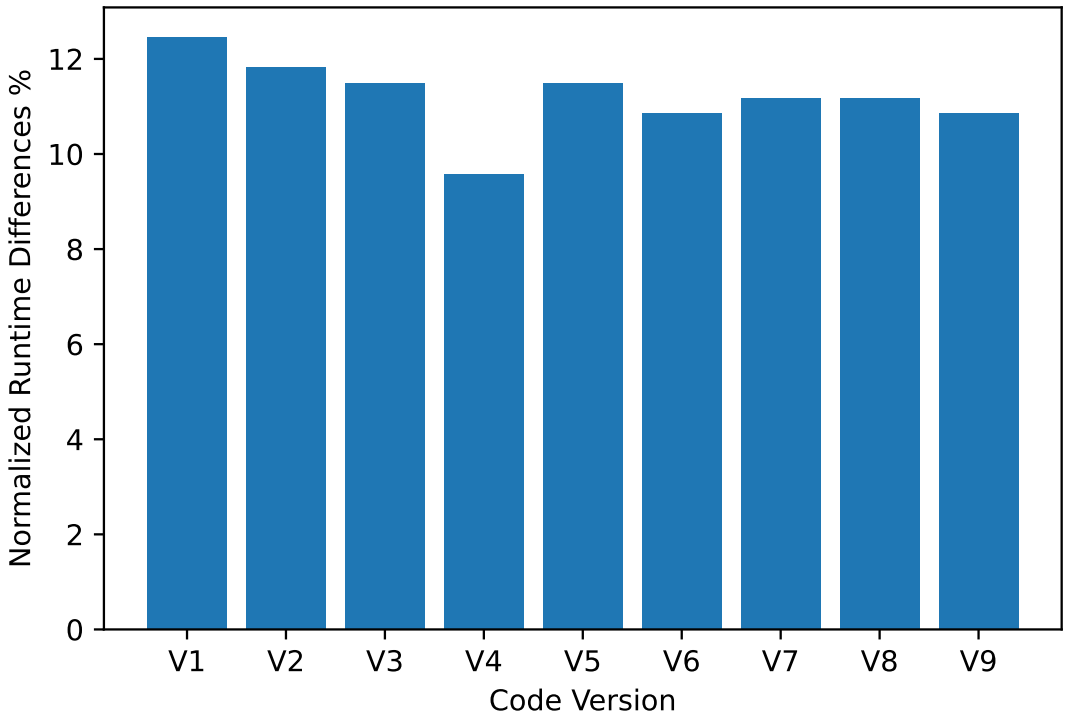

***Figure 4.8*:** Obfuscated code versions of the Puzz program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

with other techniques to provide more protection.

Moreover, the use of the LLVM open source framework, which has many implemented front-end and back-end interfaces, makes our system usable in many platforms independent of the input programs high level language and the target architecture. Which in turn makes our proposed technique an ideal candidate for application on remote platforms such as the AVCC, especially in the absence of strong user side governance of the execution environment or knowledge of potential adversaries to protect against.

4.6 Summary

AVCC platforms are fairly new and they are still under investigation. Security and privacy problems are among the most important issues facing there wide-scale adoption. One of the most serious security threats are side channel attacks (SCA). In our work, we proposed an obfuscation mechanism to protect software running on AVCC against timing SCA. Virus and malware developers use obfuscation to hide their code from scanners and detectors. We used similar concepts to implement a dynamic yet randomized control-flow obfuscation technique using conversion of conditional branches. This would result in seemingly unpredictable disruptions to normal code behaviour, making it difficult for attackers to leak information about the running program. In this current approach we provide enhancements to our early work. In particular, we compensate for the cases where there were limited opportunities for conditional branch conversions, which previously introduced hindrance to our obfuscation

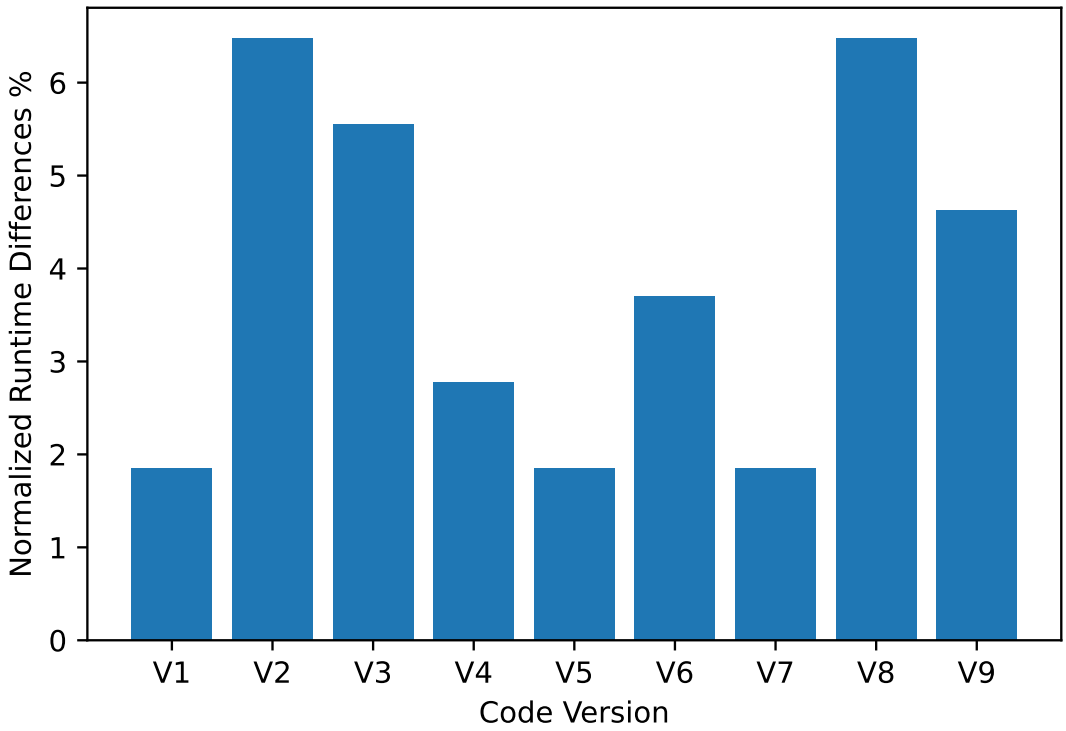

***Figure 4.9*:** Obfuscated code versions of the Oscar program and their corresponding percentage of normalized runtime differences with respect to the original unmodified version.

mechanisms. Therefore, we were able to extend more protection for a wider range of generic programs running on embedded system based platforms, which is typically the case in hand in AVCC.

The results show that our LLVM-based obfuscation system is platform agnostic, independent of the input language and quite lightweight, which makes it a good candidate for application the AVCC platform.

Chapter 5

A PRIVACY-AWARE SYSTEM FOR AUTHENTICATION IN AUTONOMOUS VEHICLES CLOUD COMPUTING

5.1 Introduction

AVCC platforms are a honeypot for intrusions attacks. Attackers aim to gain access to cloud resources by impersonating as a legitimate user. Moreover, they can implant themselves as participants of the cloud system itself and gain access to sensitive data. Therefore, a robust authentication mechanism is needed to mitigate misusing attacks on the AVCC and protect all stakeholders. Moreover, the authentication technique should protect user's privacy, therefore it's required that system would be able search over encrypted data and authenticate participants. Additionally, the proposed system should have a low overhead in terms of communications and computations. Our goal is to countermeasure different kinds of attacks, for example, unauthorized access, man-in-the-middle attacks or fraudulent identities for AVCC platforms.

In order to achieve that, we used a secure function encryption algorithm, based on Kim et al. work [174]. Their technique utilizes function-hiding technique that requires nothing but knowledge of ciphertexts and doesn't compromise anymore data about actual identity information beyond the secure inner product of the encrypted authentication request and ID data also stored in an encrypted format. We extended their scheme to fit our scheme and support searching multiple encrypted ID data stored at some independent referee server, and match an authentication requested with it.

In particular, this chapter has the following contributions:

- ***First***, we implement an authentication mechanism based on a low-cost and efficient authentication using function encryption, which is an efficient encryption algorithm;
- ***Second***, we prove that our system is a privacy-preserving encrypted identity searching technique that mitigates linkability and information leakage attacks;
- ***Finally***, we tested our system on different usage scenarios to search a pre-stored encrypted set of ID data and authenticate a recognized client. Our findings proved that the suggested technique entails reasonable overheads in terms of communication and computation, which is acceptable and applicable for AVCCs.

The rest of this chapter is laid out as follows. In Section 5.2, we describe the system's network and threat models, also we mention our design goals. In Section 5.3, we demonstrate the scheme behind our system. In Section 5.4, we show a privacy and security analysis of our proposed algorithm. Section 5.5 illustrates experimentation and result. Section 5.6 provides a summary of the relevant literature. Finally, in Section 5.7, we summarize the chapter and suggest future work.

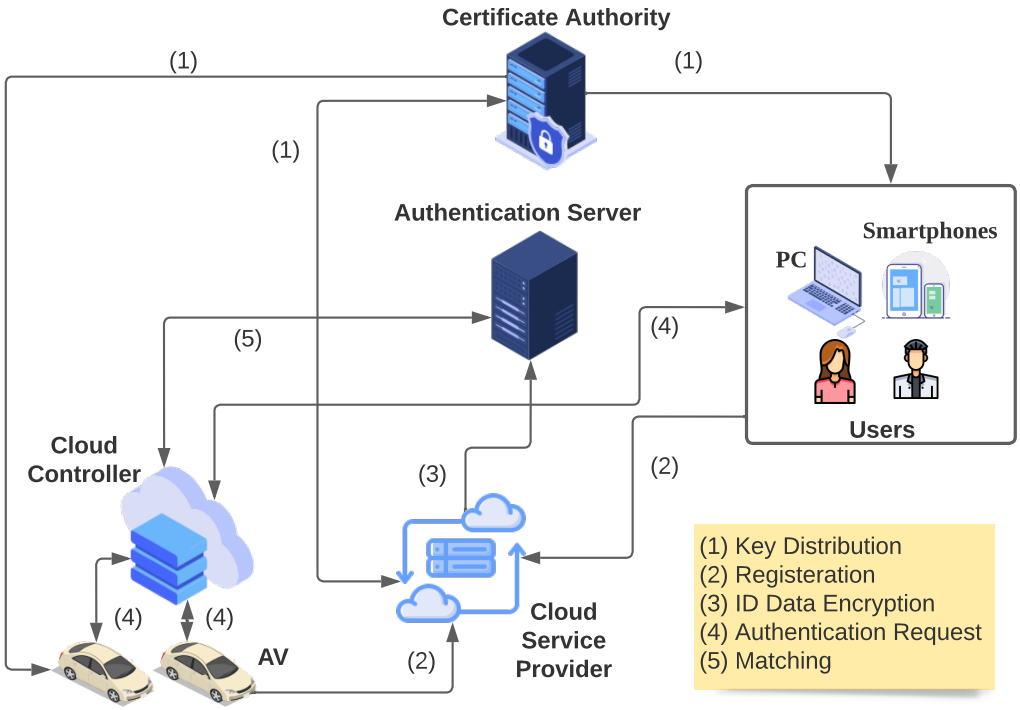

***Figure 5.1*:** Our proposed secure AVCC network model.

5.2 System Models

5.2.1 Network Model

In Figure 5.1, we show the components of our proposed network model. We have a Certificate Authority (CA), A Cloud Service Provider, A Cloud Controller, An Authentication Server, Users on their computers or smartphones and AVs. We assume that all stakeholders have a secure means of communications to connect with each other, and we focus here on authentication of users and vehicles trying to access or join the AVCC, respectively. The encryption scheme is divided into five phases. First, the Certificate Authority (CA) distributes the keys to all entities, including the cloud service provider, vehicles and users. Then in the second phase, users and vehicles register at the cloud service provider with their original ID data.

For the sake of simplicity and since all following steps will be essentially the same for users and vehicles, from here on, we will refer to the first party the authenticator (that is the cloud controller) and the second party being authenticated as the participant (which is the user or the vehicle), and the party who does the actual matching on encrypted data is the Authentication Server. The main difference between the users and vehicles is in the corresponding ID size n, which is chosen to be double the size of their original unique identification data size (N), which is, in the case of a user could be a unique randomly

generated ID number or any sort of biometric data such as his/her fingerprint, voice print or iris image ...etc. For vehicles, the unique ID could be its license plate number, Vehicle identification Number (VIN), or even a PUF (Physically Unclonable Function) unit installed in the vehicle.

Consequently, in the third phase the cloud service provider encrypts participant registered IDs using its secret key to generate encrypted indices and store them at an independent authentication server. Then, whenever a participant is trying to access the cloud (either a user trying to utilize the cloud or a vehicle would like to offer its services and join the cloud), they encrypt their ID using the key that was given to them from CA and generate an authentication request, which is called a trapdoor and sends it to the authenticator. Finally, the authenticator delegates the process to the authentication server, which perform a search and match operation, during which the server executes a series of operations that constitute a functionally encrypted secure dot product using the indices and trapdoors and then figure out if the participant is legitimate to access the cloud, then they can be authenticated, otherwise he will be denied access. We delve into more detail about these steps in the following Section.

5.2.2 Threat Model

External perpetrators and malicious insiders, such as AVCC service providers, cloud controllers, authentication servers and other cloud users, are all viable attackers. We consider them as being "honest-but-curious," meaning that they don't wish to tamper with the system functionality, but instead they may be tempted to eavesdrop on private information such as indices or authentication requests. This attack model is called *"Known-Ciphertext"*.

