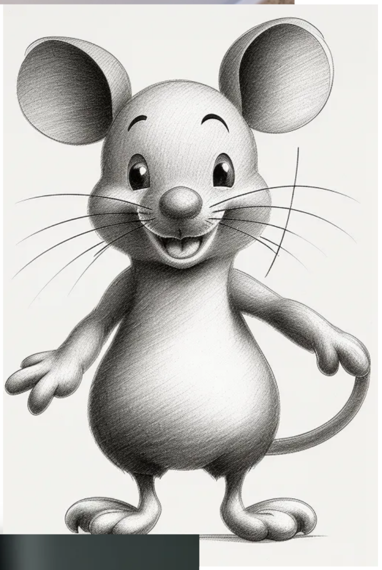
Automotive Software Attestation: Self, Remote, and Peer Building Trust in Autonomous Driving Safety Systems

by

Robert Michael Kaster

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer and Information Science)
in the University of Michigan-Dearborn
2024



Robert Michael Kaster

rkaster@umich.edu

ORCID iD: 0000-0003-0061-5286

Ammor magnus doctor est. I must thank my wife Judith for the exciting journey and for the motivation to return to UM after the kids were away at college.

ACKNOWLEDGEMENTS

Over the course of my academic and professional careers, I have had the privilege of interacting with very intelligent, gifted, and influential people. Among these, Professor Di Ma stands out for her patient and persistent counsel. It was an unexpected experience to have someone with her expertise invest so much time inculcating advanced knowledge while helping me cultivate research and academic skills. Meeting with Professor Ma has been the highlight of the week with her genuine personal interest and stimulating conversation.

I had the privilege of taking Birhanu Eshete's first class at UM Dearborn and have enjoyed interacting with him throughout my entire time here. He has consistently been both personally encouraging and intellectually challenging. I want to thank him for participating on my committee and for unlocking many doors in the world of cybersecurity.

James Freudenberg's presentation of control theory for my Master's degree was both captivating and challenging. He sparked a continued interest that led to my decision to pursue doctoral studies. His admonition to "respect the unstable" has helped me throughout my career developing automotive safety systems. With his recent work leading the automotive engineering program in Ann Arbor, I appreciate him taking the time to step out of his comfort zone and serve on my dissertation committee for automotive cybersecurity.

Probir Roy joined the university during quarantine and was my fellow companion in our first virtual department meeting mixers. He reviewed my early papers and provided valuable feedback and challenged me to better explain my ideas.

I want to thank Hafiz Malik for lending his expertise in automotive cybersecurity from the ECE department to serve on my committee. Professor Malik's reputation among the engineers that have studied under him is greatly appreciated and led to him sharing his expertise on automotive security and speaking at our Bosch cybersecurity day to our leaders and technical experts.

I need to explicitly recognize my debt to Linxi Zhang, who has graciously offered suggestions and help from my first project detecting poor configuration settings on mobile apps to man-in-the-middle attacks, supporting presentations, to suggestions on formatting this thesis. Linxi has been a consistent help, even after becoming an assistant professor. I'd also like to thank Ahmad Nasser who first informed me that UM Dearborn offered this program and who took my request to accelerate the secure boot process and turned it into a research problem. Thanks also to Andre Weimerskirch from MCity and Lear for helping to get me started in the Ph.D. program.

Penultimately, I would like to thank Oliver Korasiak, Paul Thomas, and Lauren Iwema and Bosch LLC as a whole for allowing, supporting, and funding my pursuit of a Ph.D. I would also like to thank Frederik Stumpf, Anthony Freilach, Ivan Granero, Francisco Briceno, Ashish Behl, Bartosz Bakalarczyk, Andreas Dolezal, Martin Emele, Martin Schmiedecker, Josh Rambo, Michael Tucci, Rita Barrios, Peter Morris, Sven Schran, Jaime Lopes, Srikanth Narayanan Venkatesa, Jens Schmuelling, Bob Tremain, Naveenkumar Balasubramanian, Carina Boettcher, Jorge Merchan Guajardo, Josh Beck, Zachariah Pelletier, Tim Arntson, Ramona Jung, Markus Hofmann, and Holger Niemann from Bosch for their support while I pursued doctoral research while maintaining my full-time work responsibilities. Dan Zajac and Michael Westra from Ford, Stephen McFarland from Toyota, Tobias Gärtner from Amazon, Tom Forest and David Nairn from GM, Stephen McKnight and Bill Mazzara from Stellantis, Faye Francy, Josh Poster, and Ricky Brooks from the Auto-ISAC, John McShane from Synopsys, Douglas Gordon from CNH, David Brumley from Carnegie Mellon, Willem Melching from Commaai, Ira MacDonald from High North, and John Auld from Cybellum, Philip Lapczynski from Renesas, John Cotner and Allen Houck from NXP for criticism and test samples that contributed to this work.

Ultimately, I need to recognize my wife Judith, who has pushed me to do rather than just think, for my children for whom I wanted to provide an example of seeking God with your mind and loving your neighbor by innovating to improve their lives in this magical and meaningful world, for my growing family that promises to do even more, and my parents who pushed me to start my journey at UM and family that's been with me for the entire journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

"Can you trust a car's software (SW)?" This question attracts attention from professionals, politicians, and professors for three reasons: ability, access, and interest. Autonomous and driver assistance systems have responsibility and authority in controlling the vehicle - increased ability. Many cars today are connected to the outside world with Bluetooth, Wi-Fi, USB, and cellular connections - increased access. Talented researchers, organized crime and nation state intelligence agencies are turning their attention to develop sophisticated cyber-attacks on expensive critical infrastructure - increased interest. People all over the world trust their vehicles with their lives and livelihoods. The increase of authority within, connectivity without, and motivation to exploit all serve to increase the risk of compromised vehicle software, the importance of checking, and the required depth of verification. My research focuses on three critical challenges for automotive SW attestation, each expanding on the question for a different 'you'.

1. **Self Attestation** How can a real-time *automotive controller verify its own SW* before starting and maintain compatibility with start-up time, functional safety, and limited power requirements? Secure boot has successfully protected systems from executing untrusted SW, but low-power controllers lack sufficient time to check every memory cell while satisfying real-time functional safety requirements. Automotive controllers need to maintain security through multiple cycles of remote, unsupervised operation and reach a secure state in a safe manner when an anomaly is detected. To accelerate the boot time, we propose Sliced Secure Boot: build fingerprints by slicing orthogonally through memory blocks, protect each cell with a reusable fingerprint using a reproducible pattern with sufficient entropy, and randomly check one fingerprint pattern during boot. Sampling does not offer equivalent protection to exhaustive checks, but careful sampling can provide a sufficient level of detection while maintaining compatibility with both startup time and functional safety requirements.

2. **Remote Attestation (RA)** How can *someone verify the SW integrity of an automobile* without access to sensitive intellectual property? Researchers have explored using RA on computers and Internet of Things (IoT) devices, but these use cases presuppose that the verifier has access to the full SW image or can control the memory access of the prover. Our Digital Shadow (DS) approach takes a 'snapshot' of a Digital Twin (DT) with sufficient detail to identify, but not enough to reconstruct. It builds a modified Merkle tree signature based on each module's current SW and compares this with a fingerprint that the Original Equipment Manufacturer (OEM) maintains of the DT. We measured the time required to calculate a vehicle fingerprint with 14 different types of vehicle controllers and the amount of data to support the concept. Verifying the SW integrity confirms that the correct SW is *available* but does not prove that it is actually *controlling* the vehicle. We expand DS to monitor internal and external plausibility verification of the run time and fault codes. This enables a quick verification of the vehicle's cyber-health state with limited trust requirements on the party performing the check.

3. **Peer Attestation** With advanced vehicle architectures, software defined vehicles, cloud computing, and service-oriented architectures, *how can a module build trust in the information it is receiving* in a manner that is dynamic, nuanced, and adaptive? Today's vehicle architectures build trust on a framework that is static, binary, and rigid. The Zero Trust (ZT) paradigm supports this dynamic need, but current implementations are focused on protecting information and do not consider the additional challenges that automobiles face interacting with the physical world. We propose expanding ZT for a Cyber-Physical System (CPS) by weighing the potential safety impact of taking action based on information provided against the amount of trust in the message and develop a method to evaluate the effectiveness of this strategy. This strategy offers a solution to the problems of implementing real-time responses to active attacks over vehicle lifetime.

Acronyms

ABS Anti-lock Braking System. 87

AES Advanced Encryption Standard. 12, 28, 42, 51–53

ASIL Automotive Security Integrity Level. 31, 93, 94, 102

CAN Controller Area Network. 16, 19, 21, 25, 84, 90, 101

CBC Cipher Block Chain. 42

CBC-MAC Cipher Block Chaining Message Authentication Code. 22, 28, 42

CMAC Cipher-based Message Authentication Code. 42, 83

CPS Cyber-Physical System. xiv, 7, 8, 16, 18, 22, 55, 93, 111

CVE Common Vulnerability Enumeration. 5

CVN Controller Verification Number. 16

DS Digital Shadow. xiv, 9, 59–61, 63, 64, 66, 67, 70, 72, 73, 77, 79, 80, 84, 85

DSA Digital Signature Algorithm. 14, 41, 121

DT Digital Twin. xiv, 9, 17, 60, 66, 71, 77, 79, 80, 85, 112

ECU Electronic Control Unit. 5–10, 14–17, 22–28, 31–34, 40, 41, 43, 55–60, 62, 63, 66, 67, 69–71, 75–80, 82–85, 87, 88, 90–95, 101, 102, 105, 111, 112, 117, 121, 122

FMEA Failure Mode and Effect Analysis. 93, 94, 102

FOTA Flash Over The Air. 6, 64, 109

FPGAs Field Programmable Gate Arrays. 13

HMAC Hash-based Message Authentication Code. 42

HSM Hardware Security Module. 12, 22–36, 42, 46, 49–53, 55, 84

HTA Hardware Trust Anchor. 121

HW hardware. 12, 19, 42, 51, 78–80, 85

IDS Intrusion Detection System. 10, 11, 18, 19, 88–90, 94–96, 98–102, 104, 106, 109, 112

IoT Internet of Things. xiv, 11, 14–16, 59

IP Intellectual Property. 60

IT Information Technology. 7, 11, 15, 16, 18, 88, 90, 91

IV Initial Value. 42

LUT LookUp Table. 24, 39, 46, 47, 52, 56

MAC Message Authentication Code. 9, 22, 27–29, 32, 42, 52, 59, 71, 73, 75, 76, 83, 84, 112

ML Machine Learning. 90, 96

MTD Moving Target Defense. 104, 110

NIST National Institute for Standards and Technology. 17, 88

OEM Original Equipment Manufacturer. xiv, 6, 26, 31, 34, 58–63, 66, 70–72, 77–79, 82, 85, 104

PBASS Probabilistic Boot Authentication Sampling Scheme. 14, 37

PDP Proof of Data Possession. 16, 61

PII Personally Identifiable Information. 59

PRNG PseudoRandom Number Generator. 34, 52

RA Remote Attestation. xiv, 9, 15, 16, 58, 61

RAM Random Access Memory. 53

RTD Run-Time Detection. 55

SDN Software Defined Network. 19, 20

SDV Software Defined Vehicle. 10, 11, 20, 34, 87, 88, 92, 104, 107, 109, 110, 112

SOC Security Operations Center. 18

SSB Sliced Secure Boot. 9, 21–24, 26, 27, 29, 31, 37, 38, 40, 43, 52, 53, 56, 57, 111, 122, 123

SW software. xiii, xiv, 5–12, 14, 15, 18, 22, 24, 26–28, 41, 42, 53–56, 58, 59, 61–63, 66, 70, 77–79, 83, 85–89, 103, 104, 109, 111, 112, 121

TRF Time Reduction Factor. 51–53

uC microcontroller. 7, 9, 71, 83

VIN Vehicle Identification Number. 66, 82

VSOC Vehicle Security Operation Center. 18, 88, 102–106, 109, 112

ZT Zero Trust. xiv, 11, 17, 19, 88, 90, 91, 93, 109, 110, 112

ZT4SDV Zero Trust for SW Defined Vehicles. 19, 89, 95, 104, 106

ZTA Zero Trust Architecture. 17

CHAPTER 1

Introduction

1.1 Motivation

SW attestation has become a critical factor in the modern automobile for three reasons: ability, access, and interest. The Electronic Control Unit (ECU) has dramatically increased its responsibility and authority in controlling modern vehicles - ability. In 2021, 42,915 people died in the US alone from traffic-related fatalities [5]; advanced technology offers the best hope to reduce this number [2] [3] [107]. Vehicles are rapidly adding functionality to improve economy and comfort, while enhancing the user's experience as well. Undergirding a vehicle is a network of microcontrollers, monitoring and interacting with the physical world in real time. As of January 1, 2024, the Common Vulnerability Enumeration (CVE) database lists 766 Bluetooth vulnerabilities, 104 in just 2023 [131]. With an estimated 87% of new vehicles now being outfitted with Bluetooth [167], connectivity to common remote interfaces is a factor for nearly all new vehicles - access. Governments [55] [67] and researchers [183] have issued warnings that not just friendly researchers [44] [190] but nation state actors and organized crime have begun turning their interest to cyber-physical systems - interest. How can we trust that all of this new powerful, connected SW is actually doing what it is supposed to do?

1.2 Questions to Answer

Trust is essential for automotive ECUs to support enhanced comfort and life saving features in modern vehicles. Systems, especially safety-relevant systems, need to be able to determine if the SW should be trusted before they begin controlling moving vehicles. A robust answer to this question requires several layers of protection. **The first layer** is within the vehicle, within each ECU itself. The ECU checking the integrity of its own SW is a process known as **self-attestation**. The most effective way to do this is secure boot. **The second layer** is demonstrating SW integrity to interested parties outside of the vehicle, this is known as **remote attestation**. **The final layer** is building trust to other devices that are using the information to control the vehicle. We call this process **peer attestation** - each ECU needs to prove its trustworthiness to the other ECUs that are using the information it provides. These other ECUs have other responsibilities and may not be capable of generating the information on their own.

1.3 Challenges

It is helpful to understand the unique challenges modern vehicles face in building trust to support enhanced comfort and life saving features as both connectivity to the outside world and authority in influencing vehicle operation increase. While ECUs from the 90's used non-reprogrammable memory, modern ECUs primarily use reprogrammable flash and OEMs are pushing to implement Flash Over The Air (FOTA). We can expect tomorrow's vehicles to see SW updates and modifications. As automotive engineers draw proven security techniques from computers, internet applications, and smartphones, unexplored elements of real-time constraints, limited memory space, physical safety concerns, unreliable network connections, and long life introduce factors that were not considered when these techniques were developed.

Lethargic response times in a laptop or smartphone may be annoying and eventually lead to the device being replaced, but every ECU in the vehicle is constantly racing to complete its calculations before they are needed to control the vehicle. ECUs perform calculations to determine the state of the vehicle, relying on timely updates from sensors and other ECUs. If these updates do not come before the calculations are performed, the systems cannot correctly determine which actions are appropriate to take. Instead of a video stream glitching on a phone, the motor controller cannot determine when to inject the fuel, the braking controller cannot apply the brakes to keep the car from skidding, and the steering controller cannot turn the wheels to keep the vehicle on the road. The real-time operation nature places rigid requirements on how long calculations can take.