5.2.3 Design Goals

The proposed privacy-preserving authentication scheme should attain the following design goals:

1) Authentication Query Search Over Encrypted ID Data from Several Participants. The scheme should be able to use encrypted authentication queries to search over encrypted ID data stored by different cloud participants.

2) Scalability and Efficiency. The scheme should be capable of quickly searching through a large number of indices in order to reply to authentication requests. Additionally, the authentication query's size should be reasonable to minimize communication overhead.

3) Indices and Authentication Requests Confidentiality. The cloud server should not acquire any useful information about the stored index data or authentication trapdoors.

5.3 Proposed Scheme

In this section, we describe the operational steps of our proposed scheme. The algorithm is based on [174], bilinear pairing method. For that we recall $\mathbb{G}_1$ and $\mathbb{G}_2$ are two asymmetric groups of prime order q. Then, we select $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ as generators. Then, we generate the mapping $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, where $\mathbb{G}_T$ is also of prime order q, such that

$e(g_1, g_2) \neq 1$ and $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$, $\forall x, y \in \mathbb{Z}_q$. Also, all operations in the groups has to be computable [175].

Initialization

The CA initiates the system's structure by generating and distributing cryptographic keys to the cloud service provider, vehicles and users. The CA should execute the following oracles to do this:

- $SystemSetup\left(1^{\lambda}, S\right) \rightarrow \mathcal{CAMK}$. The initialization algorithm, needs two inputs: a security parameter λ, and the set S where $S \subseteq \mathbb{Z}_q$, such that $|S| = poly(\lambda)$, of prime order q. Then, it generates an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$, having $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ as generators. Then, based on the ID size n, CA samples $\mathbf{A} \leftarrow \mathbb{GL}_n\left(\mathbb{Z}_q\right)$, such that $\mathbf{A}$ is an $n \times n$ dimensional matrix.Then, the scheme computes the public security parameters:

  $\text{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e, g_1, g_2)$,

  and the CA secret master key:

  $\mathcal{CAMK} = (\mathbf{A})$.

  Then, for each cloud service provider's cryptographic operations, the CA samples two random matrices $\mathbf{M}_1 \leftarrow \mathbb{GL}_n\left(\mathbb{Z}_q\right)$ and $\mathbf{M}_2 \leftarrow \mathbb{GL}_n\left(\mathbb{Z}_q\right)$, Such that $\mathbf{M}_1$ and $\mathbf{M}_2$ are matrices of size $n \times n$.

  That said, we have to notice that this steps will be done twice to create two set of security parameters and secret keys one for dealing with the users and the other for vehicles' dealings. The difference will be in the input the ID size n, which is probably going to be different form users (lets say n_u) to vehicles (say n_v). But as we said before, for simplification we will use the former generalized notation, referring to users and vehicles as cloud participants.

- $CloudServiceProviderKey(\mathcal{CAMK}, \mathcal{CSP}) \rightarrow \mathcal{CSPK}$. The CA computes a cryptographic key for the Cloud Service Provider $\mathcal{CSP}$, using its randomly generated matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ as follows:

  $\mathcal{CSPK} = (\mathbf{AM}_1, \mathbf{AM}_2)$

  $\mathcal{CSP}$ uses this key in participant authentication operations, to store participant IDs as encrypted indices and matching afterwards.

- $ParticipantKey(\mathcal{CAMK}, \mathcal{P}_i) \rightarrow \mathcal{PK}_i$. The CA computes the secret key for each registered participant in the system with the identity $\mathcal{ID}_i$ as follows. It samples $\mathbf{A}_i'$ and $\mathbf{A}_i''$ from $\mathbb{Z}_q^{n \times n}$ where $\mathbf{A}_i' + \mathbf{A}_i'' = \mathbf{A}^{-1}$ and outputs the secret key as:

  $\mathcal{PK}_i = \left(\mathbf{M}_1^{-1}\mathbf{A}_i', \mathbf{M}_2^{-1}\mathbf{A}_i''\right)$

Registering and Encrypting Participant ID Data

First, each participant (a user or a vehicle) has to register their ID data with the cloud service provider. Then, the $\mathcal{CSP}$ creates a collection of encrypted indices as the encryption of their ID data using their corresponding key $\mathcal{CSPK}$ that we created in the previous step.

That is, for participant $\mathcal{P}_i$, $\mathcal{CSP}$ creates $\mathcal{I}_i$ for the embedding of their ID data of size n, $(\mathcal{PID}_i)$ by executing the following oracle:

$CreateIndex\,(\mathcal{CSPK}, g_1, g_2, \mathcal{PID}_i) \rightarrow \mathcal{I}_i$. Such that, we first compute the ones complement of the $\mathcal{PID}_i$ lets call it $\mathcal{PID}'_i$ then we compute the new participant ID as $\mathcal{PID}''_i$, which is the concatenation of the original ID and its one complement.

I.e: $\mathcal{PID}''_i = \mathcal{PID}_i \frown \mathcal{PID}'_i$

Notice that this new ID has a size that is double the original, corresponding to n that we used as input ID size.

Then $\mathcal{CSP}$ samples $\alpha \leftarrow \mathbb{Z}_q$ and then generates the index as:

$\mathcal{I}_i = \left(g_1^{\alpha}, g_1^{\alpha \cdot \mathcal{PID}''_i \cdot AM_1}, g_1^{\alpha \cdot \mathcal{PID}''_i \cdot AM_2}\right)$.

Thus, $\mathcal{I}_i$ is stored at the authentication server for later authentication at some cloud controllers to allow a participant to access cloud facilities if he/she was a user or lend its resources if it was a vehicle.

Encrypting Participant Authentication Requests

If a user wants to utilize the cloud system, he/she would have to first log into their registered account at the could service provider. Then, based on availability and resource needs, the cloud service provider would direct them to a certain cloud controller for further authentication and access will be granted there.

Same goes when a vehicle wants offer their computing capabilities and join a cloud, they should log into their registered account at the could service provider. Then, $\mathcal{CSP}$, based on various metrics such as the vehicle available capabilities, its geographic location and after examining which nearby cloud needs more resources, the cloud service provider would direct the vehicle to a certain cloud controller for further authentication and the join operation will be completed there.

Therefore, the next step for any user or a vehicle is to encrypt their authentication request (out of their ID data) and generate a trapdoor using their own key, that was distributed to them by the CA in the initialization phase.

That is for the authentication of any participant, who want to access/join the cloud, they should encrypt their ID data (lets call it vector $\mathcal{X}_j$, since it is still unauthenticated) by invoking the following oracle:

$CreateTrapdoor\left(\mathcal{PK}_j, g_1, g_2, \mathcal{X}_j\right) \rightarrow \mathcal{T}_j$.

First, again as we did before, we compute the ones complement of $\mathcal{X}_j$ lets call it $\mathcal{X}'_j$ then we compute the new participant ID as $\mathcal{X}''_j$, which is the concatenation of the original ID and its one complement, i.e: $\mathcal{X}''_j = \mathcal{X}_j \frown \mathcal{X}'_j$. Then, we calculate its transpose $\mathcal{X}''^T_j$.

Then, we sample $\beta \leftarrow \mathbb{Z}_q$ and finally we generate the trapdoor as:

$\mathcal{T}_j = \left(g_2^{\beta}, g_2^{\beta \cdot M_1^{-1} A' \cdot \mathcal{X}''^T_j}, g_2^{\beta \cdot M_2^{-1} A'' \cdot \mathcal{X}''^T_j}\right)$.

Thus, $\mathcal{T}_j$ is sent to the cloud controller, which in turn forwards it to the authentication server for matching and authentication.

Matching and Authentication

For authentication of a participant, the authentication server should run Inner Product Encryption (IPE) algorithm to match the incoming authentication request (trapdoor) with the stored participants' IDs (indices).

In particular, the authenticator performs a secure dot product operation between the trapdoor $\mathcal{T}_j$ and each stored index $\mathcal{I}_i$, by invoking the following oracle.

$Match\left(\mathcal{I}_i, \mathcal{T}_j\right) \rightarrow < \mathcal{PID}''_i \cdot \mathcal{X}''_j >$. The authenticator computes the following:

1. $E_1 = e\left(g_1^{\alpha}, g_2^{\beta}\right) = e\left(g_1, g_2\right)^{\alpha\beta}$

2. $E_2 = e\left(g_1^{\alpha \cdot \mathcal{PID}''_i \cdot AM_1}, g_2^{\beta \cdot {M_1}^{-1} A' \cdot \mathcal{X}''^T_j}\right)$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \mathcal{PID}''_i \cdot AM_1 {M_1}^{-1} A' \cdot \mathcal{X}''^T_j}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \mathcal{PID}''_i \cdot AA' \cdot \mathcal{X}''^T_j}$

3. $E_3 = e\left(g_1^{\alpha \cdot \mathcal{PID}''_i \cdot AM_2}, g_2^{\alpha \cdot {M_2}^{-1} A'' \cdot \mathcal{X}''^T_j}\right)$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \mathcal{PID}''_i \cdot AM_2 {M_2}^{-1} A'' \cdot \mathcal{X}''^T_j}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \mathcal{PID}''_i \cdot AA'' \cdot \mathcal{X}''^T_j}$

4. $E_4 = E_2 * E_3$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \mathcal{PID}''_i \cdot AA' \cdot \mathcal{X}''^T_j + \alpha\beta \cdot \mathcal{PID}''_i \cdot AA'' \cdot \mathcal{X}''^T_j}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \langle \mathcal{PID}''_i \cdot (AA' + AA'') \cdot \mathcal{X}''^T_j \rangle}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \langle \mathcal{PID}''_i \cdot A(A' + A'') \cdot \mathcal{X}''^T_j \rangle}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \langle \mathcal{PID}''_i \cdot AA^{-1} \cdot \mathcal{X}''^T_j \rangle}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot \langle \mathcal{PID}''_i \cdot \mathcal{X}''^T_j \rangle}$
   $= e\left(g_1, g_2\right)^{\alpha\beta \cdot < \mathcal{PID}''_i \odot \mathcal{X}''_j >}$
   $= E_1^{< \mathcal{PID}''_i \odot \mathcal{X}''_j >}$

5. Knowing E_1 and E_4, we calculate:
   Discrete $\log(E_4, E_1) \longrightarrow < \mathcal{PID}''_i \odot \mathcal{X}''_j >$

A correct match happens if the product result equals exactly half the ID size or $n/2$. In which case the hamming distance between the trapdoor and the matched index is zero, which means they are the same. Hence, the participant is securely authenticated, without revealing their privacy (their actual ID) to the cloud controller. Otherwise, if no match for the trapdoor is found, they will be denied access.

5.4 Privacy and Security Analysis

In this section, we analyze the privacy preservation features of our system.

Preposition 1. *Search and Match Over Encrypted ID Data from Several Participants.*

Proof. It's mathematically proven in the previous section that our authentication server can compute the inner product between a trapdoor and a collection of encrypted indices stored by cloud service provider and use it as basis for authenticating a legitimate participants.

Preposition 2. *Efficiency and Scalability.*

Proof. It is proven by experiments that our scheme was able to quickly searching through a large set of indices in order to reply to authentication requests, that's because our mathematical operations are fairly straightforward and parallelizable. Additionally, the authentication request size is reasonable, thereby we minimized communication overhead.

Preposition 3. *Indices and Authentication Requests Confidentiality.*

Proof. In the proposed scheme, our searchable encryption scheme encrypts identification data of cloud participants. This technique considers known ciphertext attack model. That is, without knowledge of their cryptographic keys, decrypting the indices and trapdoors is impossible, even if they can obtain the encrypted data.

5.5 Experiments and Discussions

We implemented our proposed algorithm using Python and executed it on a computer running Ubuntu 18.04.6 LTS on an Intel® Xeon™ E5-1650v4 @ 4 GHz processor and 40 GB of RAM. In our experiments, we used randomly generated ID vectors of different sizes. The purpose of the experiments was to determine the communication and the computation costs of our technique. In addition, we experimented with increased number of users to examine our system's scalability . We used different randomly generated sample data in each experimental step and calculated the average results from these various runs.

5.5.1 Communication Overhead

Our technique is fairly straightforward, that is, depending on the chosen ID size (n), the encryption key at either side ($\mathcal{CSPK}$ for the cloud service provider or $\mathcal{PK}_i$ for the participant) is represented by two matrices of size $n * n$. Considering that integer data types are stored in 4 Bytes in Python, consequently the size would eventually be $8 * n * n$ bytes. Additionally, given that we used asymmetric pairing curve (BN256) cryptographic library where the size of a group element is 32 Bytes, consequently, the size of each encrypted vector (index or trapdoor) is (1+2*n)× 32 Bytes. Therefore, since we experimented with different ID sizes, the communication overhead will vary accordingly, (e.g: the key would reach up to 131 KB, when the n was 128). We have to mention that increasing the ID size (and hence the key size) would make our system more secure, because it would be more difficult to brute-force

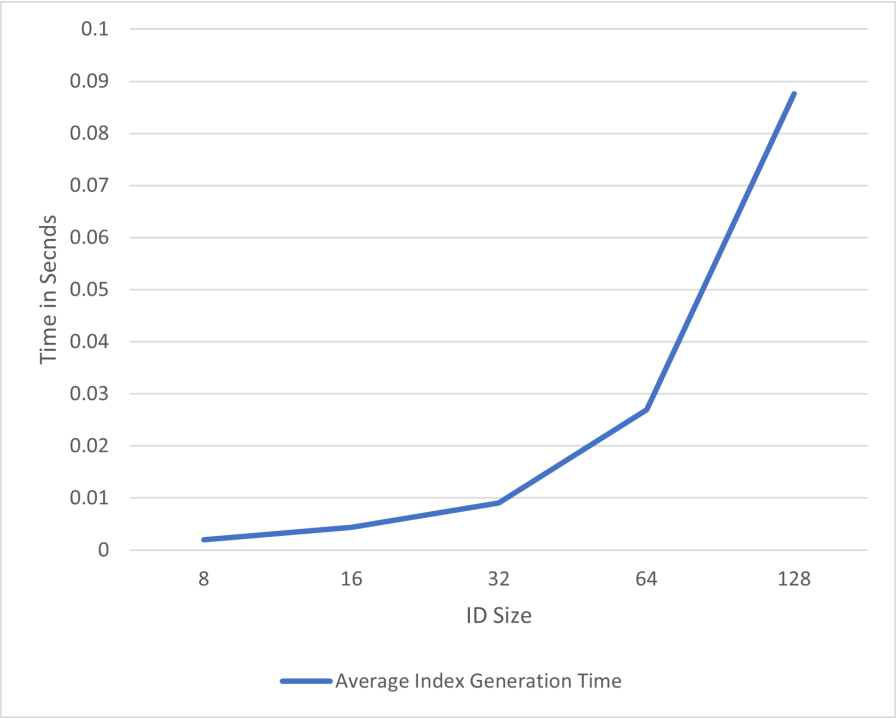

***Figure 5.2*:** Encryption time results across different ID sizes for index and trapdoor.

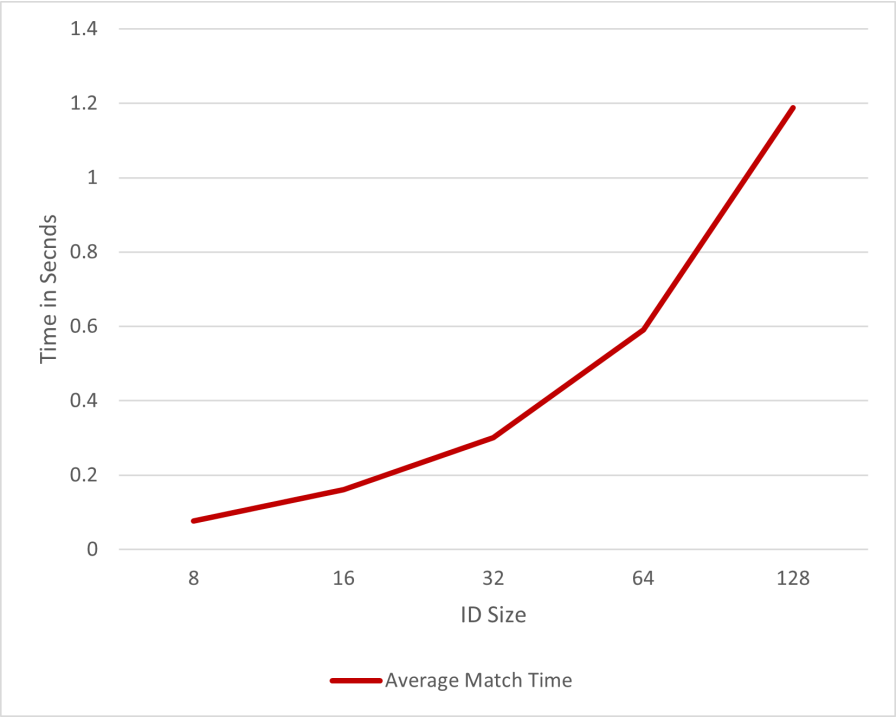

***Figure 5.3*:** Matching time results across different ID sizes.

or guess the keys. Though this would be a trade-off in terms of communication overhead

and also computation overhead as we will see next.

5.5.2 Computation Overhead

Key Size Effect

In this experiment our goal was to determine the effect of changing the key on the encryption and matching operations. Therefore, we chose an arbitrary small number of users (2) and an ID size range between 8 to 128. Figure 5.2 shows the average time reacquired to encrypt either the index (The $CreateIndex$ Oracle) or the trapdoor ($CreateTrapdoor$) as well, since they essentially include the same matrix calculations. As we can see the curve is proportional to the ID size and might increase even more if we choose a very large ID size. That's why we need to choose a reasonable ID size according to our needs. Nevertheless, for an ID size of 128 , the timing results are within the order of *msec*. Likewise, the same can be noticed for the average matching time, which includes the $Match$ Oracle and all of its operations. As we can see from Figure 5.3, the results were also within acceptable margins.

Consequently, the results point out that all encryption processes were executed in the matter of *msecs* and our matching process takes under under 1.2 *sec* even with ID size of 128. Nevertheless, these numbers can be expected to be further improved if parallel computing platforms were utilized to aid with the matrix operations. That said, our findings show that our scheme is performing well, whilst using a moderately equipped machine, which suggest that the efficiency will increase when applied in an actual AVCC with more advanced computational power.

Encryption Scalability

We studied the performance of our system in gradually increasing usage scenarios, increasing the number of participants every time with a fixed ID size of 64, then with a size of 32. These two scenarios exemplify two modes of operation, with the first being stronger and more secure per se, and the latter being more lightweight in terms of overheads as mentioned before.

That said, We measured the total time required to encrypt the entire database of participants indices and plotted the results in Figure 5.4. The Figure shows that our system's performance consistently, even with the increasing number of system users. The encryption times were within reasonable margins, taking around 9.5 seconds to encrypt a database of 1000 indices with the larger encryption key size (where n was 64). Albeit in the other case (where n=32), the total time was around 4 seconds, while maintaining a reasonable level of security.

Search Time Efficiency

The same usage scenarios were examined here as well. For testing purposes, we used a number of authentication request equal to 20% of the entire database of stored indices at that iteration and reported the average from the results. We studied the average time required for the authentication server to perform $Match$ Oracle between the authentication request and

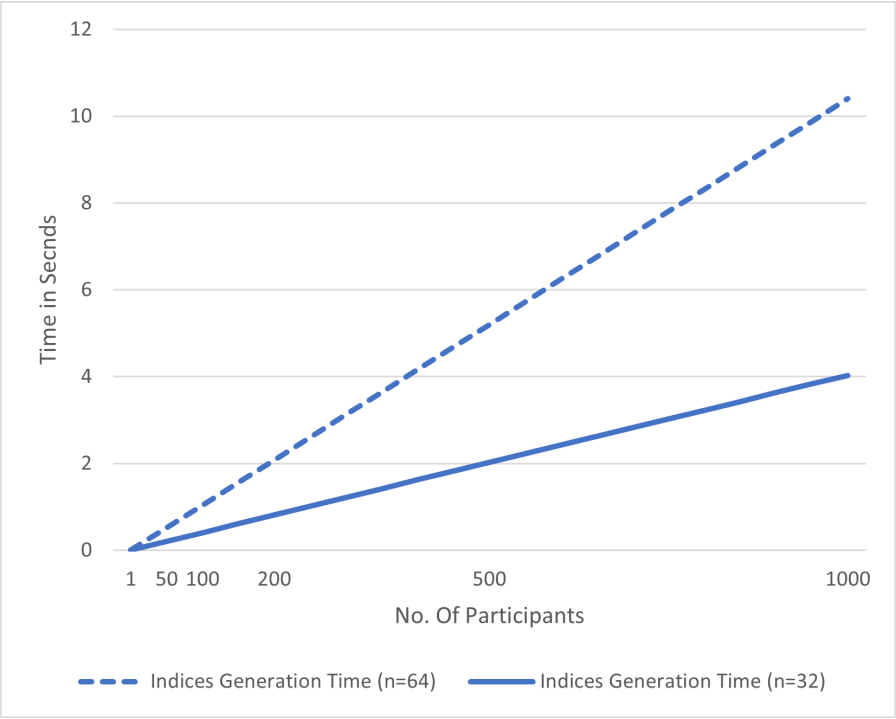

***Figure 5.4*:** Indices encryption time results with different numbers of users for $n = 64$ & $n = 32$.

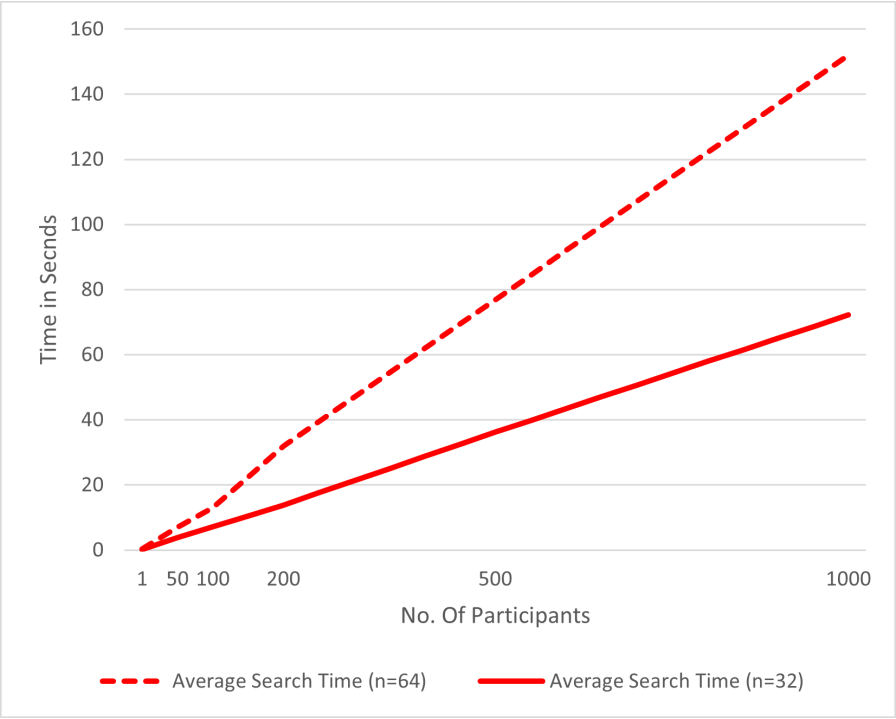

***Figure 5.5*:** Search time results with different numbers of users for $n = 64$ & $n = 32$.

all the stored indices. If a match was found (juding by the inner product results as mentioned

***Table 5.1*:** Security & Privacy Attacks On AVCC Applications.

Ref. No.	Threat	Proposed Method	Critique
[176]	Compromising user privacy	CP-ABE	Entails heavy computations and communication overhead, especially during the start up and key distribution phase.
[177]	Compromising user privacy and authentication	Hierarchical ABE	System is exposed to too many parties, and there are many privacy concerns. Additionally, having to deal with the complicated system of attribute distributions.
[178]	Attacks on user authentication	Biometric and ECC-assisted authentication	ECC involves heavy computations and therefore consumes a lot of power.
[179]	Attacks on user authentication	Single-server 3-factor AKA protocol and the non-interactive identity-based key establishment protocol	The centralized server becomes a single point of failure, also there is more communication overhead.

before,then the participant is validated and the search stops, otherwise they will be denied access. As shown before in Figure 5.3, these operations were completed in the matter of a second give or take some *msec*; therefore, the average time of search is in the order of seconds as shown in Figure 5.5, albeit with our moderate server. As expected, we notice that the results grow proportionally with database size of stored indices. The figure shows that the case when (n=32) is more suitable for application in real-time scenarios because the time required to search through a thousand user was around one minute.

In summary, our finding demonstrated that the proposed scheme is efficiently scalable and befitting for application in AVCC systems. Moreover, adopting parallel search techniques using multiple servers could further enhance the system's time performance.

5.6 Related Work

Although AVCC is a fairly new paradigm, it's based on the VCC and VANET technologies. Therefore, it inherited their problems and security concerns. Such as DoS, jamming, hijacking authentication, racketeering, copyright infringements, stealing data, sabotage, and information leakage via side-channels and reverse engineering [14]. Fortunately, some of the approaches to mitigate these issue seem relevant here as well, thus they can be applicable.

In [176] the authors propose an algorithm to ensure VCC security and privacy. They use Pseudo-ID instead of vehicles' real ID to provide conductors' privacy, Identifier-Based

Signature mechanism is used to guarantee vehicles' authentication, and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) algorithm is used for key distribution. They claim that their liGhtweight secURe AutheNticaTion and keY distribution scheme for vehicular cloud computing (GUARANTY) ensures a secure keys distribution to minimize the encryption and decryption computation cost. In their scheme, vehicles use a symmetrical cryptography in their communication. But in our opinion their system does entail some heavy computations and communication overheads, specially during the start up and key distribution phase.

In [177] the authors propose SmartVeh, as a secure and efficient message access control and authentication scheme in VCC. It uses a hierarchical, attribute-based encryption technique to achieve fine-grained and flexible message sharing, which ensures that vehicles whose persistent or dynamic attributes satisfy the access policies can access the broadcast message with equipped on-board units (OBUs). Additionally, Message authentication is enforced by integrating an attribute-based signature, which achieves message authentication and maintains the anonymity of the vehicles. In order to reduce the computations of the OBUs in the vehicles, they outsource the heavy computations of encryption, decryption and signing to a cloud server and road-side units. Nevertheless, we fear that their system is exposed to too many parties, and there are many privacy concerns. Additionally, having to deal with the complicated system of attribute distributions comes with computational overhead and communication cost. Relying on a cloud server for doing these computations is not always practical, and may present a new attack surface.