Limited memory and computation resources play an important role in ECU design. A typical automotive microcontroller (uC) relies solely on flash memory built into the chip and single digit megabyte memory space is common. With a clock speed measured in MHz, the system is not capable of performing demanding cryptographic calculations or storing large numbers of keys. Advanced vehicle architectures may use high-powered zone, domain, and vehicle computers with requirements approaching supercomputer performance. These systems have much more powerful processors, but also significantly more memory to check.

The safety of the vehicle occupants is a primary goal in the design of nearly all automotive systems. Traditional Information Technology (IT) systems are concerned with protecting confidentiality, integrity, and availability of the information they store and process [158] [168]. The fact that the vehicle interacts with the physical world takes priority as a cybersecurity goal [43] [113] [114] [118] [119] [132] [151] [152] [165]. Any changes to the SW need to first be evaluated and verified, sometimes with in-vehicle testing to ensure the safety of the vehicle before a change can be rolled out to the fleet.

Vehicles are transportation devices, their raison d'être is to move people and products from one location to another. Mobility means that the vehicle must be able to function without a low-cost, high-availability network connection. This provides an additional challenge as the individual components or vehicles may not be able to report attacks or receive a new SW package.

Automotive controllers that interact with CPSs in real-time operate in a much more restricted ecosystem than laptops or smart phones. The user has very little direct interaction with the system (no keyboard or display) and has no means to load or configure new SW. Detecting modified embedded SW is in a sense easier – a stable memory space makes malware detection a question of looking for differences from the original SW. For many ECUs, it is sufficient to say that the SW has been modified, not necessarily identify which programs, apps, or memory cells have been changed. Any untrusted changes to the memory means that the ECU may not be safe to engage in critical operations [139].

Automotive security measures need to protect the vehicle over its entire lifetime. The *median* US vehicle age exceeds 12 years [58], roughly five times that of a smartphone's [52] [173].

1.4 Contributions

Systems, especially safety-relevant systems, need to be able to determine if the SW should be trusted before they begin controlling moving vehicles. Manufacturers, owners, users, and regulators need to be able to determine if the vehicles should be trusted. Individual components within the vehicle need to know if they should trust the information they are receiving to take safety-critical actions. This paper contributes to the tripartite question of trust necessary to secure the next generation of automobiles.

1.4.1 Self Attestation

Secure boot broadly refers to a process with three objectives: verify SW authenticity, achieve a desired level of assurance, and test SW before it is executed. Automotive controllers face fresh challenges beyond a typical computer or smart phone - they must maintain compatibility with the functional safety concept for CPSs, complete regular tasks within the confines of real-time requirements, squeeze all the code and data into limited on-chip memory, and thrive in unprotected environments. If our available time to check the SW only allows us to check a portion of the memory, how can we achieve a high level of assurance that the SW has not been compromised?

A pragmatic adversary wants to maintain basic system operation but add features, disable protections, or modify behavior; these changes will present as a series of contiguous code segments (executable change) or clustered groupings (calibration modification). Sliced Secure Boot (SSB) takes the memory as a series of blocks and slices through them - creating a fingerprint of the slice. It uses an ECU-specific key to ensure that an adversary cannot create the signature after modifying the memory and an ECU-specific pattern to prevent scalable attacks. We developed a formula to characterize the adversary's escape rate and verified this with simulations. We evaluated this concept on several automotive uCs and verified that the reduced time was compatible with automotive safety concepts.

1.4.2 Remote Attestation

As the connectivity and autonomy increase in automotive controllers, the need for external parties to verify the integrity of the SW also increases. We propose using DTs to maintain a copy of the intended SW and DSs to verify vehicle SW, efficiently and robustly.

RA in the automotive world faces several interesting challenges - the prover neither has access to the known-good SW nor controls the individual vehicles. The intellectual property value, the risk from increased cyber attacks, and liability with potential misuse complicate sharing - even with parties that need to verify the authenticity and freshness. The adversary may have direct control over the target vehicle and external communication, and even copy a module's memory.

A DS is a hierarchical tree built from signatures of each individual ECU. Our initial scheme uses a hash-based algorithm to build the signatures, but it does not provide robustness against replay attacks. Our second scheme uses a Message Authentication Code (MAC)-based algorithm with a random starting point address this weakness and demonstrate that the vehicle *has access to* the correct SW. We then include the fault codes and run time to ensure that the desired SW is actually *in control of the vehicle*. We propose reasonable estimates for time and data required to support this concept and provide measurements representative of real vehicle modules.

1.4.3 Peer Attestation

The automotive industry sees great potential enabling new features by implementing Software Defined Vehicle (SDV) architectures [16] [32], accompanying this is the need for SW modules to verify that their partners can be trusted to perform critical operations. With traditional automotive architectures, each task is assigned to a specific ECU. Trusting that an input is authentic or that an output can be executed means verifying that the assigned ECU is functioning properly and sending and receiving the messages. If the task can be dynamically reassigned to different ECUs, then a new means of building trust needs to be developed.

With new vehicle architectures, it is anticipated that ECUs will dynamically assign services to different SW modules that are running in the vehicle or even in the cloud. Before a module can assign a function to another module, it may want the module to perform some sort of attestation, proving that it can be trusted - either to provide reliable data or to faithfully execute a command. With limited memory space available, it is not feasible for each module to maintain a copy of its peers' program code.

Our contribution takes this proposal and builds a proof of concept. We build a paradigm to allow a single component of trust to dynamically interact with two sensors. We work through an example and demonstrate how the dynamic trust paradigm can allow an Intrusion Detection System (IDS) to directly interact with the system in a manner that is tolerant of false positive events and method to evaluate its performance. We then expand the concept to multiple trust components with a more sophisticated usage profile that can also process probabilistic intelligence information and adapt system behavior to active threats without requiring new SW.

1.5 Publications

The **Self Attestation** work was presented at Escar USA 2021 on sampled secure boot [97], at Escar Europe 2021 on reusable fingerprints [100], and the completed concept published in the Society of Automotive Engineers Journal of Connected and Automated Vehicles [99].

The **Remote Attestation** work was presented at the ACM Cyber-Physical Systems and Internet of Things Week 2023 in San Antonio [98].

The **Peer Attestation** work was submitted to three conferences scheduled in the first half of 2024. The first paper focuses on modifying the concepts of ZT to support an automotive IDS and is accepted to be presented at the IEEE International Conference on Computing and Machine Intelligence. The second expands this concept to support probabilistic intelligence and flexible architectures. The third paper provides more details at developing a dynamic trust concept allowing advanced vehicle electrical architectures to support a software defined vehicle with flexible and frequent updates.

1.6 Organization

Chapter 2 provides an overview of relevant literature reflecting automotive security measures and needs, and overview of security measures implemented in IT and IoT systems. Chapter 3 introduces our accelerated secure boot proposal that is compatible with automotive safety needs and evaluates its performance and security. Chapter 4 examines our remote attestation scheme that allows a third party to verify a vehicle's SW integrity without access to the full SW within reasonable time and data restrictions. Chapter 5 introduces our dynamic trust model for SDVs and examines an application to a single and multiple trust component system. Chapter 6 summarizes the impact of these innovations and discusses future work.

CHAPTER 2

Literature Review

2.1 Automotive Security Appliance - HSM

Bosch proposed a reference design for an Hardware Security Module (HSM) suitable for use in automotive controllers in 2011 [35], and worked with several microcontroller suppliers to implement this in their controllers [53] [90] [91] [127] [147] [163] and is widely used in the industry today. Figure 2.1 summarizes this reference design with a physical core in the microcontroller dedicated to security operations, dedicated security hardware (HW) blocks including an Advanced Encryption Standard (AES) accelerator, a true and pseudo-random number generators, and protected memory for storing keys and code. The security SW runs on this security core and performs various security critical functions, such as secure boot. Automotive functions, such as steering and braking, are supported by functional SW, which runs on the main core using the main memory. The main core does not have access to the protected memory, but the HSM can access both the protected and the main memory. SAE J3101 describes a general approach to an automotive hardware trust anchor [49]. Arm has also developed a TrustZone with similar objectives [73] [94].

2.2 Self Attestation

Arbaugh et al. developed a scheme to build trust within computers by having a trusted root check the signature of each SW component before it is executed - building a chain of trust to ensure

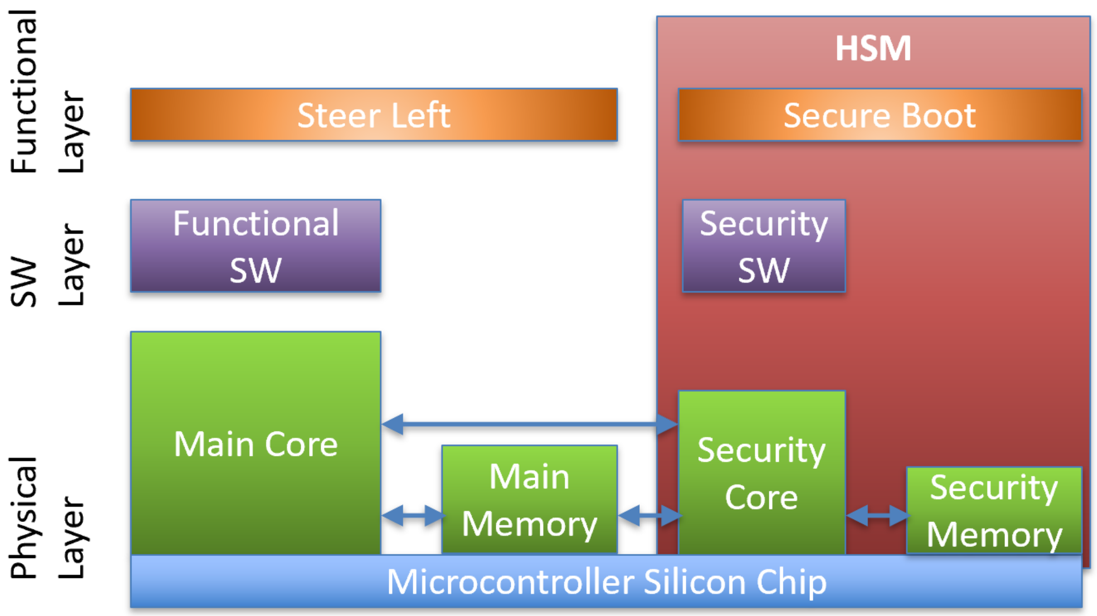


Figure 2.1: HSM Definition

proper operation [17] guaranteeing that the system is in a secure state when the boot process completes. No code executes unless it is implicitly trusted or explicitly verified. More recent work has emphasized the importance of secure boot [60] [115]. This strategy is well suited to systems that tolerate start-up delays and load multiple programs at different times and has been implemented for computers and smart phones for years but controllers without a file system that execute directly from flash memory do not lend themselves to this type of segmentation. Operating systems such as Windows [128] [129], iOS [12] [13] [14] [15], Ubuntu [185] [188], Arch Linux [18] [19], and Android [72] [177] have successfully integrated secure boot based on this concept. Secure boot has been discussed for embedded controllers [155] [181] and Field Programmable Gate Arrays (FPGAs) [111] [153]. Several automotive security firmware vendors have developed secure boot concepts based on Arbaugh's principles [71] [75] [159] [193], but are not able to check the entire memory space before the system begins operating.

Researchers have investigated methods of bypassing secure boot [8] [191].

Ateniese et al. proposed a two-component system for remote attestation: a data owner generates and maintains a set of single-use fingerprints, and the untrusted server maintains the data and calculates the fingerprints for the owner to verify that the data are still intact [25]. This concept provides a hopeful starting point for self-attestation but requires excessive memory space to support the entire life of the product. Recent research in remote attestation of embedded IoT devices provide interesting perspectives for automotive controllers [110].

Nasser et al. explored the concept of using a sampling-based strategy to accelerate the boot process with Probabilistic Boot Authentication Sampling Scheme (PBASS) [138]. They proposed creating a single-use fingerprint check for automotive systems when the system in a secure state to verify memory integrity on the next boot cycle. They demonstrated that this concept can provide a reasonable level of detection while meeting timing requirements but did not address the struggles with guaranteeing the resources to securely calculate the next fingerprint. If the compromise successfully evades detection, all future fingerprints built on the image would not detect the compromise.

A Digital Signature Algorithm (DSA) provides a powerful tool to verify authenticity when delivering new SW [189] [196] and has been used effectively with modern microprocessors. The use of asymmetric cryptographic algorithms provides scalability but requires a significant amount of computation. DSAs are commonly used to verify the integrity of new SW, an event that occurs relatively infrequently and only when the vehicle is in a dormant state. The microcontrollers used in ECUs such as braking or steering controllers do not have sufficient time to feasibly utilize these algorithms on every boot cycle.

2.3 Remote Attestation

Remote attestation challenges a device to prove, using time or behavior measurements, to demonstrate that it has not been modified and will behave and use resources as expected [28]. Remote Attestation strategies for embedded and IoT devices need to be compatible with low powered microprocessors, limited run time, and are required to operate without a dedicated internet connection.

Coker et al. constrain the system to to provide evidence about the state of the running SW to determine if an attack has occurred [48]. They highlight five principles: fresh information (what the system is doing now), comprehensive information (assess the entire system), constrained disclosure (information distribution defined for need), semantic explicitness (clear indication of what is being attested), and trustworthy mechanism (provide evidence of mechanism).

Nunes et al. proposed several strategies to remotely verify data integrity in semi-trusted remote servers including embedded and devices [142] [143] [144] [145] [146]. The data owner uses a trusted verifier to measure the current state of the remote memory in a not-yet trusted prover. Before transferring the data to the remote server, the data owner uses seed-based random patterns to sample memory locations throughout the memory space and stores the associated fingerprints in a trusted location. Then during use, the *verifier* requests the *prover* to check one of the seed's patterns and submit the current fingerprint for comparison. This strategy assumes that the party performing the verification had access to the entire dataset and can compel the prover to verify arbitrary memory cells. Xian integrated classical security mechanisms into RA [200].

Francillon et al. explored a minimalist approach to RA [79]. They emphasized verification of the internal state of the device is necessary for embedded devices, especially to ensure that the targets will behave as intended.

With new vehicle architectures, it is anticipated that ECUs will dynamically assign services to different SW modules that are running in the vehicle or even in the cloud. Before a module can assign a function to another module, it may want the module to perform some sort of attestation, proving that it can be trusted - either to provide reliable data or to faithfully execute a command. With limited memory space available, it is not feasible for each module to maintain a copy of its peers' program code. Dushku et al. developed a scheme to allow IoT devices to verify integrity with the use of logical clocks [65]. IoT remote attestation strategies addressed peer attestation in swarms of identical or similar modules [7] [10] [34] [38] [59] [66] [105] [164] [186].

Kuang et al. prepared a comprehensive survey of RA in IoT devices in 2022 [104]. Chantis et al. cataloged differences between IoT and IT security [42].