The authors in [178] propose a biometric and elliptic curve cryptography (ECC)-assisted authentication framework for VCC. They claim that their presented framework obtains most of security features and attributes for secure communication in the presence of active and passive attackers. Further, They provide formal security model and its proof, which is based on random oracle model. Also, they compare the performance of their protocol with similar frameworks in the same environment in terms of communication and computation overheads.

In [179] the researchers present an integrated Authentication and Key Agreement (AKA) framework for vehicular clouds. They integrated the single-server 3-factor AKA protocol and the non-interactive identity-based key establishment protocol, and evaluated its performance based on a simulated experimental platform. The authors in [180] discussed the construction of the authenticated session key agreement protocol for vehicular cloud. They claimed that their proposed framework also maintains the anonymity of participating vehicles. In Table 5.1 we briefly discuss and comment on the ideas mentioned in this section, and our critique of these techniques in comparison to our proposed scheme.

5.7 Summary

AVCC is a new technology and still under investigation. There are some security issues and privacy concerns to be addressed specially when dealing with AVs. More importantly, there is a need to securely authenticate parties trying to access these platforms in a privacy-preserving manner. In this chapter, we proposed a security scheme to protect AVCC participants and users against a multitude of cyber-attacks. We considered situations where attackers are honest-but-curious, whether they are attacking from the inside or eaves-

dropping from the outside. Our authentication scheme utilizes the efficiency of function encryption technique which provides protection against known-cihpertext attack models. The experiments demonstrate that our technique has low overhead in terms of communication and computations and efficiently scalable, which makes suitable and applicable for AVCC platforms.

Chapter 6

PRIVACY-PRESERVING BIOMETRIC-BASED AUTHENTICATION SCHEME FOR ELECTRIC VEHICLES CHARGING SYSTEM

6.1 Introduction

Cyber-attacks, privacy leaks, and other security threats are increasingly making national headlines each passing day. These threats are ever-evolving, posing real concerns regarding most emerging smart systems' safety, reliability, and operability. As the popularity of electric vehicles (EVs) increases, the demand for EV charging stations increases. Thus, implementing a secure and trustworthy authentication system is an eventuality for these stations to reduce the misuse of the charging stations and increase their spread. This work leverages human biometrics in terms of iris recognition along with a lightweight yet efficient encryption mechanism to provide a privacy-preserving robust authentication scheme. Biometric qualities are used to properly verify a person's identity based on physiological (finger, iris, hand, face) and behavioral (gait, signature, voice) characteristics. The iris recognition system is the most reliable and popular biometric technology.

In particular, iris recognition requires less computational intensive power than face recognition software. As well as has a better false accept rate than fingerprint recognition systems, and the left and right eye textures are unique from each other and will not match [181]. Many iris databases exist in the literature, which we have plenty of them to choose from, which were studied in [182].

That said, recent research shows that the integrity of iris data can be negatively impacted by new cyber-attacks like spoofing and manipulation attacks that aim to compromise and/or alter iris information. For example, an anti-spoofing security mechanism is used to detect these attacks by measuring the frequency of the valley and different ridge parameters. However, such a technique is not practical for real-time applications like iris recognition authentication due to its design complexity and computational overhead. In addition, storing sensitive biometric data becomes a significant issue, and great care is needed when keeping this type of information because it uniquely identifies a person. Companies have to be wary of attackers constantly trying to break into their systems and be diligent in keeping their systems secure. A simple trick to secure the Iris biometric data is to scramble or reorganize blocks of pixels in a captured iris image in a way only the company knows [183].

Our scheme's main objective is to defend against multiple types of new cyber-attacks such as spoofing, the man in the middle, and fraudulent and non-authorized access attempts. These types of attacks can be solved using numerous techniques that enhance and secure biometric data used in the EV charging station authentication system. A lightweight encryption algorithm, k-Nearest Neighbor (KNN) encryption, is implemented to secure the iris data at the charging station and provide a searchable encryption technique to use at the

server to verify the authenticity of the users. This design was created to be non-intrusive, have a low cost, produce high-performance results, and maintain low latency.

In this chapter, we develop a symmetric-key searchable encryption scheme for multi-data-owner and multi-user settings suitable for the EVs charging system authentication, where data (or iris images) are uploaded from multiple owners (charging companies), and multiple users (charging stations) search the data. The user who registers with any charging company can charge his/her EV from any charging station. We cover the cases where the user cannot find the charging stations under the control of his/her charging company. Using existing searchable encryption schemes (designed for single-data-owner and multi-user settings) for the charging system will require that each charging station has its own set of keys and encrypts each iris data vector using the keys from each of the charging companies. As a result, there are many calculation and communication overheads involved, and managing the keys becomes challenging.

Our authentication scheme enables each charging company and station to use a single key and encrypt the iris data vectors just once while allowing the server to match the encrypted vectors. Additionally, a limitation of existing symmetric-key searchable encryption techniques [16–18] is that anyone who receives the encrypted vectors, for example, by overhearing charging company and charging station communications, can compute the similarity score and deduce side information. Our scheme efficiently solves this challenge, which requires the cloud server to search over encrypted data using its secret key. Additionally, our system has a low overhead. Therefore, it will allow for a fast response such that user identification and verification are made quickly and efficiently with minimum wait time for the user.

The remaining sections of this chapter are organized as follows. In Section 6.2, we illustrate the system models and our intended design goals. Also, Section 6.3 demonstrates the proposed scheme. Then, in Section 6.4, we analyze the privacy aspects of our scheme. Additionally, we present the experimental results and performance evaluation in Section 6.5. We discuss some of the related literature in Section 6.6. Finally, we summarize the chapter in Section 6.7.

6.2 System Models

6.2.1 Network Model

As shown in Fig. 6.1, the considered network model consists of four main entities: charging companies (CCs), charging stations (CSs), cloud server, and an offline trusted authority (TA). The TA should disseminate a unique secret key to the cloud server, each CC, and each CS. Charging companies encrypt their users' iris data and upload the encrypted data (called indices) to the cloud server to use it in the authentication process. The CS should create an encrypted authentication query (called trapdoor) with the user's iris data, who wants to charge his/her EV and send the encrypted query to the server to authenticate the user. The cloud server should use the encrypted authentication query to look up the indices by computing a similarity score between them and then return an authentication result for usage at the CS. The server doesn't have to know any sensitive information about the queried

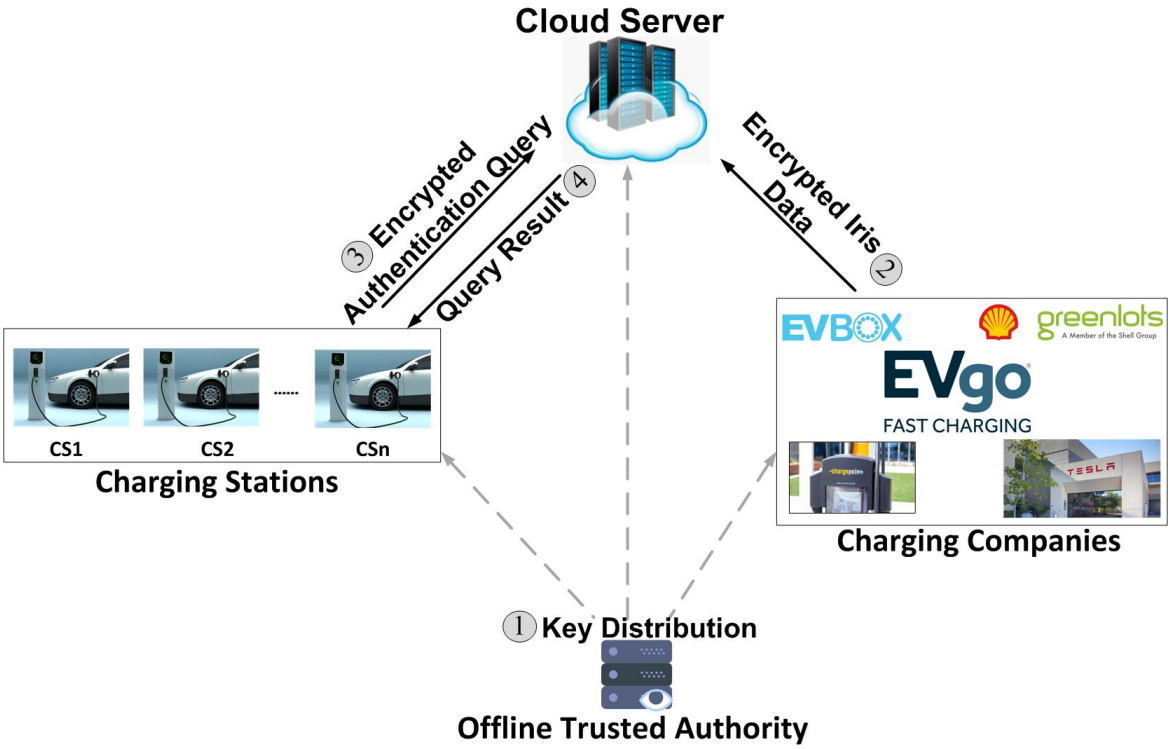

***Figure 6.1*:** The considered network model.

users.

6.2.2 Threat Model

External eavesdroppers and internal attackers, such as charging companies, charging stations, and the cloud server, are all possible. In our scheme, attackers are considered "honest-but-curious," which means they carry out the proposed scheme honestly and do not intend to disrupt the scheme's correct operation. They are, nevertheless, interested in sensitive information such as encrypted indices or authentication queries. We consider the Known-Ciphertext attack model in particular. The attacker in this model has access to only searchable encrypted iris indices and authentication queries supplied by charging companies and charging stations, respectively.

6.3 Proposed Scheme

In this section, we explain in details the proposed scheme. Table 6.1 gives the main notations used in the chapter.

6.3.1 System Initialization

First, the trusted authority runs the following set of oracles sequentially to initialize the system:

1) *SystemSetup* $\left(1^k\right) \rightarrow \mathcal{TAK}$. The system setup algorithm takes the security parameter 1^k as an input and outputs the TA secret key $(\mathcal{TAK})$, where, $\mathcal{TAK} = \{F, L_1, L_2, H_1, ..., H_8\}$, F is a random binary vector of length k, and a set of random invertible matrices $\in \mathbb{R}^{k \times k}$. k is the size of the iris data vector.

2) *KeyGenCloud* $(\mathcal{TAK}) \rightarrow \mathcal{CK}$. The TA creates the cloud server secret key $(\mathcal{CK})$, where $\mathcal{CK} = B$, B is an invertible random matrix $\in \mathbb{R}^{k \times k}$.

3) *KeyGenChargingCompany* $(\mathcal{CC}_i, \mathcal{TAK}) \rightarrow \mathcal{CCK}_i$. For each charging company in the system with identity $\mathcal{CC}_i$, the TA creates a charging company secret key $(\mathcal{CCK}_i)$ as

$$\begin{aligned} \mathcal{CCK}_i = \{&F,\ BH_1^{-1}W_i,\ BH_2^{-1}X_i,\ BH_3^{-1}W_i,\ BH_4^{-1}X_i,\\ &BH_5^{-1}Y_i,\ BH_6^{-1}Z_i,\ BH_7^{-1}Y_i,\ BH_8^{-1}Z_i\} \end{aligned} \tag{6.1}$$

where $\{W_i, X_i, Y_i, Z_i\}$ are random matrices $\in \mathbb{R}^{k \times k}$ such that $W_i + X_i = L_1^{-1}$, and $Y_i + Z_i = L_2^{-1}$.

4) *KeyGenChargingStation* $(\mathcal{CS}_x, \mathcal{TAK}) \rightarrow \mathcal{CSK}_x$. For charging station with ID $(\mathcal{CS}_x)$, the TA creates a charging station secret key $(\mathcal{CSK})$ as follows.

$$\begin{aligned} \mathcal{CSK}_x = \{&F,\ M_xH_1,\ M_xH_2,\ N_xH_3,\ N_xH_4,\\ &O_xH_5,\ O_xH_6,\ P_xH_7,\ P_xH_8\} \end{aligned} \tag{6.2}$$

where $\{M_x, N_x, O_x, P_x\}$ are random matrices $\in \mathbb{R}^{k \times k}$ such that $M_x + N_x = L_1$, and $O_x + P_x = L_2$

6.3.2 User Registration

When a new user wants to use this system, the user will first register an account with one of the CCs. The CC will take an iris image of its customers. For each customer, the CC will use the KNN encryption technique to encrypt the customer's iris data by using its secret key and uploading the encrypted data to the cloud server. The encrypted data includes n indices of encrypted data where n is the total number of customers. Every CC will do the same with its customers and upload the encrypted iris data of its customers to the cloud server to be used in the authentication process.

6.3.3 Encrypting Iris Data

Each charging company $(\mathcal{CC}_i)$ builds a set of indices $\mathcal{I}_i = \left\{I_{i,1}, I_{i,2}, \ldots, I_{i,n}\right\}$ extracted from the company customers' iris images. The index is created by invoking oracle *CreateIndex*()
.

CreateIndex $\left(I_{i,j}, \mathcal{CCK}_i\right) \rightarrow E_{I_{i,j}}$. For a customer iris data vector $I_{i,j}$, $\mathcal{CC}_i$ uses the secret F to split $I_{i,j}$ into two column vectors I'_{ij} and I''_{ij} of the same size, as follows. If the b^{th} bit of F is zero, $I'_{ij}(b)$ and $I''_{ij}(b)$ are set similar to $I_{ij}(b)$, while if it is one, $I'_{ij}(b)$ and $I''_{ij}(b)$ are set to two random numbers such that their summation is equal to $I_{ij}(b)$. Finally, $\mathcal{CC}_i$ uses its secret key $\mathcal{CCK}_i$ to compute the iris data index $E_{I_{i,j}}$ as

$$\begin{aligned} E_{I_{i,j}} = \Big[&BH_1^{-1}W_iI'_{ij}; BH_2^{-1}X_iI'_{ij}; BH_3^{-1}W_iI'_{ij}; BH_4^{-1}X_iI'_{ij};\\ &BH_5^{-1}Y_iI''_{ij}; BH_6^{-1}Z_iI''_{ij}; BH_7^{-1}Y_iI''_{ij}; BH_8^{-1}Z_iI''_{ij}\Big] \end{aligned} \tag{6.3}$$

***Table 6.1*:** Main Notations We Used in Our Technique.

Notation	Description
$\mathcal{TAK}$	Trusted authority secret key
F	Secret binary vector
$\{L_1, L_2, H_1, ..., H_8\} \in \mathbb{R}^{k \times k}$	Server secret matrices
$\mathcal{CK}$	Cloud server secret key
$\mathcal{CC}_i$	Charging Company i
$\mathcal{CCK}_i$	$\mathcal{CC}_i$'s secret key
$\{W_i, X_i, Y_i, Z_i\} \in \mathbb{R}^{k \times k}$	Random matrices for $\mathcal{CC}_i$
$\mathcal{CS}_x$	Charging Station x
$\mathcal{CSK}_x$	$\mathcal{CS}_x$'s secret key
$\{M_x, N_x, O_x, P_x\} \in \mathbb{R}^{k \times k}$	Random matrices for $\mathcal{CS}_x$

where $E_{I_{i,j}}$ is a column vector of size $8k$. Then submit it to the cloud server.

6.3.4 Submit Authentication Query

Each charging station ($\mathcal{CS}_x$) will use its corresponding secret key $\mathcal{CSK}_x$ delivered from the $\mathcal{TA}$ to encrypt an authentication query for each user wants to charge, in order to ensure if this user is authenticated or if the request is from a malicious entity without disclosure of user's information to the CC or giving the server a chance to know any sensitive information about the user. First, the CS will take a picture of the user's face by using an attached iris camera and generate an image matrix (Q). Then the CS will perform the required processing to extract the iris data features after removing the eyelashes and other such noise and finally generating the user's iris data vector (T).

Each $\mathcal{CS}_x$ generates encrypted authentication query by invoking oracle $CreateTrapdoor()$.

$CreateTrapdoor(T, \mathcal{CSK}_x) \rightarrow C_T$. Given the user's iris data vector T, $\mathcal{CS}_x$ uses F to split T to two random row vectors t' and t'' of the same size. The splitting method is described as follows. If the b^{th} bit of F is one, $t'(b)$ and $t''(b)$ are set similar to $t(b)$, while if it is zero, $t'(b)$ and $t''(b)$ are set to two random numbers such that their summation is equal to $t(b)$. Then, $\mathcal{CS}_x$ uses his secret key $\mathcal{CSK}_x$ to create the trapdoor C_T

$$C_T = \Big[t'M_xH_1,\ t'M_xH_2,\ t'N_xH_3,\ t'N_xH_4,\ t''O_xH_5,\ t''O_xH_6,\ t''P_xH_7,\ t''P_xH_8\Big] \tag{6.4}$$

where C_T is a row vector of size $8k$. Then submit it to the cloud server.

6.3.5 Matching Iris Encrypted Data

Finally, the cloud server will begin searching over encrypted iris data by calculating the dot product operation between the trapdoor came from the CS with all the indices inside its

database in oracle *Match*. The cloud server will be using KNN encryption scheme in order to search the data without ever decrypting the data which adds an extra layer of security. The cloud server will then use the outcome of the search to validate the charging station's authentication query.

$Match\left(\mathcal{CK}, E_{I_{i,j}}, C_T\right) \rightarrow AuthenticationResult$. In this oracle, the cloud server should first use its secret B^{-1} to remove B from $E_{I_{i,j}}$ to obtain $\bar{E}_{I_{i,j}}$, where

$$\begin{aligned}\bar{E}_{I_{i,j}} = \Big[&H_1^{-1}W_iI'_{ij};\ H_2^{-1}X_iI'_{ij};\ H_3^{-1}W_iI'_{ij};\ H_4^{-1}X_iI'_{ij};\\ &H_5^{-1}Y_iI''_{ij};\ H_6^{-1}Z_iI''_{ij};\ H_7^{-1}Y_iI''_{ij};\ H_8^{-1}Z_iI''_{ij}\Big]\end{aligned} \tag{6.5}$$

Then, it can simply compute the similarity score between the trapdoor C_T and the index $\bar{E}_{I_{i,j}}$ by dot product operation ($C_T \odot \bar{E}_{I_{i,j}}$), where $\odot$ denotes dot product. If the query coming from the CS matches with one of the stored indices inside the cloud server, the server will send a positive confirmation to the CS in order to indicate that the user is a legitimate user, otherwise, the server will send a negative confirmation to deny the access for this malicious user.

Theorem 6.3.1. *Using our scheme, the cloud server can measure the similarity score of the indices and the trapdoors.* This can be done by computing $C_T \odot E_{I_{i,j}}$, as follows.

$$\begin{aligned}C_T \odot \bar{E}_{I_{i,j}} &= t'M_xW_iI'_{ij} + t'M_xX_iI'_{ij} + t'N_xW_iI'_{ij} +\\ &\quad t'N_xX_iI'_{ij} + t''O_xY_iI''_{ij} + t''O_xZ_iI''_{ij} +\\ &\quad t''P_xY_iI''_{ij} + t''P_xZ_iI''_{ij}\\ &= t'(M_x+N_x).(W_i+X_i)I'_{ij} +\\ &\quad t''(O_x+P_x).(Y_i+Z_i)I''_{ij}\\ &= t'L_1L_1^{-1}I'_{ij} + t''L_2L_2^{-1}I''_{ij}\\ &= T \odot I_{i,j}\end{aligned}$$

6.4 Privacy Analysis

In this section, we explain how the proposed scheme addresses each item in our design goals.

1. *Authentication Queries Unlinkability.* In the multi-data-owner approach presented in [184], named SRMSM, transmitting the same trapdoor several times results in the identical ciphertext that breaches the trapdoor unlinkability requirement. On the contrary, our scheme ensures that transmitting the same trapdoor many times results in different ciphertexts. Another advantage of the proposed scheme over SRMSM is that even if an eavesdropper intercepts the server's messages, he will be unable to determine the similarity scores, as the server's secret key $\mathcal{CK}$ is required.

2. *Index and Authentication Query Confidentiality.* The server should be able to search for users' authentication without learning any sensitive information. Index and authentication queries confidentiality is achieved by encrypting the data from the CSs and CCs. Which means that the data is never uploaded to the server in plaintext format. The cloud server also searches exclusively over encrypted data which means that the data is never decrypted on the server which preserves privacy by virtue that the raw data never appear on the server.

3. *Authentication Query search over Encrypted Iris data from Several Charging Companies.* The proposed scheme can use the encrypted authentication query from a CS to search over encrypted iris data stored by different CCs. In our scheme, the CSs belong to different charging companies, do not need to share keys with other companies to search over their encrypted iris data. The CS can use the same key from the TA to search all the iris data from different CCs.

4. *Scalability and Efficiency.* The proposed scheme can quickly search through a large number of indices coming from multiple CCs to reply to CS's query as it does only a dot product operation. Additionally, the communication and computational overhead is reasonable compared to the exciting schemes.

6.5 Experiments and Performance Evaluation

6.5.1 Experiment Setup and Evaluation Metrics

To evaluate the proposed scheme, we implemented it using Python and a server with an Intel® Core™ i5-1035G1 CPU @ 1.19 GHz and 12 GB of RAM running a Windows 10 Home Edition operating system. In our experiments, we used the Indian Institute of Technology Delhi (IITD) iris image dataset [185], of total size of 2240 files which is publicly available for download from the IITD website [186]. Each file is a grey scale image of a human iris. We pre-processed the dataset using *scikit-learn* Python library [187] and extracted the features vector as a unit vector. The dot product of a unit vector with exactly itself will produce 1 as an output [188], which we will use in our matching step at the cloud server.

The communication and the computation overhead of the proposed scheme were studied. Also, we thoroughly analyzed search time and compared our results with the work of Rajasekar et al [189]. Finally, we show the scalability of our system with increased number of users. All the experiments were run different iris data sample sizes and average results are reported.

6.5.2 Experiment Results

Communication Overhead

The main advantage of our technique is its simplicity. We tried to make it as lightweight as possible to suit the needs of resource-limited charging stations. That said, original iris data files had a resolution of 320 *240 pixels, meaning that they had a size around 230 KB.

***Table 6.2*:** Communication Overhead.

Original Iris Image Size	$320 * 240 * 3$
Original Iris Image Size in Bytes	230 KB
Pre-processed Iris Image Size	$300 * 1$
Encrypted Iris Image Size	$2400 * 1$
Encrypted Iris Image Size in Bytes	19 KB

***Table 6.3*:** Computation Overhead.

Operation	**Average Time**
$T_{CCKeyGen}$	11.8 $msec$
$T_{CSKeyGen}$	10.9 $msec$
T_{Enc}	1.1 $msec$
T_{dot}	3.9 μsec

Therefore, these original iris images were compressed, normalized, and flattened during the pre-processing phase to get an index of size 300*1. After the encryption process at either side (CC or CS) using either the $\mathcal{CCK}_i$ or the $\mathcal{CSK}_x$ it will grow up to an index of 2400 * 1, of Python's $float64$ data type, which is represented in 8 bytes. This will finally amount to 19 KB only being communicated for each index or authentication query. These findings are shown in Table 6.2.

Computation Overhead

Table 6.3 gives the times required for the cryptographic operations used in our scheme. Namely they are, $T_{CCKeyGen}$, $T_{CSKeyGen}$, T_{Enc} and T_{Dot}. Which respectively denote the times required for key generation for the charging company CC_i and the charging station CS_x, the time required for trapdoor encryption, and the time required to get the dot product of iris indices and trapdoors. Our findings show that all-important processes were in the order of *msecs* and the core of our matching process (the dot product operation) was in the order of μsec. This proves that our technique is lightweight even with a moderate computing device, making it quite efficient and practical when used with more powerful cloud servers, handling numerous authentication requests simultaneously.

Encryption Scalability

We studied the performance of our system in gradually increasing usage scenarios. We poked into the IITD iris dataset with different subsets with different sample sizes, increasing the sample size by 250 in each step. We measured the time required to encrypt the entirety of the subset and plotted the results in Figure 6.2. The Figure shows that our system's performance scales linearly with the size of the dataset, corresponding to the increasing number of system users. The encryption times were within reasonable margins, which is a practically acceptable cost for privacy.

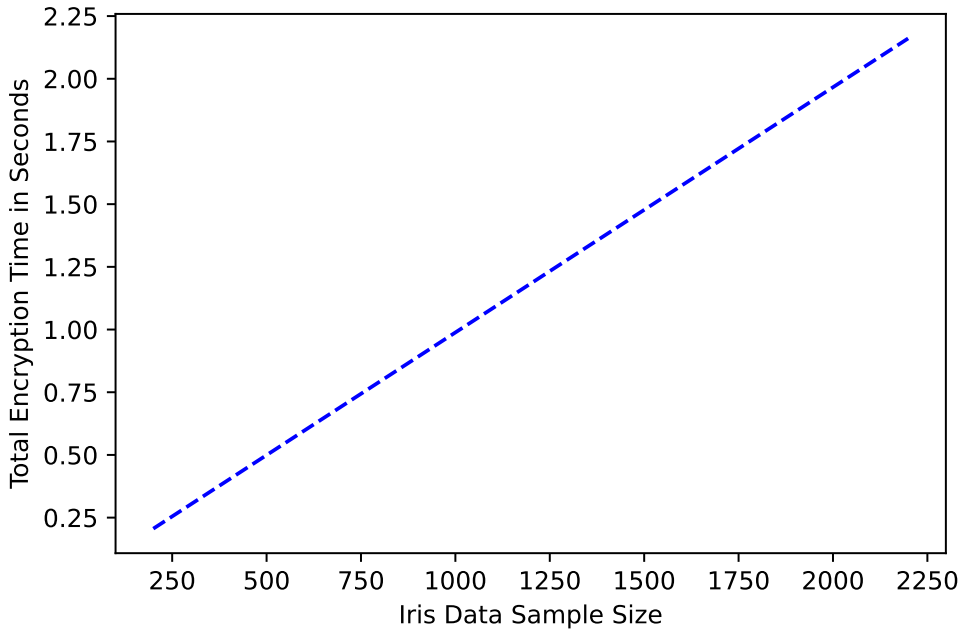

***Figure 6.2*:** Encryption times across different iris sample sizes.

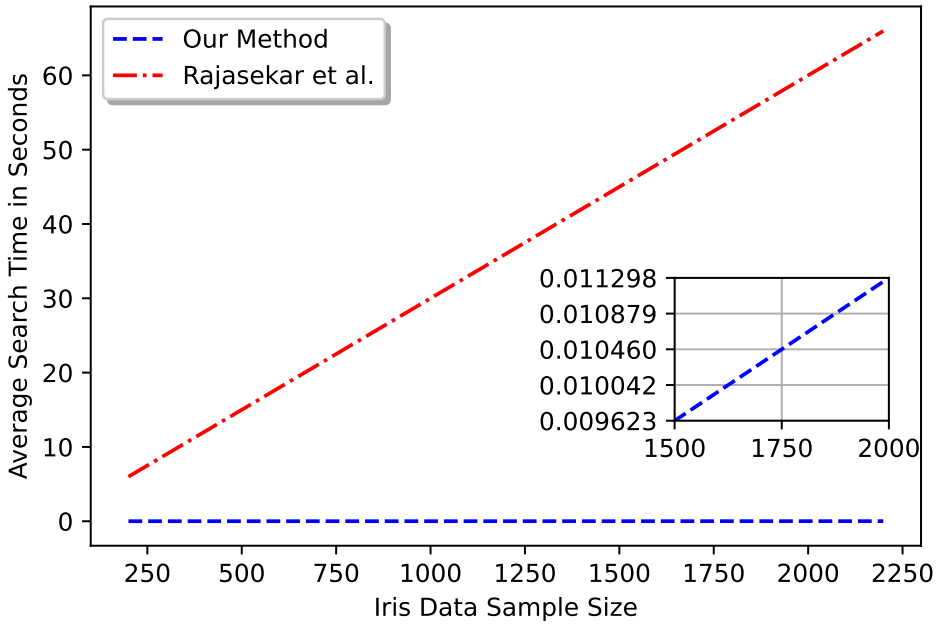

***Figure 6.3*:** Comparing our average search times across different iris sample sizes with the work of Rajasekar et al.