Ateniese et al. developed a model called Proof of Data Possession (PDP) to allow a trusted party to verify that a remote data center has maintained a copy of the data stored there, without access to the entire dataset [23] [24] [25]. The scheme precomputes fingerprints of samples throughout the memory and provides them to a trusted checker. The checker requests the data store to recreate the fingerprint and prove that the data are still available. Ma and Tsudik proposed a lean secure logging scheme [117]. Eldefrawy et al. considered secure cloud data storage [68] and formal verification [70] and RA [69]. Others have continued work on PDP [74] [95].

Ding and Tsudik proposed the concept of presence attestation to verify the state of health of the security appliance [61]. Carpent et al. expanded to consider RA via self-measurement [39]. Tan et al. considered IoT devices [179], Arias et al. as well [20] [21].

Many vehicles support a checksum-based verification of the engine controller memory. Checksums are not collision resistant and are only reliable when used in trusted environments. The Controller Verification Number (CVN) offers little protection against a motivated adversary. This strategy of checking each ECU individually does not scale well to support verifying an entire fleet.

2.4 Cyber-Physical Systems

IT systems focus on the CIA triad - protecting confidentiality, integrity, and availability. When we consider connecting the digital world to the physical world, security considerations need to expand. Additionally, physical systems tend to have limited resources and additional requirements. Research has considered the impact on CPS [1] [11] [41] [40] [50] [54] [56] [87] [187] [197] and on IoT systems [116] [178].

They also use special communication protocols such as Controller Area Network (CAN) or automotive Ethernet, which have unique approaches to cyber-attacks and defenses [46] [45] [78] [81] [83] [106] [172] [184] [194] [199] [204]. New systems are also connecting to infrastructure to improve safety and functionality but bring the need for considering new risks [84] [134] [156].

2.5 Digital Twins

Grieves proposed the concept of DTs to support manufacturing [82]. Vickers used this at NASA to provide a tighter control of pre-prototype design. Automobile suppliers are exploring DTs to support new feature development and predictive diagnostics. To support predictive diagnostics, a virtual copy of the ECU is kept in the cloud and the vehicle sends data to the DT to model the same wear that the real vehicle is experiencing. Combined with experience from the rest of the fleet, it becomes possible to predict when maintenance will be required [6] [29] [30] [31] [33] [51] [88] [92] [101] [123] [130] [141] [150] [160] [171] [180] [182] [201].

2.6 Zero Trust

Rose et al. describe NIST's proposal for Zero Trust Architecture (ZTA) to protect enterprise assets by preventing data breaches and limiting lateral movement [157]. ZT is a paradigm to meet security objectives by following these high-level assumptions:

- The network is always hostile.
- External and internal threats exist at all times.
- Local network messages are not implicitly trusted.
- Implicit trust zones need to be as small as possible.
- Never grant trust implicitly.

Google [93] [195] and Amazon [161] have taken significant steps to introduce ZT in their infrastructure and published their findings.

Gilman [80] describes how ZT separates the responsibility for transferring data and controlling access into a data plane and a control plane. For each request, the requestor first proves his identity and then the policy engine verifies that the requestor should have access to that data. The trust engine determines the amount of verification required, depending on the sensitivity of the requested data access.

2.7 External Security Monitoring

Several measures have proven helpful in the IT security world [102]. A company can engage a Security Operations Center (SOC) to actively collect information and intelligence about their computers and devices to assess the threats and risks. IT systems can actively monitor computer networks, isolate compromised systems, patch vulnerable SW, and implement recovery procedures. A Vehicle Security Operation Center (VSOC) monitors risk by evaluating the entire fleet, but current systems struggle to continuously monitor vehicles, especially vehicles that are in motion, especially vehicles that an adversary may have isolated from communication back to the manufacturer [4].

Making even a small change to the vehicle SW may require extensive development effort and validation for a CPS. So even if a SW vulnerability is identified, it may require driving multiple vehicles on test tracks to ensure that the modified SW is safe to use before taking direct action based on the intelligence.

2.8 Internal Security Monitoring

An IDS can monitor network traffic for suspicious or malicious behavior. When augmented with *Prevention* to isolate a compromised device it may be called an IDPS or IPS. Allowing an IDS to directly interact with moving vehicles presents a complex problem, since a security action cannot compromise a safety measure [166].

Marchetti et al. considered IDS for automotive applications [120] [121]. Levi et al. widen the approach to consider the entire vehicle [108]. Lin et al. proposed mixed integer linear programming [109]. Kang and Kang proposed using Deep Neural Networks [96]. Narayanan et al. proposed a hidden Markov model [135] and a rule-based detection scheme [136]. Cho proposed a clock-based IDS [47].

Zhang et al. categorize automotive IDS as rule-based or neural network-based [202] [203]. Rule-based solutions are fast for real-time operation on a small microcontroller, but creative adversaries may uncover a gap in the rules. Neural networks go beyond explicit rules, but require significant computational power. False positive detections restrict granting IDS direct control of the vehicle. Even the full precision neural network exhibited a false positive rate of 0.08%. With more than a million CAN messages per hour, the number of false detections is so high that even a simple warning would frighten or annoy any driver.

2.9 Software Defined Networks

Software Defined Network (SDN) is a concept developed to address challenges with rigid and proprietary network HW by separating network *functions* from network *devices* [89] [133]. Abstracting physical HW management from each manufacturer's unique implementations allows policy managers to enable uniform configuration in a scalable manner. Decoupling policy actions from the HW that is implementing them enables rapid deployment of changes. By monitoring traffic patterns for deviations from normal usage patterns and suspicious activity, measures can be identified far more easily in a SDN than in a physically defined architecture.

Buttyan and Hubaux used game theory to evaluate the forwarder's dilemma - a selfish node should invest the resources to forward messages without an immediate benefit because, in the aggregate, it will benefit from other nodes forwarding its messages [37]. They quantified the cost and the benefit from taking an action, then looked for strategies to justify the cost. Developing a price and currency for ZT in automobiles provides the starting point for Zero Trust for SW Defined Vehicles (ZT4SDV). Do et al. have also investigated game theory to support cybersecurity investigations [62].

2.10 Software Defined Vehicle Architecture

The industry sees great potential in adopting a model based on SDNs, with a new architecture radically increasing the pace of development and updates. Slama et al. characterize a Software Defined Vehicle (SDV) with computers linking sensors, actuators, and displays to enable autonomous driving and an intelligent personalized immersive experience [169]. SDVs promise to aggressively improve development costs, time to market, scalabilty, and the pace of changes. This is enabled by the introduction of powerful general purpose vehicle computers that enable a significant changes in functionality and dependence. It also will redefine the value stream of the entire industry. The SDV looks to adopt the practices and pacing that have invigorated the smartphone industry [9] [16] [32].

CHAPTER 3

Sliced Secure Boot - Accelerated Self Attestation

3.1 Introduction

Automotive controllers need a new approach to self-attestation due to time, resources, and safety response. For many automotive controllers, an exhaustive secure boot is not feasible; there is simply insufficient time to complete the check before the system must be fully functional. To satisfy these, we can either spend more money on faster processors, delay verification and limit potential reactions, or check just a portion of the total memory. The only time when a security appliance knows that it is safe to control the safety core is before the safety core begins operation. Giving the security appliance unfettered authority to control the safety core exposes the system to risks from random defects and end-of-life failures. If the image check completes after the system is running, the system may not be able to safely enter a secure state until the next ignition cycle. Delaying the check may mean that the only *safe* action for the current ignition cycle is to deny access to CAN message authentication. Unfortunately, maintaining the freedom from interference requirement preventing unexpected interruptions of the safety core may preclude authoritative protections of the critical vehicle functions for which the secure boot was implemented.

Additionally, if a compromise is detected after the system is operating, intervening may interfere with the safety operation [56]. Checking the entire memory space may mean protecting the next ignition cycle rather than the current one. We developed and evaluated SSB - a sampling-based scheme with reusable fingerprints that accelerates the boot check and maintains compatibility with the safety concept. First, it processes the entire memory space as a set of d blocks, each comprised of b cells. With the ECU in a safe and secure state, the HSM generates a secret key to protect the fingerprints. In the secure state, it computes b fingerprints and stores them in unprotected memory. During each subsequent ignition cycle, the verify algorithm randomly selects one fingerprint and verifies that the protected cells have not been modified. SSB constrains fingerprint generation to ensure that each ECU has unbiased patterns and randomly selects which fingerprint to use on each boot cycle.

Our first approach generated a batch of single-use fingerprints and analyzed the escape rate over multiple boot cycles [97]. If the attackers can identify which cells are going to be checked on the next boot cycle or, more precisely, which cells are not likely to be checked on the next boot cycle, they can modify these cells to craft their attack and avoid detection. Single-use fingerprints provide a security advantage by only checking a given pattern once. But with limited memory space available, protecting an estimated 100,000 ignition cycles over the vehicle life with just 16 bytes for each fingerprint would take an unacceptably large 1.6 megabytes. Furthermore, an adversary who controls power to the ECU also controls fingerprint consumption and could exhaust the available fingerprints. We expanded this concept with SSB to support reusable fingerprints [99] [100].

Our survey indicates that a Cipher Block Chaining Message Authentication Code (CBC-MAC) provides the quickest method to securely authenticate SW for embedded controllers [35] [53] [90] [91] [127] [147] [163] [176]. A Message Authentication Code (MAC) uses a symmetric key to enable quick and robust detection of modified SW. Real-time CPSs require compatibility with rigid functional safety requirements [77], granting the security appliance authoritative control only before the boot process starts.

A pragmatic adversary wants to maintain basic system operation but add features, disable protections, or modify behavior; these changes will present as a series of contiguous code segments (executable change) or clustered groupings (calibration modification). Embedded controllers do not use a file system, this restricts how a successful modification can be made - but we may not know which locations the adversary will attack. We developed a formula relating the escape rate to the number and size of the modifications and verified it with simulation. To reuse fingerprints without leaking information, a controlled pattern generation ensures that each ECU does not undersample any memory cells. This does not provide the adversary with sufficient information to predict which cells are less likely to be checked on the next boot cycle, even with physical access to meticulously observe multiple boot cycles and record monitored cells before launching the attack. Modulo addition with a stream of pseudo-random numbers provides a per-ECU unique pattern.

In summary, SSB accomplishes a feasible secure boot concept for automotive controllers. SSB both meets reasonable security objectives and maintains compatibility with automotive functional safety and timing requirements. As a proof of concept, we adapted a commercial HSM firmware [75], tested SSB on five different microcontrollers and reduced the image check time by a factor of 9 on Infineon's and 23 on Renesas' automotive microcontrollers with 64 slices, allowing us to achieve our goal of verifying the system memory in a short enough time to avoid disrupting system availability. The concept efficiently used protected memory and total system memory. Each ECU is protected by a unique fingerprint key and a unique selection pattern to protect against scalable attacks. Sampled checks are not well suited to protect against small modifications, security critical memory such as the bootloader or diagnostic access control may need to be checked on each boot cycle.

3.1.1 Contributions

The noteworthy contributions of this chapter are:

- **Accelerated Secure Boot:** The SSB concept dramatically improves boot time allowing a reasonable level of security during the time allotted to the boot process with minimal memory space requirements.

- **Compatibility with Safety Concept:** SSB allows the entire boot check process to run inside the HSM and allows the security appliance to take authoritative action when it knows that the safety core is not performing any safety-critical operations, i.e., when it is in reset.

- **Empirical Data Supporting Boot Time:** We implemented the SSB algorithm on commercial HSM firmware and validated the performance impact.
- **Escape Rate Simulation:** We provide extensive simulation results documenting the coverage and escape rate for optimal attack patterns.

3.1.2 Organization

This chapter is organized according to the following structure. Section 3.2 describes our adversary model. Section 3.3 introduces our SSB algorithm. Section 3.4 analyzes the security and proposes improvements to the algorithm. Section 3.5 analyzes the performance of the scheme on several automotive controllers. Section 3.6 discusses the impact and Section 3.7 concludes the chapter.

3.2 Adversary Model

We assume the adversaries have detailed knowledge of the algorithm, physical access to a single device and the ability to control the power - starting and stopping the ECU repeatedly. Although many automotive microcontrollers use on-chip flash, some also use external flash memory which may leak fingerprint patterns to an adversary with physical access.

Our adversaries want to retain basic ECU functionality but modify the SW to achieve a specific goal - not just demonstrate the ability to change a bit, but to purposefully change segments of code or clusters of LookUp Table (LUT)s [103] [170]. They are able to read and write any unprotected memory cells but are not able to control the HSM or access its protected memory. The adversaries cannot access keys in the HSM protected memory but can read or modify any fingerprints or data in unprotected memory.

They are willing to invest significant time and energy to compromise a single device, with the goal of uncovering an extendable technique to compromise multiple devices without physical access. Reusable fingerprints increase sensitivity to adversaries capable of monitoring memory access patterns. With physical access to the device, they may be able use a logic analyzer or side-channel attack to monitor which memory addresses are being checked during repeated boot cycles.

Safety-relevant ECUs are designed to function reliably under worst-case conditions. With the available memory space in embedded controllers limiting the number of potential keys, and the attackers' ability to control the consumption of fingerprints, one-time pad strategies face a security challenge. Safe designs must be robust against intermittent power dropouts; security concepts cannot presume that, under worst-case conditions, the security appliance can control system power cycles or have sufficient time to refresh fingerprints. Since automotive controllers cannot control how often the ECU will boot (which drives how quickly the fingerprints will be used) the security risks of reusing fingerprints can be lower than the risk of regenerating fingerprints in potentially insecure states. The adversaries may be able to deny the HSM time to regenerate fresh fingerprints - either by submitting high-priority requests such as CAN message authentication requests or cutting power to the module before the full image check can be completed.

We assume that the adversary is able to modify the unprotected memory and control the operation of the host core (once it is running). Conversely, the HSM maintains its integrity and the attacker is not able to execute code on the HSM or view (or modify) the contents of the HSM-protected memory. These assumptions are relevant for our investigation because an attacker able to control the HSM and break the root of trust would also be able to compromise a standard secure boot scheme as well.

3.3 Proposed Approach

3.3.1 Initial Scheme Algorithms – Column-wise Slicing

In contrast to Arbaugh's strategy of checking each functional block as a unit, we slice orthogonal to functionality with the goal of detecting purposeful modifications. Arbaugh views each functional block as a unit and checks each byte of code sequentially - verifying that the entire function has not been modified. Embedded controllers do not use advanced file systems with the flexibility to store program segments around the hard drive, many of them execute exclusively from on-chip memory. Since the memory structure is more rigid, instead of examining entire functional blocks, we consider ‘physical’ blocks and slice them according to the memory address. We accomplish this by considering the memory as a matrix of d blocks (rows) each b cells wide (columns) and generating a fingerprint for each column-wise slice of memory. We have then a total of $b \times d$ memory cells. Reducing b increases the detection likelihood (increasing the ratio of modified cells per block) but also increases the number of blocks and the time to check each boot cycle.

Similar to other sampling-based verification schemes, SSB incorporates several stages.

- **Phase 1 - Setup Algorithm:** Assuming that a robust process such as Uptane [189] delivers and authenticates new SW [196], immediately after authentication, the HSM launches the setup algorithm to prepare the fingerprints.

- **Phase 2 - Preboot Verify Algorithm:** during each subsequent boot cycle, while the main core is still in reset, the HSM randomly selects one of the fingerprints and verifies that the protected cells have not changed since the fingerprint was generated. If the signature matches, the HSM allows the main core to start operating. The keys may be generated by the chip vendor, ECU manufacturer, or OEM, depending on how the OEM manages trust in the supply chain. The robustness of the algorithm depends on the keys being unique and unknown to the attackers. The OEM may verify key uniqueness or inject its own keys to reduce the risk of third parties controlling the keys.

- **Phase 3 - Run-Time Algorithm:** during normal operation, the HSM periodically executes a run-time modification detection check to determine if the SW has been modified.

- **Phase 4 - Response Strategy:** If an error is detected, the HSM should take action and bring the ECU or vehicle into a secure state, in a manner that does not interfere with the safety concept.

When we have new SW or just want to create a new batch of fingerprints, the system is brought into a secure state and then the HSM slices up the memory and creates a cryptographically unique fingerprint of each memory slice. These fingerprints are stored and do not change until new SW is loaded into the ECU. When the ECU initializes for all subsequent boot cycles, one of these fingerprints is selected at random, and the fingerprint is recalculated. If the freshly calculated fingerprint does not match the stored fingerprint; we know that the memory has been changed. Some systems may need to periodically check the fingerprints while the system is running to meet their security goals. If the HSM determines that the memory has been modified, the system should follow a predetermined plan to bring the ECU into a secure state without jeopardizing vehicle safety. The following discussion details these algorithms. Table 3.1 summarizes the variables used in this analysis.

Table 3.1: Variable Definition

Variable	Description
$D = \{D_1, \ldots, D_i, D_d\}$	memory arranged in equal sized data blocks, d total
$d = \|D\|$	number of data blocks
$D_i = \{D_{i,1}, \ldots, D_{i,j}, D_{i,b}\}$	each data block consists of b cells
$b = \|D_i\|$	number of cells in each data block
$F = \{F_1, \ldots, F_f\}$	the set of f fingerprints for the ECU
$f = \|F\|$	number of fingerprints
KEY	k bit Key used for MAC protection
$b \times d$	total number of cells in memory
i	variable to select which block to address
j	variable to select which cell from block i to address
m	number of ignition cycles after memory compromised before detection occurs
s	offset to increase entropy
v	number of compromised blocks
w	number of compromised cells per block
z	total number of compromised cells $z = (v \times w)$

3.3.2 Setup Algorithm

Algorithm 1 describes the SSB Setup algorithm which runs once when new SW is installed. A MAC function builds each fingerprint with an ECU-specific secret key and the memory cells from the slice of memory. The HSM stores the key in protected memory and the fingerprints in unprotected memory. Without access to the unique fingerprint key stored in protected memory, an attacker who is able to modify the SW and the fingerprints still cannot figure out how to make them match to escape detection.

For the initial algorithm $f = b$, one fingerprint protects each column. j is incremented to step through each of the b fingerprints. The MAC function has two inputs - the secret key and all the memory cells from the slice. We initialize the AES accelerator with the secret key and then build a slice of the column, picking the memory cells from each row that lie in that column. The cells from the slice are fed into the HSM to output a MAC, which is a fixed-length number describing the sequence that is very hard to determine without the key. Each fingerprint protects one cell from each block, and each cell is protected by one fingerprint. Since b and d are integers, their product may not perfectly cover the memory space. For the simulations, we calculated the remainder of $Total\ Memory - b \times (d - 1)$. The first columns in the final row then were calculated based on d cells. The other columns were calculated based on $d - 1$ cells.

Reducing b increases the detection likelihood (increasing the ratio of modified cells per block) but also increases the number of blocks and the time to check each boot cycle. A CBC-MAC function builds each fingerprint with an ECU-specific secret key and the memory cells from the slice of memory. The HSM stores the key in protected memory and the fingerprints in unprotected memory. Without the unique fingerprint key, the attacker will not be able to calculate the required MAC for the modified SW.

```
Algorithm 1: SSB_Setup()
Input: Consider data image D as a set of $d$ blocks of size $b$
$D = \{D_1, \ldots, D_i, D_d\}, |D| = d$
$D_i = \{D_{i,1}, \ldots, D_{i,j}, D_{i,b}\}, |D_i| = b$
Output: Fingerprints $\{F_j\}$ where $j \in \{1, \ldots, b\}$ of $b$ slices
begin
    Generate $k$ bit random ECU fingerprint $KEY$
    Store $KEY$ in protected memory
    for $j \in \{1, \ldots, b\}$ do
        $F_j \leftarrow MAC(KEY, D_{1,j} || D_{2,j} || \cdots || D_{d,j})$
        Store $F_j$ in unprotected memory
    end
end
```

The SSB Setup algorithm completes two nested loops. The initial algorithm assigns $f = b$, one fingerprint protecting each column. The outer loop prepares the b MAC-based fingerprints. The MAC function has two inputs - the secret key and all the memory cells from the slice. The inner loop builds the slice by selecting one cell from each block. The key and the cells from the slice are fed into the HSM to output a MAC, which is a fixed length number describing the sequence that is very hard to determine without the key. Each fingerprint protects one cell from each block and each cell is protected by one fingerprint.

3.3.3 Verify Phase

At each ignition cycle, Algorithm 2, the SSB Verify algorithm, randomly selects one column, computes the signature, and compares it with the stored signature. When the microcontroller starts up, the HSM does not allow the main core to begin operation until it completes the check. Since the HSM knows that the main core has not started, it can authoritatively control the HSM while maintaining compatibility with the safety concept. The boot process is accelerated by reducing the number of cells that we check. Once the Verify phase is complete, the HSM releases control of the system to the main core and completes the secure boot process.

Algorithm 2: SSB_Verify()

```
Input: D, F, KEY
Output: Pass/Fail
begin
    Randomly select fingerprint index $j \in \{1, \dots, b\}$
    $f' \leftarrow MAC(KEY, D_{1,j} || D_{2,j} || \cdots || D_{d,j})$
    if $f' = F_j$ then
        Return Pass
    else
        Return Fail
    end
end
```

3.3.4 Monitor Phase

It may be necessary to periodically check the health of the memory while the system is running. This concept can also be used to support an enhanced run-time manipulation detection scheme.

Real-time systems may not have sufficient cycle time to guarantee completion of both the image check and their core function, especially with compromised host core SW. Real-time systems require predictable time windows for the check to complete. The Run-Time Verification algorithm shown in Algorithm 3 periodically selects a random slice of the memory and verifies that the fingerprint still matches. If this is run continuously, it will verify the entire memory, running in the background as a series of short tasks. Checking a much smaller portion of memory allows the HSM to allocate a predictable time for the computation, perhaps with a higher priority. If the slices are small enough to not interfere with critical functions, the check could be run as an uninterruptible atomic process. Sequentially selecting the fingerprints could allow the attacker to craft an attack, as the just-checked cells will not be checked until the $b-1$ fingerprints have been checked. Randomly selecting the fingerprints would prevent this and incrementally builds trust that the memory has not been compromised.

Algorithm 3: SSB_Run-Time_Verify()

```
Output: Pass/Fail
begin
    while $Normal\ Operation = TRUE$ do
        SSB_Verify()
    end
end
```

3.3.5 Action Phase

Many ASIL systems approach failure response echoing Archilochus' maxim πολλ' οιδ' αλωπηξ, αλλ' εχινος εν μεγα - the fox knows many tricks, but the hedgehog just a good one. If you cannot be sure that you're safe, then roll up into a little ball and shut down. An ordered shutdown for random defects allows the system to prioritize completing a critical maneuver, then perform a graceful shutdown. If the HSM detects a failure after the system is running, current systems may not be able to safely, authoritatively, and immediately shut down the ECU. A vehicle-level approach can minimize interference with the safety core by the HSM denying access to the protected secret keys, e.g., by refusing to authenticate messages or allow reprogramming to occur.

Figure 3.1 describes a comprehensive system behavior for the SW verification process. First, the ECU starts up and hands control to the SSB Verify algorithm. If verification fails, we assume that the ECU is compromised and conduct a complete test on the system with the OEM's digital signature to ensure that the failure was not due to a random read error. The next step is to use the random defect memory correction algorithm if the error is not due to a malicious attack. If this is not possible, then the ECU should go into locked mode until the rehabilitation process can be used to reprogram the ECU. This rehabilitation process is outside the scope of this paper. The strategy of retesting failed boot checks to reduce nuisance faults triggering warranty claims should be tempered

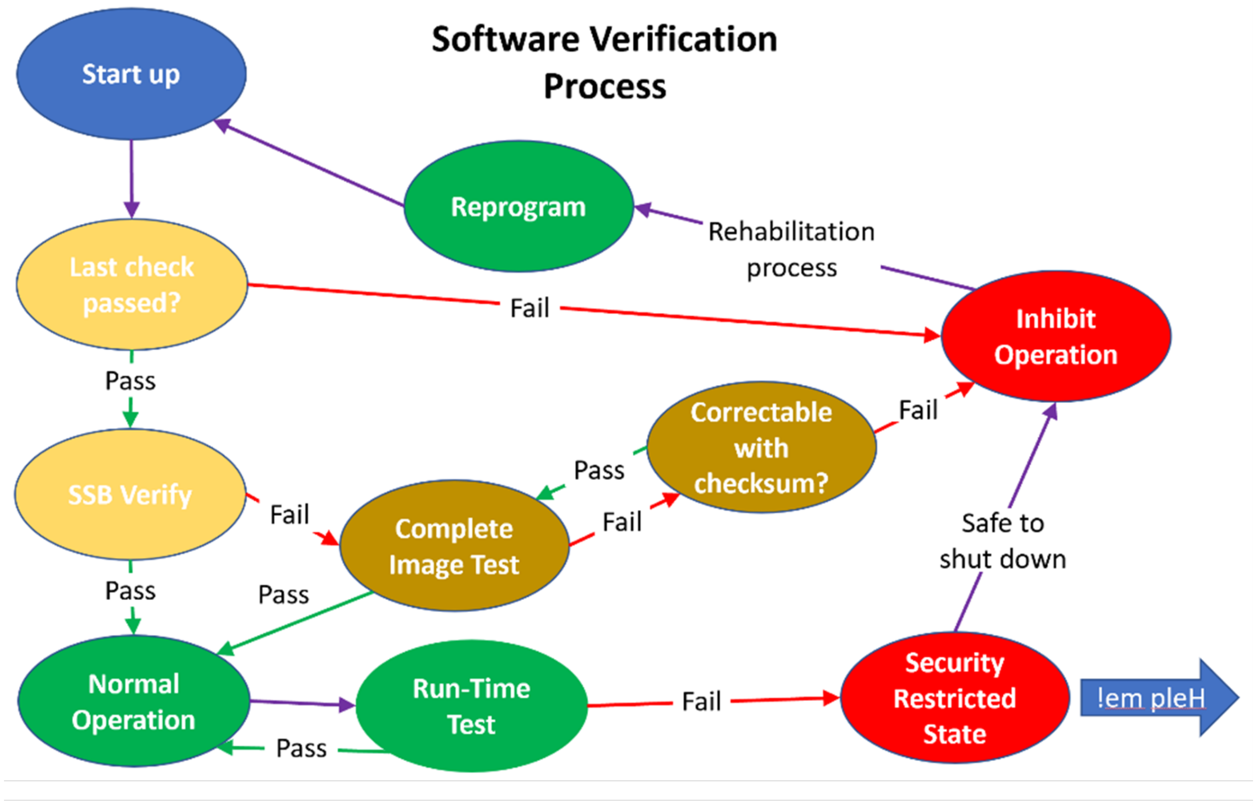


Figure 3.1: SW Verification Process

by the risk that rechecking a compromised system may offer an advantage to an attacker.

The process step background colors indicate the state of trust in the system. Yellow indicates that the system is in an untrusted state, green indicates that the system is now in a trusted state, and red indicates that the system is known to be compromised. The system starts in an untrusted state; before being allowed to enter a trusted state, it must pass an image test. This strategy only penalizes system startup behavior for suspicious devices, it does not penalize healthy systems.

While it may be desirable for the HSM to authoritatively assert control over a compromised host core from a security perspective, the system must also operate in a safe way when a defect occurs in the HSM. It is difficult to develop a concept that satisfies both concerns inside a single ECU – safe for random defects and secure for intentional attacks. Instead of just denying access to MAC keys, the HSM could sign messages with a “Compromise Detected” key. The HSM usually does not have access to the communication bus, and cannot directly notify the other ECUs in the vehicle or request assistance except through the host core, but using an invalid key to authenticate regular state of health messages would signal to the other ECUs that a compromise has been detected and request help. Figure 3.1 shows the HSM notifying the other ECUs in the vehicle that it needs help, the encryption is depicted with the request for help spelled out in reverse “!em pleH”. The host core must continue sending the periodic messages; otherwise, its partners would detect a problem with the ECU. Alerting the host core that the compromise was detected could potentially trigger malicious behavior since it may not have much longer to influence the vehicle. The host core does not know the secret key and would be unable to detect if the HSM were requesting help or notifying other ECUs that the integrity has been compromised.

Implementing a graceful limp home function *within a compromised* ECU presents significant challenges for all automotive ECUs and a complete treatment is outside the scope of this paper. As a tangible example, consider when the HSM inside the braking system controller fails a run-time check. The HSM recognizes that the memory has been corrupted, but it does not know what the vehicle is doing. The ASIL-D rated safety controller cannot accept a shutdown request from the HSM in the middle of a safety-critical maneuver because it must also maintain safe operation when a transistor inside the HSM fails. It is problematic to grant the HSM the authority to forcefully shut down the safety controller if it detects a problem and futile to merely request a shutdown from a compromised host controller.

Implementing a graceful limp home function *on a vehicle level* offers the opportunity to reach a secure state while following a safe path. A simplistic example illustrating the concept with the previous example could include: the video camera and radar systems could determine that the road ahead and behind are clear, and it is safe to reduce the vehicle speed. The engine controller could then safely reduce the maximum vehicle speed. The steering controller could determine the vehicle is not currently performing a high dynamics maneuver. And the intelligent power distribution controller could shut down power to the braking controller. While this is difficult for a single compromised controller to achieve, a distributed approach could maintain compatibility with the functional safety concept at each step along the way. Each critical vehicle system would need to have an appropriate response path mapped out. Chapter 5 proposes how SDV architectures may take further advantage of this approach.

3.3.6 Complicate Shadowing in Fingerprint Generation

Adding entropy to the fingerprint generation process makes it difficult for the adversary to predict fingerprint patterns. Embedded controllers need to work within several constraints.

- **Reproducible:** The random pattern used to create the fingerprints should not exceed the computational capabilities of the HSM in the verify phase to be a successful boot.
- **Unpredictable:** The random pattern should provide a unique pattern for each block, or at least a pattern with sufficient entropy to prevent pattern prediction, with special attention to preventing scalable attacks.
- **Indexable:** The pattern must sparingly use protected memory space.