Search Time Analysis

For each iteration, we tested a number of queries equal to half of the entire sample subset size at that particular step and averaged over the resulting times. The results show the average time required for the server to search a subset of encrypted iris data with the query by performing a dot product operation between the trapdoor and each stored index. If an

index was found to be similar, then the user is indeed authenticated, and this concludes the search process. As we said before in Table 6.3, this dot product operation was in the order of μsec; consequently, these average search times are also in the order of *msec*, again with our generic computer, scaling linearly with the number of searched indices (iris data sample size). In Figure 6.3, we compared our average search time with the result of Rajasekar et [189]. In their paper, Rajasekar et al. claim that their proposed technique with fuzzy logic achieved a recognition time (maps to our search time) of 3 *sec* with a sample size of 100 iris data. For that same sample size, we achieved an average of less than 2 *msecs*, which is 10^3 smaller in order of magnitude. And as with the size of the searchable data-set grows, the search time is still in the order of *msecs*, as shown in the zoomed subfigure in Figure 6.3. This proves that our system is more efficient and lightweight than this existing technique.

In summary, our experimental results proved that our proposed scheme is relatively lightweight and suitable for application in EV charging systems with mathematically proved privacy and unlinkability goals.

6.6 Related Work

The work of [190] surveyed some of the most recent biometric authentication techniques and how they fair against each other. Human iris biometrics is considered one of the most reliable methods to distinguish users as it is unique, requires no contact, perfectly stable. Moreover, it doesn't damage over time like fingerprints which can easily become damaged. Also, it cannot be forgotten or easily taken from the person like id cards, badges, etc. [191].

Iris recognition is being directly used in the Qualcomm Snapdragon 835 SoCs, enabling all sorts of new applications with iris recognition such as Face Unlock on Android and Apple phones. For example, the authors in [192] discussed system requirements for constraint-free iris image acquisition in smartphones for iris authentication purposes. Nonetheless, iris-based authentication schemes can be used on many different types of security systems such as mobile phones, banking systems, border control, car systems, etc. [193, 194].

In most systems, such as the one proposed in this paper, scanned iris biometric data of individuals are compared to pre-stored iris biometric data that is located in the cloud [195]. The authors in [196] studied the problem of iris code storage in vast databases of iris templates. They used a non-homogenous K-d tree structure to improve retrieval accuracy and efficiency.

In spite of that, biometric authentication may be susceptible to spoofing attacks. Spoofing is a process used to deceive biometric authentication systems. There are several known ways on how to spoof biometric systems, [197, 198]. One way to detect spoofing biometrics through printed images, textured contact lenses, and synthetic creations of iris images is using a detection method called DESIST [199]. DESIST is composed of a multitude of algorithms that are used to detect specific types of spoofing.

The authors in [189] combined iris recognition with the use of cryptography to combat identity fraud attacks. They claimed to propose a computationally lightweight and accurate technique using hyper-elliptic curve cryptography (HECC). They used a fuzzy logic algorithm to enhance recognition accuracy and speed. Nevertheless, we showed that our scheme outperforms their technique in terms of recognition time and computational overhead.

6.7 Summary

Electric vehicles are gaining popularity every day. Consequently, electric charging systems are becoming more and more ubiquitous. Hence, there is a need to access these stations in a secure and trusted manner. This chapter presented an authentication scheme using iris recognition. The proposed technique utilizes the efficiency of KNN encryption techniques which provides confidentially for users even against honest-but-curious service providers. Moreover, the experimental results on the IITD iris dataset showed that our system has low communication cost and reasonable computational overhead compared to the existing schemes.

Chapter 7

CONCLUSION AND FUTURE WORK

AI is gradually changing the technological services we use in our everyday life. Among which is smart transportation, coined AVs. These new machines are smart, and they know a lot of information about their users. They utilize this information to provide new attractive offerings and protect passengers from imminent dangers among other things. But the problem is that the collection of all that data presented new security and privacy attack surfaces that need to be appropriately handled.

In this work, we studied these security and privacy issues and their mitigation strategies in a layer-based model. Then, we focused on security and privacy issues on Autonomous Vehicles Cloud Computing (AVCC) platforms. We proposed a compiler-based software protection technique specifically tailored for the AVCC with two delivery models. Then, we presented an encrypted authentication system to handle cloud usage and participation requests form clients and AVs respectively. Then, we presented a more light weight authentication technique more suitable for vehicle charging.

In summary, we discussed open research directions, showed some gaps in existing literature and paved the way for future work. Hence, we plan to further improve the security and privacy of the AVCC platform, perhaps focus on resource sharing and aggregation. We also plan to evaluate our ideas in more realistic environments and against a wider range of attack scenarios.

BIBLIOGRAPHY

[1] Center of Disease Control (CDC). Road Traffic Injuries and Deaths. Accessed Mar. 30, 2022. [Online]. Available: https://www.cdc.gov/injury/features/global-road-safety/index.html

[2] J. Janai, F. Güney, A. Behl, A. Geiger *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020.

[3] M. Zangui, Y. Yin, and S. Lawphongpanich, "Sensor location problems in path-differentiated congestion pricing," *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 217–230, 2015.

[4] S. B. Raut and L. Malik, "Survey on vehicle collision prediction in vanet," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 2014, pp. 1–5.

[5] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE, 2014, pp. 241–246.

[6] H. Zhou, W. Xu, J. Chen, and W. Wang, "Evolutionary v2x technologies toward the internet of vehicles: challenges and opportunities," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 308–323, 2020.

[7] D. J. Glancy, "Privacy in autonomous vehicles," *Santa Clara L. Rev.*, vol. 52, p. 1171, 2012.

[8] S. Berghaus and A. Back, "Requirements elicitation and utilization scenarios for in-car use of wearable devices," in *2015 48th Hawaii International Conference on System Sciences*. IEEE, 2015, pp. 1028–1037.

[9] S. Hess, A. Meschtscherjakov, T. Ronneberger, and M. Trapp, "Integrating mobile devices into the car ecosystem: tablets and smartphones as vital part of the car," in *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2011, pp. 210–211.

[10] V. L. Thing and J. Wu, "Autonomous vehicle security: A taxonomy of attacks and defences," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2016, pp. 164–170.

[11] S. Karnouskos and F. Kerschbaum, "Privacy and integrity considerations in hyperconnected autonomous vehicles," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 160–170, 2017.

[12] R. W. Coutinho and A. Boukerche, "Guidelines for the design of vehicular cloud infrastructures for connected autonomous vehicles," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 6–11, 2019.

[13] J. Cui, L. S. Liew, G. Sabaliauskaite, and F. Zhou, "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles," *Ad Hoc Networks*, vol. 90, p. 101823, 2019.

[14] C. P. García, S. ul Hassan, N. Tuveri, I. Gridin, A. C. Aldaya, and B. B. Brumley, "Certified side channels," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2021–2038.

[15] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford university Stanford, 2009, vol. 20, no. 9.

[16] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, Jan 2014.

[17] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN Computation on Encrypted Databases," *Proceddings of the 35th SIGMOD International Conference on Management of Data*, pp. 139–152, 2009.

[18] C. Yang, W. Zhang, J. Xu, J. Xu, and N. Yu, "A fast privacy-preserving multi-keyword search scheme on cloud data," *Proceedings of the 2012 International Conference on Cloud Computing and Service Computing, CSC 2012*, pp. 104–110, 2012.

[19] B. A. Forouzan, *TCP/IP protocol suite*. McGraw-Hill, Inc., 2002.

[20] H. Zimmermann, "Osi reference model-the iso model of architecture for open systems interconnection," *IEEE Transactions on communications*, vol. 28, no. 4, pp. 425–432, 1980.

[21] J. Joy and M. Gerla, "Internet of vehicles and autonomous connected car-privacy and security issues," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[22] L. Adacher, A. Gemma, and G. Oliva, "Decentralized spatial decomposition for traffic signal synchronization," *Transportation Research Procedia*, vol. 3, pp. 992–1001, 2014.

[23] C. Wuthishuwong and A. Traechtler, "Coordination of multiple autonomous intersections by using local neighborhood information," in *2013 international conference on connected vehicles and expo (ICCVE)*. IEEE, 2013, pp. 48–53.

[24] P. Gora, "Simulation-based traffic management system for connected and autonomous vehicles," in *Road Vehicle Automation 4*. Springer, 2018, pp. 257–266.

[25] P. Wagner, "Traffic control and traffic management in a transportation system with autonomous vehicles," in *Autonomous Driving*. Springer, 2016, pp. 301–316.

[26] I. Rubin, A. Baiocchi, Y. Sunyoto, and I. Turcanu, "Traffic management and networking for autonomous vehicular highway systems," *Ad Hoc Networks*, vol. 83, pp. 125–148, 2019.

[27] S. A. Fayazi and A. Vahidi, "Vehicle-in-the-loop (vil) verification of a smart city intersection control scheme for autonomous vehicles," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2017, pp. 1575–1580.

[28] F. Ashtiani, S. A. Fayazi, and A. Vahidi, "Multi-intersection traffic management for autonomous vehicles via distributed mixed integer linear programming," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 6341–6346.

[29] S. El Hamdani and N. Benamar, "Autonomous traffic management: Open issues and new directions," in *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*. IEEE, 2018, pp. 1–5.

[30] S. Chuprov, I. Viksnin, I. Kim, L. Reznikand, and I. Khokhlov, "Reputation and trust models with data quality metrics for improving autonomous vehicles traffic security and safety," in *2020 IEEE Systems Security Symposium (SSS)*. IEEE, 2020, pp. 1–8.

[31] K. Rabieh, M. M. Mahmoud, and M. Younis, "Privacy-preserving route reporting scheme for traffic management in vanets," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 7286–7291.

[32] R. Hussain and H. Oh, "On secure and privacy-aware sybil attack detection in vehicular communications," *Wireless personal communications*, vol. 77, no. 4, pp. 2649–2673, 2014.

[33] R. Reshma, T. Ramesh, and P. Sathishkumar, "Security situational aware intelligent road traffic monitoring using uavs," in *2016 international conference on VLSI systems, architectures, technology and applications (VLSI-SATA)*. IEEE, 2016, pp. 1–6.

[34] S. Ucar, S. C. Ergen, and O. Ozkasap, "Ieee 802.11 p and visible light hybrid communication based secure autonomous platoon," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8667–8681, 2018.

[35] J. Ploeg, E. Semsar-Kazerooni, G. Lijster, N. van de Wouw, and H. Nijmeijer, "Graceful degradation of cooperative adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 488–497, 2014.

[36] S. Dadras, R. M. Gerdes, and R. Sharma, "Vehicular platooning in an adversarial environment," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 167–178.

[37] B. DeBruhl, S. Weerakkody, B. Sinopoli, and P. Tague, "Is your commute driving you crazy? a study of misbehavior in vehicular platoons," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015, pp. 1–11.

[38] R. M. Gerdes, C. Winstead, and K. Heaslip, "Cps: an efficiency-motivated attack against autonomous vehicular transportation," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 99–108.

[39] D. D. Dunn, *Attacker-induced traffic flow instability in a stream of automated vehicles*. Utah State University, 2015.

[40] D. D. Dunn, S. A. Mitchell, I. Sajjad, R. M. Gerdes, R. Sharma, and M. Li, "Regular: Attacker-induced traffic flow instability in a stream of semi-automated vehicles," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 499–510.

[41] W. Jiang, H. Li, S. Liu, X. Luo, and R. Lu, "Poisoning and evasion attacks against deep learning algorithms in autonomous vehicles," *IEEE transactions on vehicular technology*, vol. 69, no. 4, pp. 4439–4449, 2020.

[42] "A collaborative approach for improving the security of vehicular scenarios: The case of platooning," *Computer Communications*, vol. 122, pp. 59–75, 2018.