On a conceptual level, we want to reshuffle the cells so that the adversary cannot predict which cells will be protected by which fingerprint. We would like to have a high entropy, but reproducible, mapping for how the fingerprints in this block are ordered. With the column-wise algorithm, essentially the offset is always zero. To provide a reproducible stream of random offsets, each ECU needs a unique random seed to generate the stream. First, use the true random number generator inside the HSM to generate the seed, or an OEM who provides the key could also generate the seed in the same process. Second, feed an ECU-unique seed into a PseudoRandom Number Generator (PRNG) to generate a reproducible stream of random bits and use $\lceil log_2 b \rceil$ bits to form a random offset for each block. Modulo addition shifts the fingerprints for that block by s_j. Lastly, use the PRNG to generate the random offsets for each block, so we only need to store the seed in protected memory.

At startup, the seed initializes the PRNG, which then provides a stream of bits that is unique for the ECU, difficult for an adversary to guess, but easy for the HSM to reproduce at each boot cycle, and we only need to store the seed in protected memory, not the entire bit stream. Let us examine a few concepts that allow a low-powered processor to quickly calculate the patterns, but still maintain the one-to-one mapping from fingerprints to memory cells in each block.

- **Modulo Addition:** Modulo addition, especially with a power of two base, is easily computed by the HSM. This provides an unpredictable starting point for each block, quickly shifting each fingerprint in the block by the same offset. This provides b possible patterns for each block.

$$s_j \leftarrow \{0, 1, \ldots, b-1\} \tag{3.1}$$

$$i' \leftarrow i + s_j \mod b \tag{3.2}$$

- **Modulo Subtraction:** We use an additional bit from the random bit stream to control the direction that the fingerprints are built. This provides $2b$ possible patterns, but still only b effective unique fingerprint mappings as the analysis will examine in detail. Multiplying i by -1 builds the fingerprints in the inverted order.

$$s_j \leftarrow \{0, 1, \ldots, b-1\} \tag{3.3}$$

$$sign \leftarrow \{-1, 1\} \tag{3.4}$$

$$i' \leftarrow i + sign \times s_j \mod b \tag{3.5}$$

- **Modulo Multiplication:** Using modulo multiplication with relative primes gives a ring of randomly scattered patterns. For each block i, the HSM selects a random offset s_i and then multiplies the column index j to determine the cell for each block. Scattering the fingerprints rather than building them sequentially makes shadowing of contiguous code blocks very difficult as each cell must be individually matched. Modulo multiplication in arbitrary bases is not supported in current HSMs, even using a computationally efficient base would still tax the HSM and introduce an unacceptable delay.

$$s_i \leftarrow \{1, 2,\ \ldots,\ b-1\} \tag{3.6}$$

$$i' \leftarrow s_i\ \times\ j \mod\ b \in\ \mathbb{P} \tag{3.7}$$

- **Random:** The best theoretically possible option is to generate a unique pattern for each block. We would need to maintain d patterns, one for each block. To keep track of the order for each of the b cells in the block, we would need to store $d \times b$ numbers, which is the total number of cells.

$$s_{i,j} \leftarrow \{1, 2, \ldots, b\},\ s_{i,j} \neq s_{i,j'}\ \forall\ j,\ j'\ \in \{1,\ \ldots,\ b\},\ i\ \in \{1, \ldots, d\} \tag{3.8}$$

$$i' \leftarrow f(s_{i,j}) \tag{3.9}$$

- **Reduced Random:** We consider a smaller, pragmatic number of patterns, sufficient to ensure entropy and generate these patterns. These should be unique to the ECU and stored in protected memory.

3.3.7 Complicate Shadowing in Fingerprint Selection

Shadowing resistance can also be enhanced by further constraining the fingerprint selection.

- **Oversampling:** Oversampled fingerprints with different groupings of cells across blocks limit the attacker's ability to successfully shadow across multiple fingerprints. Single-use fingerprints require an unbiased *source* but reusable fingerprints require an unbiased *result*. For a single-use fingerprint, the probability that a cell will be checked should not depend on its previous history. For reusable fingerprints, we must ensure that there are not cells which are under-protected considering previous history.

  A cell is considered under-protected if the $Pr\,(Check\ cell\ on\ next\ boot) < \frac{1}{b}$. Assuming the adversary can monitor repeated boot cycles to determine which cells are under-protected or even unprotected by any fingerprints, these cells are more likely to be targets for potential attack. Using multiple different fingerprints to protect each cell could improve the shadowing resistance, as the adversary would have to defeat different patterns, but the sampling should still ensure that each cell is protected by the same number of fingerprints.

- **Skipping:** Both PBASS and SSB consider the number of compromised cells in each block and have not taken advantage of the fact that the adversary's executable code consists of contiguous cells. If we know that the compromised segments have a minimum length of w, checking every w^{th} cell reaches 100% detection by the $\lceil \frac{b}{w} \rceil^{th}$ check. Although this puts a hard limit on how many checks are required to guarantee detection, the attacker could reduce w to defeat this approach.

- **History:** A sequential selection of the fingerprints speeds detection but an adversary could just wait until a specific cell is checked, then be confident that it will not be checked again for the next b boot cycles. A hybrid approach favoring recently unchecked fingerprints, could balance the increase in speed with the predictability, e.g., select one of the last $\frac{b}{2}$ cells checked with 25% weighting and 75% for the half that has not recently been checked.

Simulations with random modification placement did not show a strong advantage with randomly placed modifications but consider oversampling and history as potential approaches against an intelligent adversary.

3.3.8 Asymmetric Memory Focus

All cells are not equal from a security perspective. The ability to modify a single bit in the life cycle counter may have a significant security impact. Conversely, the ability to modify large sections of unexecutable memory may not offer any advantage to an attacker. Sampled memory checks need to carefully weigh the security impact of the memory cells and may require some cells to be included in multiple or even every fingerprint. Unused memory may not even require a cryptographically secure fingerprint. With every bit in its erased state, calculating a collision-resistant secret does not provide any additional confidence for self-attestation.

There are three options for handling security sensitive cells. The first, is to check the security sensitive cells every boot cycle and then start the sampled boot. The second, is to include the security sensitive cells in every fingerprint. We could also consider protecting cells with multiple fingerprints, but not every fingerprint – effectively reducing b for a sensitive region of memory.

3.4 Security Analysis

If we only sample a portion of the memory, how likely are we to detect the compromised code, especially as we continue reusing the fingerprints over multiple ignition cycles? Since SSB covers every memory cell with exactly one fingerprint, this maps to the ball and urn problem from combinatorics. We select b urns and assign each one of them to represent a fingerprint. We examine each of the d blocks from memory, placing a ball into the corresponding urn for each compromised cell. After examining the entire memory space, some of the urns will have balls in them and some may be empty. If we randomly select an urn and it contains at least one ball, checking the corresponding fingerprint would detect the compromised memory. If the selected urn happens to be empty, then it would escape detection. For our analysis, we randomly select an urn and check if it contains a ball. If it does, we detected the compromised memory. If not, we continue selecting urns until we find one with a ball. The number of urns we have to check before finding one with a ball corresponds to the number of boot cycles that would escape detection. This equivalent problem was used to significantly accelerate the simulation process to generate billions of simulations to provide smooth curves on a logarithmic scale.

3.4.1 Automotive Controller Memory Patterns and Shadowing

Automotive microcontroller memory typically diverges into two usage patterns: executable code, and calibration tables. LUTs provide a flexible and computationally efficient tool, where a small microcontroller manipulates a large number of variables with complex formulae, perhaps even without closed-form solutions, allowing engineers to tune behavior with varying granularity in different operating conditions. An adversary can attempt to modify component behavior by modifying the program by stringing together contiguous segments of code [122] or by changing the values of LUTs. In both cases, modified cells will tend to cluster together, resulting in a high number of compromised cells in a small region.

If we do not have sufficient time to check every memory cell before booting, we can take advantage of the restrictions that the microcontroller architecture places on the adversary. Embedded controllers typically do not have dynamic file systems and adversaries need to work within a more rigid memory structure. We can improve the detection rate over just randomly selecting cells from the entire memory space by dividing the memory into uniform length blocks, and testing one cell from each block. If an adversary modifies w out of the b cells in the block, the likelihood that the modification will escape detection from a randomly selected cell is given in Equation 3.10.

$$Pr(Escape) = \left(\frac{b - w}{b}\right) \tag{3.10}$$

If the adversary divides the attack into w cells in v different blocks, Equation 3.11 shows the escape rate with randomly placed modified cells, evenly distributed among the v blocks. If the modified cells are not uniformly distributed, the detection rate increases above Equation 3.11. The block with the highest number of modified cells will dominate the escape rate. Diving deeper into the escape rate for randomly placed modified cells does not provide much improvement in the security of the concept as skilled adversaries will focus their attention on shadowing the modifications over the random approach to impact as few fingerprints as possible.

$$Pr(Escape) = \left(\frac{b-w}{b}\right)^{v} \tag{3.11}$$

As we continue to power up the system on subsequent boot cycles, randomly selecting one fingerprint to verify, Equation 3.12 shows the escape rate after the m^{th} check.

$$Pr(Escape) = \left(\frac{b-w}{b}\right)^{m\times v} \tag{3.12}$$

As the following analysis demonstrates, modifying even a modest number of cells will be quickly detected. An adversary with knowledge of the SSB algorithm can attempt to maximize the shadowing of the compromised cells. The first priority is to try to align the compromised cells in each successive block to coincide with fingerprints already monitoring compromised cells, minimizing the number of fingerprints that will check compromised cells. With 100% shadowing, each of the v modified code segments will be protected by the same fingerprints. We still will have at least w fingerprints that are compromised, so the escape rate is bounded by Equation 3.13.

$$Pr(Escape) \leq \left(\frac{b-w}{b}\right)^{m} \tag{3.13}$$

If we use column-wise fingerprint generation, it is easy for the attacker to achieve 100% shadowing, because the simple fingerprint pattern is known. We want to implement sufficient entropy in the fingerprint generation pattern so that it becomes very difficult for the attacker to achieve effective shadowing. To protect a single ECU, the modulo multiplication and full random patterns were most effective, as the attacker would have to not just know the starting offset, but also make very small code snippets. Using ECU-specific seeds makes the effort to uncover patterns and develop an effective attack prohibitively expensive, as this would probably entail destroying the ECU, preventing a scalable attack model.

3.4.2 Fingerprint Robustness

When we think about using a sampled rather than an exhaustive basis to ascertain that the SW image is intact, there are a few questions to consider.

- **New SW:** When an ECU receives new SW, it needs to verify the origin, freshness, and applicability of the SW. DSAs and asymmetric cryptography work well to establish sufficient levels of trust. Since this happens infrequently and SW updates can be postponed until the vehicle is in a safe state, the time it takes to verify with slow algorithms is warranted.

- **Primary Detection:** We would like to know how effective the scheme is at detecting compromised code. Exhaustive checks guarantee detection, but all sampling schemes, since they do not check every single cell, must evaluate the detection effectiveness in terms of probability of escaping detection. How effectively can the scheme detect the compromise on the first ignition cycle after introduction?

- **Secondary Detection:** Considering the case where the attacker is able to evade detection on the initial boot cycle, will it eventually detect the modification? Does the probability of escaping detection decrease on each subsequent ignition cycle?

- **Sustainable Detection:** Let us assume that the adversary has studied the behavior of a specific module and is able to build a profile of the fingerprints. Can the adversary determine a strategy to avoid detection altogether? How effectively can he shadow his modifications to reduce the detection rate?

- **Scalability:** Let us assume that the adversary invests significant effort into successfully compromising a single ECU. A scheme that can withstand significant attacks and then leave only a single ECU compromised may make the entire effort undesirable for many attackers.

3.4.3 Algorithm Selection

For our initial analysis, we identified the Cipher Block Chaining Message Authentication Code (CBC-MAC) algorithm as the fastest algorithm in most automotive controller HSM AES accelerators, as a high potential candidate for secure boot image checks. We investigated the improvement ratio observed in our sampled check over a full image check. We set the Initial Value (IV) to zero as is commonly defined for the CBC when used to calculate a MAC [76] [125] since a codebook attack is out of scope for a MAC. This algorithm requires the use of a dedicated key. Although length-based attacks present a challenge for the CBC-MAC algorithm in general, our use case posits a dedicated HSM core verifying on-chip memory against a key it controls. For the initial concept, the algorithm simply pads with zeros at the end of the last block if needed. Selecting a value of b that fills the AES block or reading additional memory cells could improve this implementation as a next step. The following measures provide robustness against an adversary modifying the SW image and fingerprints (MACs):

1. The HSM controls the padding, the IV, and the number of cells checked.
2. The HSM controls which memory cells are checked.
3. The HSM never reveals its calculation for the current fingerprint - only if it matches the stored fingerprint.
4. The HSM may reveal the original memory fingerprint values by storing them in unprotected memory to an adversary who is able to read out the memory.
5. The attacker cannot force the HSM to calculate a new fingerprint or reveal what the fingerprint should be for an arbitrary modification.

We do not have a second party that is trying to determine if the message has been modified in transit, but the same party running the same SW to verify if the image has been modified. Other MAC algorithms such as CMAC and HMAC could also be used if the AES HW accelerator supports them, but selecting a slower algorithm means checking fewer cells in the time available for the image check.

3.4.4 Escape Rate Detection

Sampled checks balance the number of cells modified (attack), the number of cells checked (time), and the escape rate (detection). Non-exhaustive checks cannot guarantee detection, so let's explore how to evaluate the detection rate. First, we recognize that from a security perspective, not all cells are the same. Researchers have successfully compromised ECUs by changing just a few cells on ECUs without secure boot checks [86] [124]. Cells that control secure access, diagnostic access, SW signatures, or modify safety-critical features may need to be checked on every boot cycle. Sampled checks are not effective at detecting single-cell changes, but exhaustive checks are not efficient at protecting large blocks of code. Can we efficiently check the remaining overwhelming majority of the cells—those that do not directly control security functions?

To evaluate the effectiveness of SSB, let z be the number of cells the adversary needs to modify to achieve the objective modified behavior and let us explore how they are configured. The security engineer can vary the minimum start up time improvement factor and the target maximum escape rate. The attacker attempts to balance the desired functionality and the minimum number of modified cells to achieve the functionality while hiding the small snippets in memory.

Fortunately SSB is very effective at detecting even a small number of modified cells. To make the analysis interesting, let's examine a very minimal modification, probably significantly smaller than an adversary would attempt to implement. Assume that the adversary needs to change 0.01% of the cells in a microcontroller with 8 megabytes of on-chip memory using 4-byte words as cells. So $z = \frac{0.01\% \times 8388608}{4} = 210$ words. With $b = 64$, what is the likelihood of detecting these compromised cells in the memory?

Let us assume that the adversary wants to create a contiguous block of executable code. If $z \geq b$ then every fingerprint will contain at least one modified cell, no matter how the attacker orients the compromised code, and detection is guaranteed. Familiar with the algorithm, the adversary can break up the code into v smaller segments, each linking to the next segment with a jump command, save the last. Each of the modified v segments will contain $w = \lceil \frac{z+v-1}{v} \rceil$ compromised cells. Consider the case where $w < b$, each of the v blocks with modified segments will contain w modified cells. If the compromised segment straddles two blocks, the impact on the escape rate remains unchanged.

Figure 3.2 depicts the escape rate over the first three ignition cycles as the number of segments with the different fingerprint generating patterns varies. We swept v from 1 to 63 and then calculated w to give $z = 210$ active modified cells. We generated the fingerprints using several fingerprint generating algorithms:

1. A simple column-wise algorithm with fingerprints made up from the key and each cell in the column from each block.
2. A modulo addition algorithm with each block i having a random $offset_i = \{0, \dots, 63\}$.
3. A modulo subtraction algorithm with each block i having a random offset and direction: $offset_i = \{0, \dots, 63\}$, $direction = \{-1, +1\}$.
4. A modulo multiplication algorithm with each block i having a random product: $product_i = \{1, 3, 5, \dots, 63\}$.

Figure 3.2 illustrates the results of 1,000,000 simulations of each configuration, modifying the v segments of code w cells long, then counting how many boot cycles it takes before the modification is discovered.

All four strategies provide similar protection on the first boot cycle. Since $v, w \in \mathbb{Z}$, their product, or the total number of modified cells, will often exceed our target value of $z = 210$, this changing total causes sawtooth ripples in the curves. We select the combination of $v = 11$ and $w = 20$ as an interesting configuration, as v increases, the additional jump commands increase the total number of compromised cells. The escape rate averages 1.6% on the first boot. On the second boot, the column-wise, addition, and subtraction strategies achieved an escape rate of

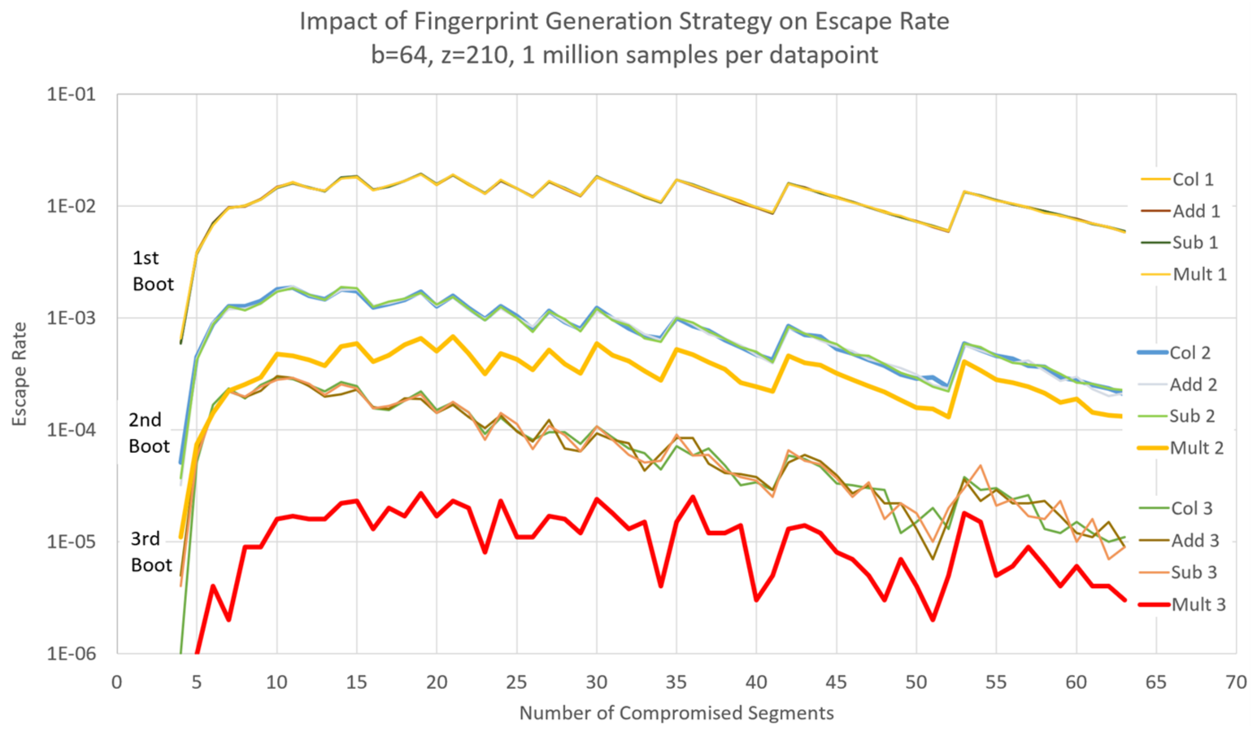


Figure 3.2: Escape Rate Fingerprint Generation

0.19%. The multiplication strategy achieved an escape rate about four times better with 0.046%. This significant improvement comes from more effectively spreading the contiguous strings into more fingerprints. On the third boot, the first three strategies achieved 0.029% escape rate, and the multiplication about ten times more effectively at 0.0017%.

Figure 3.3 investigates the behavior of this attack profile in more detail. We also include an analysis of a column-wise approach, where the adversary effectively shadows the compromised segments. If the adversary has the ability to arbitrarily place the modified code, adding entropy to the fingerprint generation provides a significant improvement to the detection. Randomly placing the compromised code in memory does not discriminate between column-wise, modulo addition and modulo subtraction, but we anticipate adversarial intelligence to implement significant shadowing with column-wise sampling. For column-wise, modulo addition, and subtraction, the first modified code segment affecting w fingerprints. When the second code segment is added, the portion of the compromised code that overlaps the first segment provides no benefit to detection – just the portion that extends either at the beginning or end of the original segment catches new fingerprints.

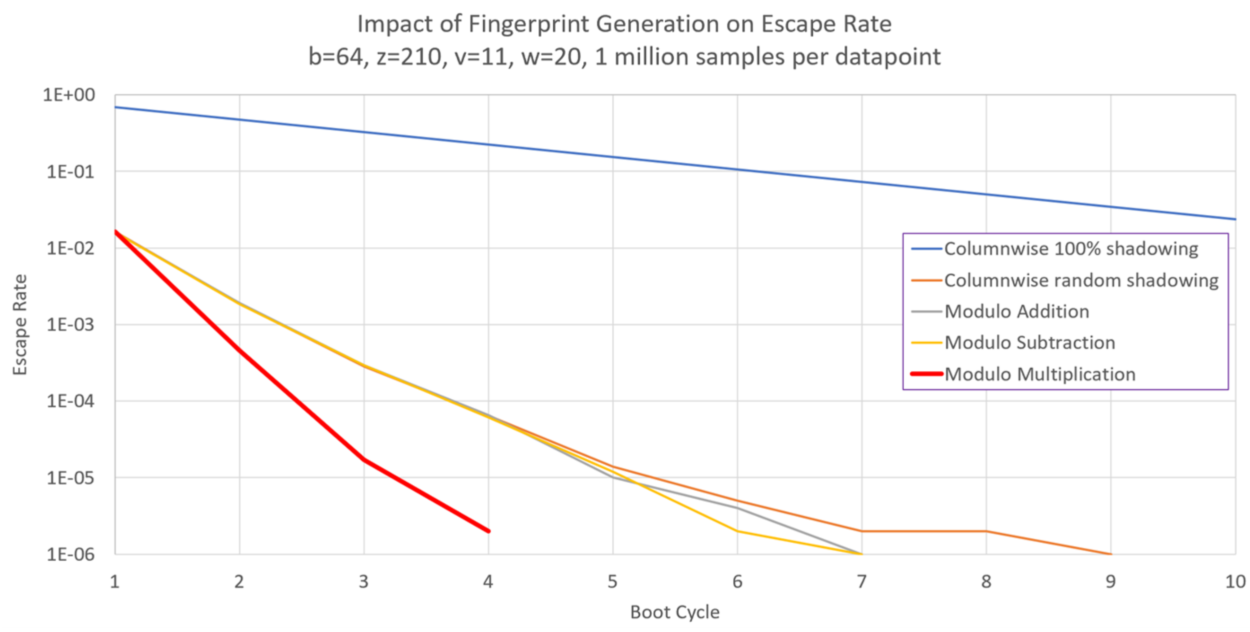


Figure 3.3: Impact of Fingerprint Generation on Escape Rate

Modulo subtraction provided no significant improvement over modulo addition because the effective shadowing is the same for both methods. If fingerprints 1 through 20 cover the first modified code segment, and then fingerprints 20 through 1 cover the second, it does not increase the number of fingerprints that catch a modified cell. So although the order of the fingerprints catching these cells changes, the shadowing does not improve. Modulo multiplication offers a significant advantage because the fingerprint coverage distributes the modified cells more effectively across multiple fingerprints.

There are $b!$ possible fingerprint combinations to protect each cell with one fingerprint. columnwise fingerprints provide 1 combination, modulo addition provides b, modulo subtraction provides $2b$, but only b are unique when considering shadowing. The tested modulo multiplication variant with just the odd factors gives $b/2$ unique combinations. To ensure that each cell is protected by a single fingerprint, we need to ensure that b and $index$ are relatively prime. Selecting a prime number for b allows $b - 1$ possible configurations, but multiplication in general, and modulo multiplication in particular are not trivial calculations for an HSM. A computationally feasible strategy would be to create a set of random arrangements and store a LUT in the protected memory.

Then a pseudorandom stream selects a configuration from the LUT for each block. The number of available configurations needs to be controlled, but our example with just 32 variants provided an attractive alternative - a relatively small number of patterns effectively distributes the modified cells into different fingerprints. Modulo addition with a power of 2 base is a trivial operation, as the higher order bits can simply be discarded.

If the adversary is able to monitor the memory patterns (e.g., with off-chip memory), identifying the blocks that use the same pattern allows effective shadowing. We can increase the resilience against this in two ways. The first would be to increase the number of patterns (which may be impractical with limited memory available). The second is to use multiple patterns to oversample the cells. For our modulo multiplication example, if we oversample with $f = 3b$, the redundant fingerprint patterns protect each cell with a different pattern, we would have $32 \times 31 \times 30 = 29,760$ different protection patterns. With 8 MB of memory arranged as 2,097,152 4-byte cells, we have 32,768 blocks. Even if the attacker can optimize an effective shadow for one set of fingerprints, the other fingerprint patterns would not be effectively shadowed and the unshadowed fingerprints would dominate in selection frequency.

Figure 3.4 shows the effectiveness of modulo multiplication increases over column-wise sampling as the number of compromised code segments increases. With just a single modified code segment $v = 1$, the two strategies have the same escape rate. When we increment to $v = 2$, modulo multiplication begins to demonstrate more effective shadowing resistance. As we continue adding modified code segments, modulo multiplication continues to increase its effectiveness as it more effectively casts the modified cells into different fingerprints. The bumps in the curves occur when the escape rates approach one sample out of the million simulations.

In Figure 3.5, we plot the curve fitting exponents from Figure 3.4 and the dramatic influence of effective scrambling on improving detection for randomly placed modified code segments.

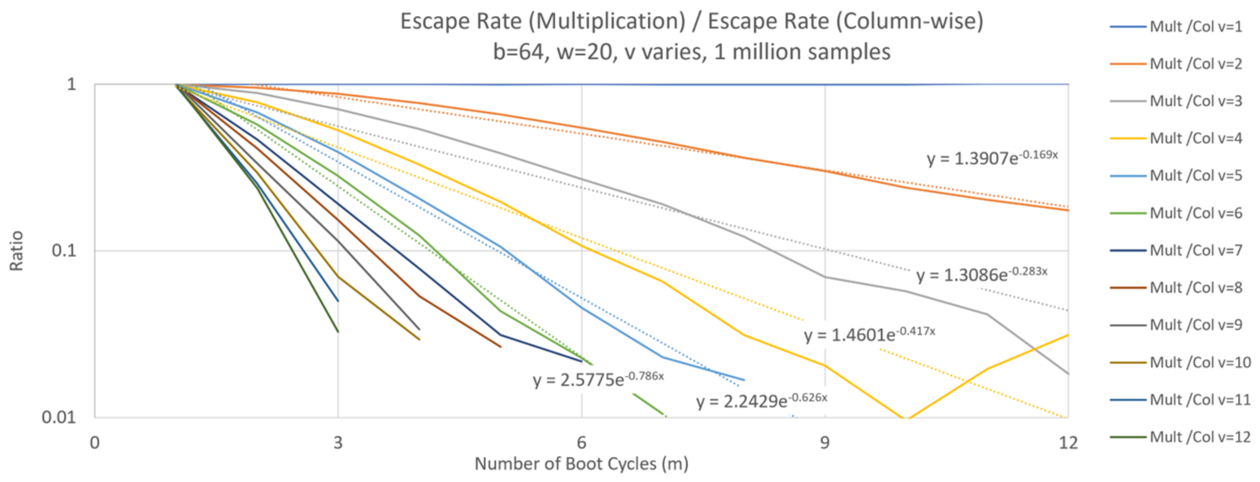


Figure 3.4: Escape Rate Multiplication vs. column-wise

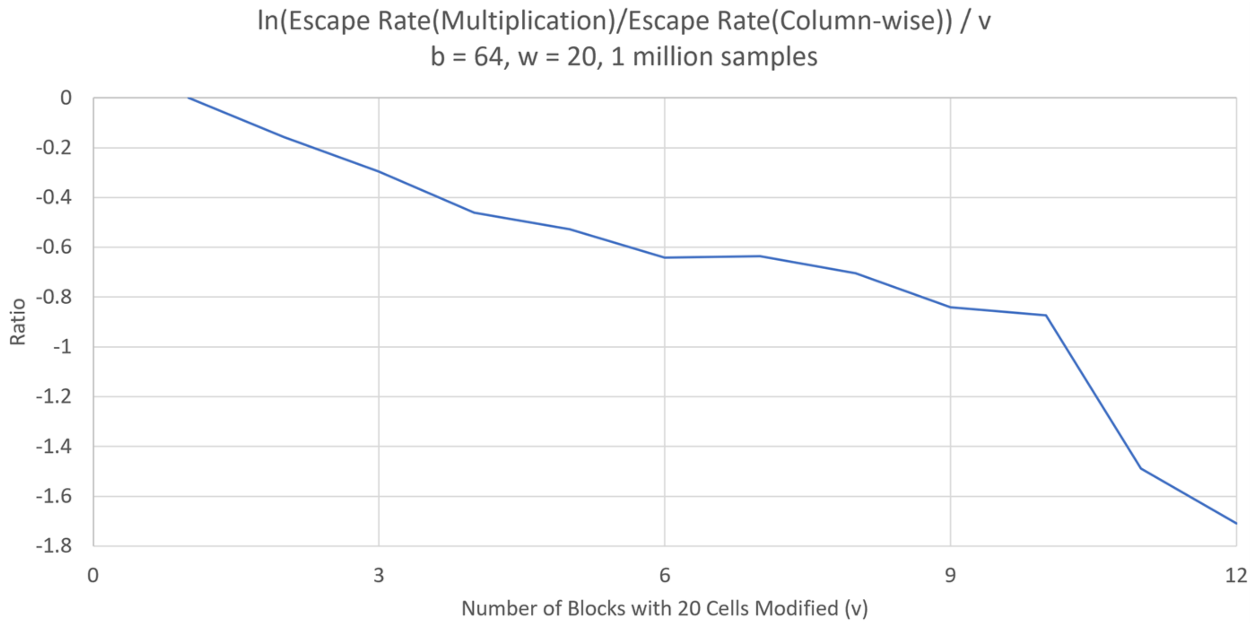


Figure 3.5: Escape Rate Multiplication vs. Escape Rate column-wise

3.4.5 Effective Scrambling

It is difficult to derive a closed form equation for the likelihood of shadowing, and it is not very useful, as the attacker is not using a random-based approach. What appears to be helpful, is if we can try to provide a set of patterns that meet the following criteria:

- **Unbiased and even distribution of fingerprints:** Ideally, we would like to see an unbiased mapping for each cell position into each fingerprint. Table A.1 shows the possible patterns of modulo multiplication with base 16. If we use the columns as memory cells, the rows as the offset factors, and the table values as the fingerprint that the cell should be mapped to, then it is clear that the even factors are not suitable (half of the fingerprints check no cell) and avoid $b \equiv 0 \mod b$ as well. The even numbers are not relatively prime and share 2 as a common factor with 16. One additional point to notice is column number 8. Since it always maps to the same fingerprint, this could provide a foothold to an attacker. Using a prime base, as in Table B.1 with $b = 17$, provides a perfectly balanced scrambling of the fingerprints. Each cell maps to one fingerprint and can map to any of the fingerprints. We used $b = 64$ to allow consistent evaluation with the other methods, but $b = 61$ would have been better for modulo multiplication or perhaps $b = 65$ with even offsets.