[43] F. Gonçalves, B. Ribeiro, V. Hapanchak, S. Barros, O. Gama, P. Araújo, M. J. Nicolau, B. Dias, J. Macedo, A. Costa *et al.*, "Secure management of autonomous vehicle platooning," in *Proceedings of the 14th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, 2018, pp. 15–22.

[44] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, and R. L. Cigno, "Plexe: A platooning extension for veins," in *2014 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2014, pp. 53–60.

[45] M. Nabil, A. Sherif, M. Mahmoud, A. Alsharif, and M. Abdallah, "Efficient and privacy-preserving ridesharing organization for transferable and non-transferable services," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[46] N. D. Chan and S. A. Shaheen, "Ridesharing in north america: Past, present, and future," *Transport reviews*, vol. 32, no. 1, pp. 93–112, 2012.

[47] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[48] H. Xu, F. Ordóñez, and M. Dessouky, "A traffic assignment model for a ridesharing transportation market," *Journal of Advanced Transportation*, vol. 49, no. 7, pp. 793–816, 2015.

[49] E. Deakin, K. T. Frick, and K. M. Shively, "Markets for dynamic ridesharing? case of berkeley, california," *Transportation Research Record*, vol. 2187, no. 1, pp. 131–137, 2010.

[50] T. Canada. (2010) High occupancy vehicle lanes in canada. Accessed Mar. 30, 2022. [Online]. Available: https://tc.canada.ca/en/corporate-services/acts-regulations/canada-transportation-act-review-report.

[51] C. F. Daganzo and M. J. Cassidy, "Effects of high occupancy vehicle lanes on freeway congestion," *Transportation Research Part B: Methodological*, vol. 42, no. 10, pp. 861–872, 2008.

[52] E. Lutostanski. (2014) Carema offers discounted rates on Toll 183A. Accessed Mar. 30, 2022. [Online]. Available: https://communityimpact.com/austin/transportation/2014/02/17/carma-offers-discounted-rates-on-toll-183a-2/

[53] A. Sherif, A. Alsharif, J. Moran, and M. Mahmoud, "Privacy-preserving ride sharing organization scheme for autonomous vehicles in large cities," in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2017, pp. 1–5.

[54] A. Sherif, A. Alsharif, M. Mahmoud, and J. Moran, "Privacy-preserving autonomous cab service management scheme," in *Proceedings of the 3rd Africa and Middle East Conference on Software Engineering*, 2017, pp. 19–24.

[55] M. Li, L. Zhu, and X. Lin, "Efficient and privacy-preserving carpooling using blockchain-assisted vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4573–4584, 2018.

[56] L. Zhu, K. Gai, and M. Li, "Blockchain-enabled carpooling services," in *Blockchain Technology in Internet of Things*. Springer, 2019, pp. 75–91.

[57] M. Baza, N. Lasla, M. Mahmoud, G. Srivastava, and M. Abdallah, "B-ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Transactions on Network Science and Engineering*, 2019.

[58] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, 2016, pp. 270–282.

[59] L. P. Cox, "Truth in crowdsourcing," *IEEE Security & Privacy*, vol. 9, no. 5, pp. 74–76, 2011.

[60] N. Cheaz, A. Diaz, M. E. Head, and J. H. Kerr, "Delayed parking optimization of autonomous vehicles," Apr. 21 2020, uS Patent 10,625,733.

[61] S. J. Lauffer, J. P. Joyce, D. A. Gabor, S. Abbas, and R. Steven, "Electric parking brake for autonomous vehicles," Feb. 4 2020, uS Patent 10,549,731.

[62] B. Rech, S. Gläser, M. Engel, H.-J. Günther, T. Buburuzan, S. Kleinau, B. Lehmann, and J. Hartog, "Method for a data processing system for maintaining an operating state of a first autonomous vehicle and method for a data processing system for managing a plurality of autonomous vehicles," Mar. 31 2020, uS Patent 10,607,422.

[63] J. Ni, X. Lin, and X. Shen, "Toward privacy-preserving valet parking in autonomous driving era," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2893–2905, 2019.

[64] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic k-taa," in *International conference on security and cryptography for networks*. Springer, 2006, pp. 111–125.

[65] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.

[66] C. Huang, R. Lu, X. Lin, and X. Shen, "Secure automated valet parking: A privacy-preserving reservation scheme for autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 169–11 180, 2018.

[67] J. Chinrungrueng, U. Sununtachaikul, and S. Triamlumlerd, "A vehicular monitoring system with power-efficient wireless sensor networks," in *2006 6th International Conference on ITS Telecommunications*. IEEE, 2006, pp. 951–954.

[68] P. Colijn, L. A. Feenstra, J. S. Herbach, and K. Patterson, "Fleet management for autonomous vehicles," Jun. 25 2020, uS Patent App. 16/719,302.

[69] W. Xu, C. Yan, W. Jia, X. Ji, and J. Liu, "Analyzing and enhancing the security of ultrasonic sensors for autonomous vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5015–5029, 2018.

[70] J. Joy and M. Gerla, "Internet of vehicles and autonomous connected car-privacy and security issues," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[71] C.-M. Chen, B. Xiang, Y. Liu, and K.-H. Wang, "A secure authentication protocol for internet of vehicles," *Ieee Access*, vol. 7, pp. 12 047–12 057, 2019.

[72] B. Ying and A. Nayak, "Anonymous and lightweight authentication for secure vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 626–10 636, 2017.

[73] M. Wazid, P. Bagga, A. K. Das, S. Shetty, J. J. Rodrigues, and Y. H. Park, "Akm-iov: Authenticated key management protocol in fog computing-based internet of vehicles deployment," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8804–8817, 2019.

[74] J. Kang, R. Yu, X. Huang, and Y. Zhang, "Privacy-preserved pseudonym scheme for fog computing supported internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2627–2637, 2017.

[75] J. Kang, Z. Xiong, D. Niyato, D. Ye, D. I. Kim, and J. Zhao, "Toward secure blockchain-enabled internet of vehicles: Optimizing consensus management using reputation and contract theory," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2906–2920, 2019.

[76] M. M. Hossain, R. Hasan, and S. Zawoad, "Trust-iov: A trustworthy forensic investigation framework for the internet of vehicles (iov)." in *ICIOT*, 2017, pp. 25–32.

[77] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: architecture, protocols, and security," *IEEE internet of things Journal*, vol. 5, no. 5, pp. 3701–3709, 2017.

[78] C. Bala, W. Schuldzinski, M. Uhlmann, F. Pittroff, J. Lamla, M. Schuhen, M. Askari, S. Schürkmann, C. Wiencierz, U. Röttger *et al.*, *Beiträge zur Verbraucherforschung Band 9 Der vertrauende Verbraucher: Zwischen Regulation und Information*. Verbraucherzentrale NRW, 2020, vol. 9.

[79] J. Contreras-Castillo, S. Zeadally, and J. A. G. Ibañez, "Solving vehicular ad hoc network challenges with big data solutions," *IET Networks*, vol. 5, no. 4, pp. 81–84, 2016.

[80] M. H. Eiza and Q. Ni, "Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 45–51, 2017.

[81] S. Halder, A. Ghosal, and M. Conti, "Secure ota software updates in connected vehicles: A survey," *arXiv preprint arXiv:1904.00685*, 2019.

[82] S. M. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* IEEE, 2005, pp. 588–593.

[83] K. Mansour, W. Farag, and M. ElHelw, "Airodiag: A sophisticated tool that diagnoses and updates vehicles software over air," in *2012 IEEE International Electric Vehicle Conference*. IEEE, 2012, pp. 1–7.

[84] M. Steger, M. Karner, J. Hillebrand, W. Rom, C. Boano, and K. Römer, "Generic framework enabling secure and efficient automotive wireless sw updates," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[85] K. Mayilsamy, N. Ramachandran, and V. S. Raj, "An integrated approach for data security in vehicle diagnostics over internet protocol and software update over the air," *Computers & Electrical Engineering*, vol. 71, pp. 578–593, 2018.

[86] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "Blockchain: A distributed solution to automotive security and privacy," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 119–125, 2017.

[87] T. Zhang, H. Antunes, and S. Aggarwal, "Defending connected vehicles against malware: Challenges and a solution framework," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 10–21, 2014.

[88] W. Niu, X. Zhang, X. Du, T. Hu, X. Xie, and N. Guizani, "Detecting malware on x86-based iot devices in autonomous driving," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 80–87, 2019.

[89] W. Jiang, H. Li, S. Liu, X. Luo, and R. Lu, "Poisoning and evasion attacks against deep learning algorithms in autonomous vehicles," *IEEE transactions on vehicular technology*, vol. 69, no. 4, pp. 4439–4449, 2020.

[90] Y. Qian and N. Moayeri, "Design of secure and application-oriented vanets," in *VTC Spring 2008-IEEE Vehicular Technology Conference*. IEEE, 2008, pp. 2794–2799.

[91] C. A. Kerrache, C. T. Calafate, J.-C. Cano, N. Lagraa, and P. Manzoni, "Trust management for vehicular networks: An adversary-oriented overview," *IEEE Access*, vol. 4, pp. 9293–9307, 2016.

[92] R. G. Engoulou, M. Bellaïche, S. Pierre, and A. Quintero, "Vanet security surveys," *Computer Communications*, vol. 44, pp. 1–13, 2014.

[93] F. Al-Hawi, C. Y. Yeun, and M. Al-Qutayti, "Security challenges for emerging vanets," in *4th Int. Conf. Inf. Technol., Jordan, Amman*, 2009.

[94] M. Kassim, R. A. Rahman, and R. Mustapha, "Mobile ad hoc network (manet) routing protocols comparison for wireless sensor network," in *2011 IEEE International Conference on System Engineering and Technology*. IEEE, 2011, pp. 148–152.

[95] A. Nanda, D. Puthal, J. J. Rodrigues, and S. A. Kozlov, "Internet of autonomous vehicles communications security: overview, issues, and directions," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 60–65, 2019.

[96] M. N. Mejri, J. Ben-Othman, and M. Hamdi, "Survey on vanet security challenges and possible cryptographic solutions," *Vehicular Communications*, vol. 1, no. 2, pp. 53–66, 2014.

[97] H. Hasrouny, A. E. Samhat, C. Bassil, and A. Laouiti, "Vanet security challenges and solutions: A survey," *Vehicular Communications*, vol. 7, pp. 7–20, 2017.

[98] M. Azees, P. Vijayakumar, and L. J. Deborah, "Comprehensive survey on security services in vehicular ad-hoc networks," *IET Intelligent Transport Systems*, vol. 10, no. 6, pp. 379–388, 2016.

[99] M. Muhammad and G. A. Safdar, "Survey on existing authentication issues for cellular-assisted v2x communication," *Vehicular Communications*, vol. 12, pp. 50–65, 2018.

[100] M. Amoozadeh, A. Raghuramu, C.-N. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 126–132, 2015.

[101] K. M. A. Alheeti and K. McDonald-Maier, "An intelligent security system for autonomous cars based on infrared sensors," in *2017 23rd International Conference on Automation and Computing (ICAC)*. IEEE, 2017, pp. 1–5.

[102] S. Boumiza and R. Braham, "Intrusion threats and security solutions for autonomous vehicle networks," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2017, pp. 120–127.

[103] P. Kleberger, N. Nowdehi, and T. Olovsson, "Towards designing secure in-vehicle network architectures using community detection algorithms," in *2014 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2014, pp. 69–76.

[104] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.

[105] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, vol. 4. San Francisco, 2011, pp. 447–462.

[106] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011, p. 20.

[107] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "Libra-can: a lightweight broadcast authentication protocol for controller area networks," in *International Conference on Cryptology and Network Security*. Springer, 2012, pp. 185–200.

[108] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *International Conference on Information Security and Cryptology*. Springer, 2011, pp. 302–318.

[109] P. Kleberger, A. Javaheri, T. Olovsson, and E. Jonsson, "A framework for assessing the security of the connected car infrastructure," in *ICSNC 2011: The Sixth International Conference on Systems and Networks Communications*, 2011, pp. 236–241.

[110] I. Broster and A. Burns, "An analysable bus-guardian for event-triggered communication," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*. IEEE, 2003, pp. 410–419.

[111] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi, "A method of preventing unauthorized data transmission in controller area network," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–5.

[112] N. Kumar and N. Chilamkurti, "Collaborative trust aware intelligent intrusion detection in vanets," *Computers & Electrical Engineering*, vol. 40, no. 6, pp. 1981–1996, 2014.

[113] M. Raya and J.-P. Hubaux, "Securing vehicular ad hoc networks," *Journal of computer security*, vol. 15, no. 1, pp. 39–68, 2007.

[114] ——, "Security aspects of inter-vehicle communications," in *5th Swiss Transport Research Conference (STRC)*, no. CONF, 2005.

[115] G. Samara, W. A. Al-Salihy, and R. Sures, "Security issues and challenges of vehicular ad hoc networks (vanet)," in *4th International Conference on New Trends in Information Science and Service Science*. IEEE, 2010, pp. 393–398.

[116] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent transportation systems*, vol. 16, no. 2, pp. 546–556, 2014.

[117] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, p. 2015, 2015.

[118] B. Cyr, J. Mahmod, and U. Guin, "Low-cost and secure firmware obfuscation method for protecting electronic systems from cloning," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3700–3711, 2019.

[119] G. Sabaliauskaite, L. S. Liew, and J. Cui, "Integrating autonomous vehicle safety and security analysis using stpa method and the six-step model," *International Journal on Advances in Security*, vol. 11, no. 1&2, pp. 160–169, 2018.

[120] G. Loukas, E. Karapistoli, E. Panaousis, P. Sarigiannidis, A. Bezemskij, and T. Vuong, "A taxonomy and survey of cyber-physical intrusion detection approaches for vehicles," *Ad Hoc Networks*, vol. 84, pp. 124–147, 2019.

[121] R. G. Engoulou, M. Bellaïche, S. Pierre, and A. Quintero, "Vanet security surveys," *Computer Communications*, vol. 44, pp. 1–13, 2014.

[122] H. Hasrouny, A. E. Samhat, C. Bassil, and A. Laouiti, "Vanet security challenges and solutions: A survey," *Vehicular Communications*, vol. 7, pp. 7–20, 2017.

[123] M. N. Mejri, J. Ben-Othman, and M. Hamdi, "Survey on vanet security challenges and possible cryptographic solutions," *Vehicular Communications*, vol. 1, no. 2, pp. 53–66, 2014.

[124] S. Karnouskos and F. Kerschbaum, "Privacy and integrity considerations in hyperconnected autonomous vehicles," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 160–170, 2017.

[125] The LLVM Compiler Infrastructure. [Online]. Available: http://www.llvm.org/

[126] C. A. Lattner, "LLVM: An infrastructure for multi-stage optimization," Master's thesis, University of Illinois, 2002.

[127] J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, "Conversion of control dependence to data dependence," in *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '83. New York, NY, USA: ACM, 1983, pp. 177–189. [Online]. Available: http://doi.acm.org/10.1145/567067.567085

[128] J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the 8th Annual Symposium on Computer Architecture*, ser. ISCA '81. Los Alamitos, CA, USA: IEEE Computer Society Press, 1981, pp. 135–148. [Online]. Available: http://dl.acm.org/citation.cfm?id=800052.801871

[129] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[130] S. A. Mahlke, R. E. Hank, J. E. McCormick, D. I. August, and W.-M. W. Hwu, "A comparison of full and partial predicated execution support for ILP processors," *SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 138–150, May 1995. [Online]. Available: http://doi.acm.org/10.1145/225830.225965

[131] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt, "Branch classification: a new mechanism for improving branch predictor performance," *International Journal of Parallel Programming*, vol. 24, no. 2, pp. 133–158, 1996.

[132] K. Kennedy and J. R. Allen, *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[133] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Department of Computer Science, The University of Auckland, New Zealand, Tech. Rep. 148, 1997.

[134] M. Mowbray, S. Pearson, and Y. Shen, "Enhancing privacy in cloud computing via policy-based obfuscation," *The Journal of Supercomputing*, vol. 61, no. 2, pp. 267–291, 2012.

[135] M. Hataba and A. El-Mahdy, "Cloud protection by obfuscation: Techniques and metrics," in *2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Nov 2012, pp. 369–372.

[136] G. You, G. Kim, S.-j. Cho, and H. Han, "A comparative study on optimization, obfuscation, and deobfuscation tools in android." *J. Internet Serv. Inf. Secur.*, vol. 11, no. 1, pp. 2–15, 2021.

[137] J. Park, H. Kim, Y. Jeong, S.-j. Cho, S. Han, and M. Park, "Effects of code obfuscation on android app similarity analysis." *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 6, no. 4, pp. 86–98, 2015.

[138] S. Dong, M. Li, W. Diao, X. Liu, J. Liu, Z. Li, F. Xu, K. Chen, X. Wang, and K. Zhang, "Understanding android obfuscation techniques: A large-scale investigation in the wild," in *International conference on security and privacy in communication systems*. Springer, 2018, pp. 172–192.

[139] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee, "Impeding malware analysis using conditional code obfuscation." in *The Network and Distributed System Security Symposium NDSS*, 2008.

[140] B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray, "A generic approach to automatic deobfuscation of executable code," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 674–691.

[141] G. Bonfante, J. Fernandez, J.-Y. Marion, B. Rouxel, F. Sabatier, and A. Thierry, "Codisasm: Medium scale concatic disassembly of self-modifying binaries with overlapping instructions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 745–756.

[142] K. Coogan, G. Lu, and S. Debray, "Deobfuscation of virtualization-obfuscated software: a semantics-based approach," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 275–284.

[143] B. Cheng, J. Ming, E. A. Leal, H. Zhang, J. Fu, G. Peng, and J.-Y. Marion, "Obfuscation-resilient executable payload extraction from packed malware," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[144] B. Alouffi, M. Hasnain, A. Alharbi, W. Alosaimi, H. Alyami, and M. Ayaz, "A systematic literature review on cloud computing security: Threats and mitigation strategies," *IEEE Access*, vol. 9, pp. 57 792–57 807, 2021.

[145] I. Kanwal, H. Shafi, S. Memon, and M. H. Shah, "Cloud computing security challenges: A review," *Cybersecurity, Privacy and Freedom Protection in the Connected World. Advanced Sciences and Technologies for Security Applications. Springer, Cham*, pp. 459–469, 2021.

[146] M. K. Sasubilli and R. Venkateswarlu, "Cloud computing security challenges, threats and vulnerabilities," in *2021 6th International Conference on Inventive Computation Technologies (ICICT)*. IEEE, 2021, pp. 476–480.

[147] D. Sampson and M. M. Chowdhury, "The growing security concerns of cloud computing," in *2021 IEEE International Conference on Electro Information Technology (EIT)*. IEEE, 2021, pp. 050–055.

[148] S. Bugiel, S. Nürnberger, A.-R. Sadeghi, and T. Schneider, "Twin clouds: Secure cloud computing with low latency," in *Communications and Multimedia Security*, B. De Decker, J. Lapon, V. Naessens, and A. Uhl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 32–44.

[149] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, ser. SFCS '86. Washington, DC, USA: IEEE Computer Society, 1986, pp. 162–167. [Online]. Available: https://doi.org/10.1109/SFCS.1986.25

[150] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, T. Rabin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 465–482.

[151] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Advances in Cryptology – EUROCRYPT 2011*, K. G. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–148.

[152] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Public Key Cryptography – PKC 2010*, P. Q. Nguyen and D. Pointcheval, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 420–443.

[153] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient TCB reduction and attestation," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 143–158.

[154] D. C. Latham, "Department of defense trusted computer system evaluation criteria," *Department of Defense , United States of America*, 1986.

[155] J. Criswell, A. Lenharth, D. Dhurjati, and V. Adve, "Secure virtual architecture: A safe execution environment for commodity operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 351–366, Oct. 2007. [Online]. Available: http://doi.acm.org/10.1145/1323293.1294295

[156] W. Futral and J. Greene, *Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters*. Springer Nature, 2013.

[157] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[158] Advanced Micro Devices, Inc. (AMD), "Strengthening vm isolation with integrity protection and more," *White Paper, Santa Clara, CA, USA*, 2020.

[159] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[160] O. Demigha and R. Larguet, "Hardware-based solutions for trusted cloud computing," *Computers & Security*, p. 102117, 2021.

[161] J. V. Cleemput, B. Coppens, and B. De Sutter, "Compiler mitigations for time attacks on modern x86 processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 1–20, 2012.

[162] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi, "Absynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures," in *Network and Distributed Systems Security (NDSS) Symposium*, 2020.

[163] C. Shen, C. Chen, and J. Zhang, "Micro-architectural cache side-channel attacks and countermeasures," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 441–448.

[164] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *IEEE Symposium on Security and Privacy (SP)*, 2021.

[165] D. McCann, K. Eder, and E. Oswald, "Characterising and comparing the energy consumption of side channel attack countermeasures and lightweight cryptography on embedded devices," in *Proceedings of the 2015 International Workshop on Secure Internet of Things*, ser. SIOT '15. USA: IEEE Computer Society, 2015, p. 65–71.

[166] N. Belleville, D. Couroussé, K. Heydemann, and H.-P. Charles, "Automated software protection for the masses against side-channel attacks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, pp. 1–27, 2018.

[167] M. Hataba, A. El-Mahdy, and E. Rohou, "OJIT: A novel obfuscation approach using standard just-in-time compiler transformations," in *The Proceedings of the 4th Dynamic Compilation Everywhere workshop held in conjucntion with 10th HiPEAC Conference*. ACM, 2015.

[168] J. Nagra and C. Collberg, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Pearson Education, 2009.

[169] C. Edwards, "Researchers probe security through obscurity," *Commun. ACM*, vol. 57, no. 8, pp. 11–13, Aug. 2014.

[170] QEMU: the FAST! processor emulator. [Online]. Available: https://www.qemu.org/

[171] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[172] J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, "Conversion of control dependence to data dependence." New York, NY, USA: Association for Computing Machinery, 1983.

[173] M. Hataba, A. Sherif, and R. Elkhouly, "A proposed software protection mechanism for autonomous vehicular cloud computing," in *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2021, pp. 878–881.

[174] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *International Conference on Security and Cryptography for Networks*. Springer, 2018, pp. 544–562.

[175] V. S. Miller, "The weil pairing, and its efficient calculation," *Journal of cryptology*, vol. 17, no. 4, pp. 235–261, 2004.

[176] H. Goumidi, S. Harous, Z. Aliouat, and A. M. Gueroui, "Lightweight secure authentication and key distribution scheme for vehicular cloud computing," *Symmetry*, vol. 13, no. 3, p. 484, 2021.

[177] Q. Huang, Y. Yang, and Y. Shi, "Smartveh: Secure and efficient message access control and authentication for vehicular cloud computing," *Sensors*, vol. 18, no. 2, p. 666, 2018.

[178] V. Kumar, M. Ahmad, A. Kumari, S. Kumari, and M. K. Khan, "Sebap: a secure and efficient biometric-assisted authentication protocol using ecc for vehicular cloud computing," *International Journal of Communication Systems*, vol. 34, no. 2, p. e4103, 2021.

[179] Q. Jiang, J. Ni, J. Ma, L. Yang, and X. Shen, "Integrated authentication and key agreement framework for vehicular cloud computing," *IEEE Network*, vol. 32, no. 3, pp. 28–35, 2018.

[180] S. Rana, D. Mishra, and S. Gupta, "Computationally efficient and secure session key agreement techniques for vehicular cloud computing," in *Advances in Communication and Computational Technology*. Springer, 2021, pp. 453–467.

[181] D. Bhattacharyya, R. Ranjan, F. A. A., and M. Choi, "Biometric authentication: A review," *International Journal of u and e Service, Science and Technology*, vol. 2, 2009.

[182] M. M. Alrifaee, "A short survey of iris images databases," *Available at SSRN 3616735*, 2020.

[183] C. Ream, "What you should know about biometrics in the cloud," *National Security Agency, Eyelock Whitepaper*, 09 2016.

[184] W. Zhang, S. Xiao, Y. Lin, T. Zhou, and S. Zhou, "Secure ranked multi-keyword search for multiple data owners in cloud computing," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 276–286.

[185] A. Kumar and A. Passi, "Comparison and combination of iris matchers for reliable personal identification," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–7.

[186] IITD Iris Database. [Online]. Available: https://www4.comp.polyu.edu.hk/~csajaykr/IITD/Database_Iris.htm

[187] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[188] A. I. Borisenko *et al.*, *Vector and tensor analysis with applications*. Courier Corporation, 1968.

[189] V. Rajasekar, J. Premalatha, and K. Sathya, "Enhanced biometric recognition for secure authentication using iris preprocessing and hyperelliptic curve cryptography," *Wireless Communications and Mobile Computing*, vol. 2020, 2020.

[190] Z. Rui and Z. Yan, "A survey on biometric authentication: Toward secure and privacy-preserving identification," *IEEE Access*, vol. 7, pp. 5994–6009, 2019.

[191] N. Nedjah, R. S. Wyant, L. M. Mourelle, and B. Gupta, "Efficient yet robust biometric iris matching on smart cards for data high security and privacy," *Future Generation Computer Systems*, vol. 76, pp. 18–32, 05 2017.

[192] S. Thavalengal, P. Bigioi, and P. Corcoran, "Iris authentication in handheld devices-considerations for constraint-free acquisition," *IEEE Transactions on Consumer Electronics*, vol. 61, no. 2, pp. 245–253, 2015.

[193] N. Ortiz, R. D. Hernande, R. Jimenez, M. Mauledeoux, and O. Aviles, "Survey of biometric pattern recognition via machine learning techniques," *Contemporary Engineering Sciences*, vol. 11, 2018.

[194] Q. Guo and J. Zheng, "An iris recognition algorithm for identity authentication," *2018 international Conference on Intelligent Transportation, Big Data and Smart City ICITBS*, 2018.

[195] Z. Shi and Guangdong, "Processing method and system for identity authentication with mobile terminal based on iris recognition," *Huizhou TCL Mobile Communication*, 10 2015.

[196] K. A. Bakshi, B. Prasad, and S. K., "An efficient iris code storing and searching technique for iris recognition using non-homogeneous k-d tree," in *2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, 2015, pp. 34–38.

[197] C. Rathgeb and A. Uhl, "Attacking iris recognition: An efficiient hillclimbing technique," *IEEE/IAPR International Conference on Patter Recognition (ICPR*, pp. 1217–1220, 2010.

[198] "Statistical attack against iris-biometric fuzzy commitment schemes," *IEEE Computer Soociety Conference on Computer Vision and Pattern Recognition Workshops*, pp. 22–30, 2011.

[199] N. Kohli, D. Yadav, M. Vatsa, R. Singh, and A. Noore, "Detecting medley of iris spoofing attacks using desist," *2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2016.