- **Different spacings:** Table B.1 provides also a balanced distribution of the spacing from adjacent cells to different fingerprints. If the adversary is constrained to use contiguous code segments, spreading out the spacing complicates shadowing. For a prime number base, this is automatically supported.

- **Sufficient entropy:** The number of possible fingerprint patterns should be sufficient to prevent an adversary from effectively shadowing compromised code segments.

- **Quick to calculate:** If the mathematics behind the implementation slows the HSM boot check, we will have to check fewer cells to compensate.

- **Threat scenario:** Can the adversary get access to the memory bus to determine which cells are linked in a given fingerprint? If we assume that the adversary cannot easily determine which cells are being protected by which fingerprint, the risk of effective shadowing is reduced. If we assume the attacker can easily monitor fingerprint usage, we may increase the number of patterns.

3.4.6 Impact of Fingerprint Distribution

How much could we improve the detection by optimizing the fingerprint patterns? To analyze this question, we compared the performance of modulo multiplication in Table A.1 and Table B.1. The first is labeled as Mult16, since we are using modulo multiplication with base 16, but only selecting the odd offset factors. The second is labeled Mult17. We generate a random offset factor between 1 and 16, then we generate a series of 5 cell long modifications that are between cells 1 and 16 (if it passes 16, then we wrap around to 1). We then multiply the factor and the modified cell in modulo 17, then subtract 1 to give us a range from 0 to 15 so that it is compatible with the other calculations. We do not want to multiply by 0, as this would always map all cells in the block to the same fingerprint.

For both models, we have 16 fingerprints. For Mult16, we only have 8 fingerprint ordering patterns (from the odd factors) and cell 8 always maps to fingerprint 8, and cell 16 always maps to fingerprint 0. For Mult17, we have 16 fingerprints. Figure 3.6 shows that Mult17 performs a bit better than Mult16, but even with 10 million samples per data point, the advantage with randomly placed compromised cells is not significant. Since our example uses a small number for the number of compromised segments v, it is still not likely that modified cells are in blocks and same locations using the same fingerprint pattern.

3.5 Performance Analysis

We prepared a proof-of-concept implementation running inside an HSM on five automotive ASIL-D rated microcontrollers. The three different Renesas RH850C family controllers [53] have the same

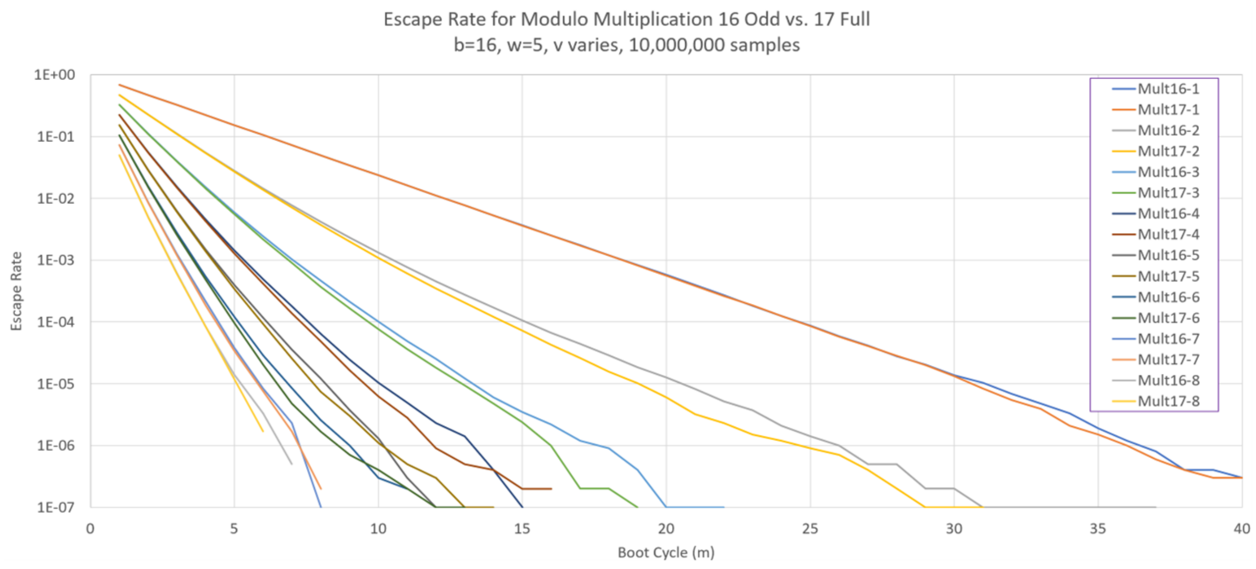


Figure 3.6: Escape Rate for Modulo Multiplication 16 Odd vs. 17 Full

HSM and behaved the same in testing, Infineon's TC37x and TC39x [91] likewise. We evaluated the influence of using a word or byte as the cell size, varied the block size, the amount of memory processed by the algorithm, and RAM allocated to the buffer. We reduced the verification time by a factor of 9 on the Infineon devices and a factor of 23 on the Renesas, bringing the boot time within a range compatible with the safety concept.

'Flash block' commonly refers to a physical group of memory cells that are erased together, but 'block' here refers to a logical group of memory cells; the investigation considered the block size ranging from 4 to 64 words. Microcontrollers are typically optimized to read contiguous sections of memory into the AES accelerator, which influences the Time Reduction Factor (TRF). We calculate TRF as the ratio of the time required to complete the **full image check** to a **sampled check**. Since $total\ memory = b \times d$, as the block size b increases, the number of cells checked and the time to check them decrease.

We use 'cell' as a collection of data, and then select the optimum size. Our initial measurements treated each byte as a cell, but the HW is optimized to read words (4 bytes) rather than bytes. Switching to words caused a 51% performance improvement. Using 16-byte long cells to feed the AES engine optimizes the read rate, but also reduces the effective snippet length w and thus the detection rate. Aligning the cell size to the smallest functional unit in the microcontroller maximizes the length of the code snippet. If a string of cells are compromised, checking all of them does not increase the detection rate over checking just one.

3.5.1 Complexity and Memory Cost

Our scheme sparingly uses HSM-protected memory and efficiently uses unprotected memory. For SSB, we store the 128-bit AES key (16 bytes) in protected memory. With $b = 64$, the fingerprints take 1 kB of memory and the proof-of-concept implementation stored these in protected memory. Storing the fingerprints in unprotected memory would allow the attacker to generate her own fingerprints, but she must first defeat the collision resistance of the AES key protecting the MAC to compute a fingerprint corresponding to the modification. Modulo addition requires an additional 128-bit seed stored in protected memory. Modulo subtraction requires the same amount of memory as the PRNG provides the values for the individual blocks.

Modulo multiplication also would require an additional seed, but we did not evaluate the performance impact, as it is likely to impact the HSM speed. Using a partial random LUT requires an index for each element, $b \times n \times log_2 b$, where n is the number of needed patterns plus a seed to be stored in protected memory but should have a negligible speed impact.

3.5.2 Time Reduction Factor

To evaluate the improvement in boot time, we propose TRF as a metric. This is simply the amount of time it took to perform a sampled boot check divided by the time it took to perform a complete boot check. Figure 3.7 plots the best-case TRF for the Infineon TC37x and the Renesas D3 microcontrollers against the block width b. We evaluated the influence of using a word or byte as the cell size, varied the block size, the amount of memory processed by the algorithm, and RAM allocated to the buffer. The verification time reduced by a factor of 9 on the Infineon devices and a factor of 23 on the Renesas, bringing the boot time within a range compatible with the safety concept. The influence of using a word or byte as the cell size, the block size, the amount of memory

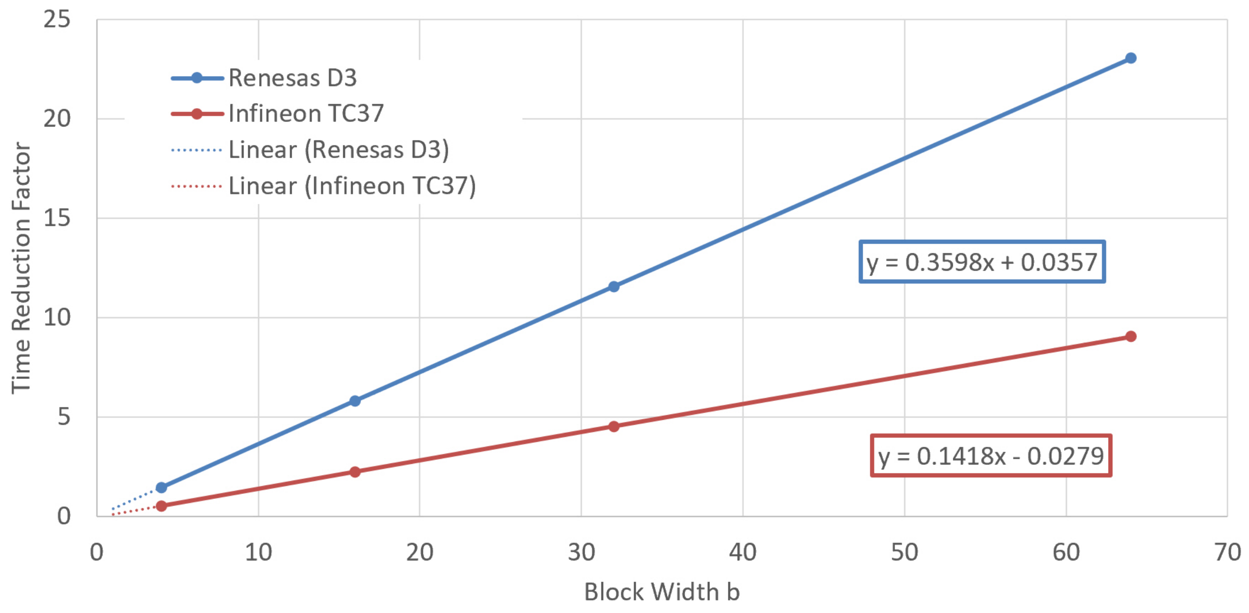


Figure 3.7: SSB Time Reduction vs. Block Width for Renesas D3 and Infineon TC37

processed by the algorithm, and the amount of RAM allocated to the buffer were analyzed. The configuration that provided the quickest verification time was selected and then used to calculate the TRF for the five microcontrollers.

We varied the block size from 4 to 64 at powers of 2. The TRF was identical for the microcontroller vendors as they use the same HSM block within the same family. The TRF increased directly proportional to the block size b, but overhead in loading the AES accelerator from memory attenuated the reduction. With a block size of $b = 64$, TRF was 9 on the Infineon devices and a factor of 23 on the Renesas ones, bringing the boot time within a range compatible with the safety concept.

For the Infineon TC37x and T39x, the TRF is $b/7.1$. With $b = 64$, this provided a TRF of 9.

3.5.3 Real-World Evaluation

Even if we persuade ourselves that SSB is robust against random constellations of modified cells, we need confidence that it will be robust against real-world attacks. Unable to evaluate publicly available real-world automotive modified SW, we checked Offensive Security's shell code exploit database [162] and found the smallest entries are slightly larger than 100 bytes. Dolezal et al. explored compromised automotive code [63] [64]. We examined a tuner-modified engine controller and compared the modified SW to the original SW and achieved 100% detection, i.e., each fingerprint checked at least one modified cell, up to $b = 2048$ with byte-sized cells. The tuners modified 0.08% of the memory space, but to achieve the modified engine controller performance, their modifications were clustered together. Of course, if the secure boot had been in place, they would have had to adapt their strategy. With only 0.08% of the memory space modified, the scheme delivered promising results on a real-world modification.

The deciding questions for successful introduction of this scheme are: "What is the smallest number blocks and cells does the adversary have to modify to achieve her desired behavior?" and "What is the minimum acceptable detection rate?"

3.6 Discussion

Several risk factors should be considered when using a sampled secure boot strategy.

- **Size:** It works well for larger modifications, but small modifications may escape detection [124]. A practical implementation will probably need to evaluate some cells with a higher sample rate, and some cells every boot cycle. A more complicated fingerprint pattern could also present an attack vector for a skilled attacker.

- **Entropy:** The ability of an attacker to escape detection strongly depends on the entropy of the patterns. Having unique fingerprint keys and fingerprint patterns for each ECU helps to limit the scalability of an attack and hence the attractiveness of the attack vector. Attacking the HSM entropy could present an interesting attack path, but it probably would also affect the standard secure boot strategy as well.

- **Objectives:** It is essential to determine a reasonable escape rate and use this to drive the parameter selection process. For most vehicle applications, a 99% likelihood of detecting compromised SW on a single boot cycle would offer sufficient protection to serve as a reasonable defense mechanism.

- **Asset Value:** Even with robust secure boot protection, embedded controller HSMs are not well suited to protect high value secrets such as fleet-wide keys. Researchers have successfully launched attacks on full image secure boot scenarios [154].

- **Response:** Responding to an anomaly detection with a decoupled, real-time, CPS requires careful planning. Security engineers are conditioned to try to bring the system to a secure state, but the safety needs must also be considered for automotive systems. This is especially important for RTD.

Several principles enable the use of reusable fingerprints to protect sampled secure boot.

- Constrain fingerprint generation to ensure uniform, unbiased protection, i.e., protect each cell by the same number of fingerprints.

- Select fingerprints randomly without bias.
- Protect the fingerprint signature with an ECU-unique key.
- Protect the fingerprint pattern with an ECU-unique key.
- Generate fingerprints when the ECU is in a protected state.
- The likelihood of escaping detection on the first boot cycle after modification should be low and decrease on each successive boot cycle, even if the adversary observes all fingerprint patterns before implementing the modifications.

3.7 Conclusions

The SSB-proposed scheme can be used to make secure boot feasible for automotive ECUs by reducing the time required to verify SW integrity to the point where secure boot does not interfere with the safety concept. Even without checking every cell in memory, the sampled check can provide an acceptable level of security to protect automotive controllers.

Using entropy in the fingerprint patterns can protect against shadowing, but several considerations come into play. Modulo addition provides a quick and lean pattern to protect against shadowing. Modulo multiplication and reduced random LUTs provide more time intensive and memory intensive respectively solutions that have significant improvements in certain configurations.

We see significant benefits from using SSB as a run-time detection algorithm, where we build trust incrementally in small chunks rather than hoping that a compromised host core will allow sufficient execution time to complete the image check before the power is shut off. As electric vehicles and autonomous vehicles modify driving cycle behavior, boot cycles may occur less frequently, increasing the need for run-time monitoring.

We mapped out a prospective state diagram for how to respond if the system detects a compromised memory image in a safe manner. It may be easier to reach a secure state in a safe manner by using a vehicle-level approach rather than trying to accomplish this with a compromised ECU.

The critical question facing automotive security engineers is not “is a sampled secure boot as good as a full one?” but the pragmatic one “Since we must meet our functional safety and start up time requirements, do we implement SSB protecting *this* ignition cycle or a full image background check protecting *the next*?” A defense mechanism that shrinks the likelihood of success to the point where the adversary decides this path is not worth the effort should be considered effective.

CHAPTER 4

Digital Shadows - Remote Attestation without SW Image

4.1 Introduction

Automobiles need a new approach to RA due to limited resources, network access, and intellectual property access. As automotive connectivity and autonomy increase, the need for external parties to trust the integrity of the SW in a vehicle also increases, even for parties without access to the complete SW. Consider a government regulator, without either physical access to the vehicle or access to confidential proprietary information, who needs to ensure all the autonomous vehicles driving on roads within its jurisdiction are using the correct SW. With 1.4 billion vehicles driving around the globe, and 284 million just in the US alone - if each vehicle sports 65 ECUs, verifying each vehicle twice a year would entail checking 100 million ECUs per day [22] [85] [174]. Other parties could also benefit; owners to verify that their vehicles haven't been hacked, a potential car buyer to ensure that the vehicle hasn't been running tuned SW causing excessive wear, a parent to "kick the tires" of a robotaxi before sending the kids off to school, and the vehicle manufacturer to periodically check the cyber-health of its fleet.

Classic RA is a process with two parties. A trusted device (the *verifier*) measures the current state of the device under test (the *prover*). RA in the automotive world faces several interesting challenges - the verifier neither has access to the known-good SW nor controls the individual vehicles that are being verified. Within this low-trust environment, the OEM may be the only party with access to the known-good SW. The intellectual property's value, the risk from increased cyber attacks, and liability with potential misuse complicate sharing with regulators or even owners. Parties beyond the OEMs may have a valid need to verify the authenticity and freshness before entrusting human lives into the hands of a vehicle's ECU. Furthermore, the adversary may have direct control over the target vehicle and the communication network to the prover. With physical access to the vehicle, the adversary may modify or copy the module's memory, add another device in the vehicle, or perhaps even capture or modify messages on the network back to the prover. Many automotive controllers have similar challenges to IoT devices with low processing power, intermittent supply power, and the ability to control cyber-physical systems and collect sensitive Personally Identifiable Information (PII) data [27].

The potential adversary objectives include malicious attempts to modify the vehicle SW to injure the passengers, commercial desires to unlock functionality or increase performance which could impact the safety or reliability of the vehicle, and benign neglect of a required updates or inadvertently loaded improper SW.

Pragmatic concerns limit the time and data available to support the process. Some background information with reasonable numbers may help security experts understand the challenges. Any process that takes more than about 10 seconds during the manufacturing process can have a significant impact on the manufacturing line and brings prohibitive costs. To be useful, the check in the field should not take more than 30 minutes. Limiting the government database to 4 terabytes and the data exchange to support all requests to less than 1 terabyte per day keeps the storage requirement manageable with off-the-shelf devices. The concept needs the flexibility to support SW updates during the life of the vehicle.

We propose DS which combines 4 elements to form a concept that addresses these concerns. Firstly, it realizes that new vehicle electrical architecture topologies, with careful tweaking, can be aligned into a tree structure with a root node and multiple layers connecting all ECUs in the vehicle. It then uses a seed and a unique key to incrementally build a MAC-based signature for each module as a low-priority background task, with each ECU calculating its signature in parallel. Inspired by

Merkle trees [126], each module sends its signature to its parent node, which then builds a signature based on its fingerprint and all its children, then sends this to its parent node. This repeats until the signature for the root node is computed. We call this root-node signature a DS - a cryptographically robust signature of the entire vehicle which can be safely shared with third parties, over untrusted networks, without leaking sensitive data.

Secondly, we assume that the OEMs are already working to implement DTs of the entire vehicle fleet for reasons other than security. A DT is a connected virtual copy of the vehicle that can keep track, in high resolution, of the status and health of the vehicle [82]. Then the DT computes the OEM's vehicle target and send it DS to the prover. The costs required to setup and maintain the DT architecture are already covered with these other uses, just the differential cost for supporting the third party verifications needs to be considered. A regulator could mandate that all vehicles rated SAE level 3 [148] or higher could implement the DS algorithm.

Thirdly, the prover compares the DS from the vehicle with the DS from the OEM and determines if the signatures match with reasonable trust requirements. The prover initiates the process and performs the comparison.

Fourthly, all ECUs compute their signature concurrently rather than sequentially, so the DS will be completed shortly after the slowest ECU completes.

We evaluated the time and data loads for this concept with a sample of key vehicle ECUs. This concept will not immediately cover every vehicle, but the ones with the highest levels of autonomy and connectivity present the greatest need for remote authentication also are the most likely to have a DT supporting them.

4.1.1 Contributions

The noteworthy contributions of this chapter are:

- **Access to sensitive IP:** The DS is built without the verifier requiring access to sensitive IP.

- **Data leakage:** The DS does not leak any sensitive data about the state of the vehicle or anything contained within such as location history or driving patterns. There are no privacy concerns with a third party viewing the DS.

- **Usage:** By incorporating run time and fault code information, the DS can be used to verify that the vehicle does not just possess the right SW, but is actually using it, and has been since the last check.

- **Data:** We provide measurements and estimates of the time required to build a DS and the data required to implement the concept.

4.1.2 Organization

This chapter is organized with Section 4.2 introducing and analyzing the initial scheme. Section 4.3 explores the first improvement against replay attacks. Section 4.4 includes the verification that the SW is actually controlling the vehicle. Section 4.5 analyzes the data and time, and Section 4.7 concludes the chapter.

4.2 Initial Scheme

4.2.1 Remote Attestation Concept

Figure 4.1 illustrates a classic RA-based scheme where the vehicle (prover) has to submit evidence to the OEM (verifier), who has a complete copy of the desired vehicle SW and performs the check.

- Our case considers three parties: the physical vehicle, the OEM with the known-good SW, and the regulator who wants to verify the authenticity of the vehicle SW. We need to expand the classic RA scheme to support a third party without access to the desired SW.

Figure 4.2 illustrates a classic PDP scheme where the OEM can designate a third party (who never has complete access to the vehicle SW) to periodically authenticate the SW.

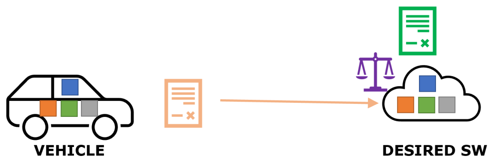


Figure 4.1: Classic Remote Attestation Model

- Pre-computing a batch of fingerprints is not suitable for our concept, because the OEM does not know when the regulator will want to authenticate the SW or when updates will occur.
- Storing the pre-computed fingerprints would require a prohibitively large amount of memory, with each ECU in each vehicle requiring a lifetime's worth of fingerprints.
- The responsibilities and relationships differ in our example. The OEM does not assign the regulator the responsibility to authenticate the SW.

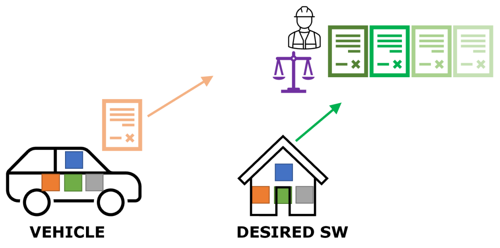


Figure 4.2: Classic Proof Data Possession Model

Figure 4.3 illustrates our hybrid concept, where we have a government regulator as the third party authenticating the SW for the vehicle.

- In this concept, the regulator controls the process, determining the time for each vehicle to perform the attestation.
- The regulator also conducts the comparison, minimizing the trust each party needs to place on the others.
- In addition to limiting the trust, practical considerations mandate a tight rein on the amount of data exchanged.

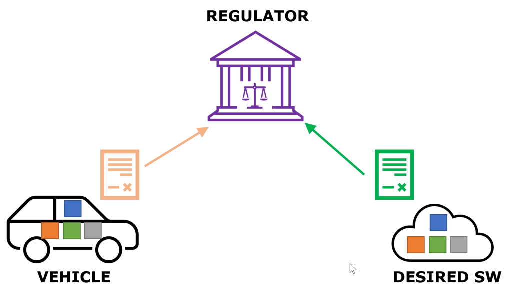


Figure 4.3: Hybrid Digital Signature Model

4.2.2 Trust Model

The trust model drives the need for a new concept. What can be shared and what cannot? How trustworthy and reliable are the communication paths? Why are we concerned about how much can be shared? The OEM is responsible for generating, validating, and distributing the SW to the individual vehicles. The enormous amount of innovation that goes into modern vehicle SW and the effort that goes into validation makes OEMs uncomfortable jeopardizing the intellectual property associated with this investment. Sharing the SW image presents a substantial risk, as the end user and regulator may not possess the same motivation and capabilities to keep it confidential. Although most vehicle controllers do not encrypt their SW at the moment, this may change. Regardless, there is a significant amount of effort required to extract the SW from an ECU inside a vehicle.

We assume that the government regulator trusts the OEM to deliver the current vehicle address, maintain an accurate and authentic digital twin of the critical ECUs in each vehicle, compute the DS on request, and communicate the DS back to the government.

We cannot assume that the communication network will maintain confidentiality, integrity, or accessibility. Supplemental controls may be required to force the vehicle to complete the test before being granted access to drive on certain roads.

The security check may not endanger vehicle safety and may not execute with high priority when the vehicle is driving. It may execute as a low-priority background task.

4.2.3 Hash-based Digital Shadow

We define a DS as a cryptographically unique one-way identifier of the SW image with the following characteristics:

1. The value should be identical if it is calculated by the actual vehicle or the digital twin.
2. The DS can be openly shared and will not divulge sensitive information about the vehicle.
3. An adversary should not be able to determine the DS fingerprint with knowledge of the SW.
4. An adversary should not be able to determine the SW with knowledge of the DS fingerprint and the algorithm.
5. The scheme should support FOTA updates and dynamic configuration of software defined vehicles.

4.2.4 Adapting a Merkle Tree for Automotive Architecture

Figure 4.4 illustrates a classic Merkle tree. Merkle proposed that each leaf node would calculate a hash of its memory and submit it to its parent branch [126]. Each parent branch computes a hash of its two children and submits it to its parent branch. This pattern continues until it reaches a root node. A Merkle tree is characterized by this binary tree structure. Traditionally, if a leaf's memory pattern changes, a small number of branch nodes hashes need to be recalculated, allowing a logarithmic amount of effort to compute the hash of the entire memory space.

Figure 4.5 depicts a reference vehicle architecture with a similar tree structure. A telematics unit connects to the outside world, redundant vehicle gateways connect the domains, and multiple domain controllers connect a few ECUs in their domain. We propose adapting the Merkle tree structure to reflect the architecture of the vehicle. Merkle's concept needs to be expanded to allow more than two children for each branch. This will not support all of Merkle's original objectives, but offers natural potential to efficiently build a unique signature for the vehicle network.

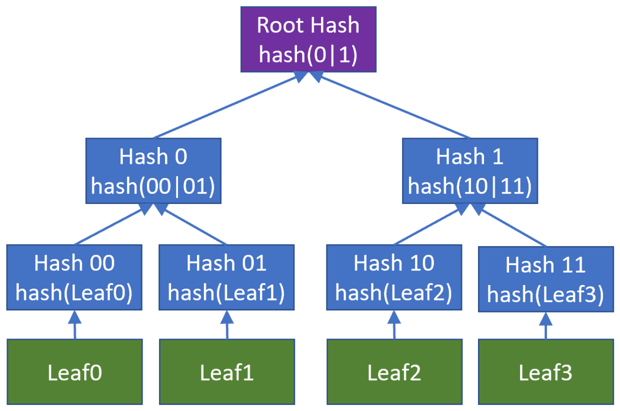


Figure 4.4: Classic Merkle Tree

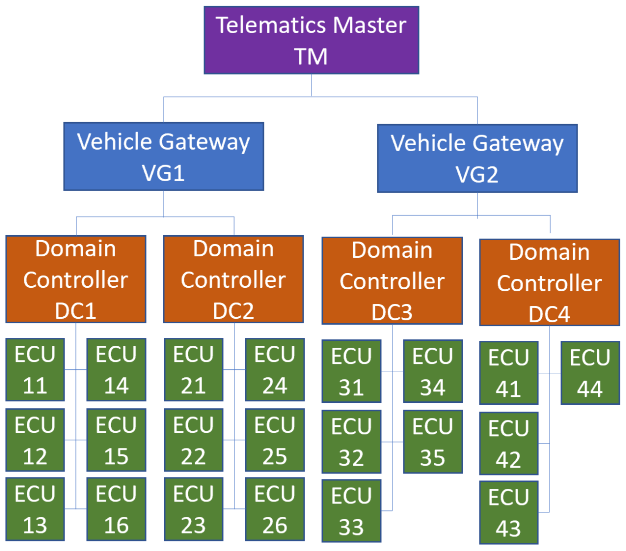


Figure 4.5: Vehicle Merkle Tree

4.2.5 Algorithm

For our initial attempt to solve this problem, let's consider a simple approach that allows the regulator to determine if the vehicle SW matches the OEM's expectation.

Let's assume that the OEM has a record of ADR_{VIN}, the address to communicate with each vehicle using it's unique Vehicle Identification Number (VIN). Since the address may change, we'll have the OEM keep track of the address for each vehicle and have the regulator request this from the OEM. To ensure identity authenticity, we assume that both the OEM and the regulator have generated asymmetric key pairs and exchanged the public keys. We also assume that the OEM maintains a DT of each vehicle, possibly for other reasons such as enabling predictive maintenance [205].

We build our DS by creating a hash of the memory space for each ECU. Each ECU then sends the hash to its parent ECU in the vehicle network. The parent builds a hash of its memory concatenated with the hashes of its children and passes it up to its parent. This process repeats until the root node computes the hash of its hash plus its children, which forms the DS. The root node then sends the hash to the regulator, who verifies that the DS of the physical vehicle matches one from the DT.

4.2.5.1 Regulator Algorithm

The regulator initiates the process by obtaining the address for the vehicle from the OEM. To prevent an adversary from misdirecting the request, the exchange is protected by encryption and authentication using pre-exchanged public keys. The regulator generates a random nonce to prevent replay attacks.

Algorithm 4: Digital Shadow Remote Attestation

```
Input: $VIN$
Output: $\{PASS, FAIL\}$
Known: $REG_{PRI}, OEM_{PUB}$
begin
    Generate random $NONCE$
    $SIGN_{OEM} = Sign(REG_{PRI}; VIN||NONCE)$
    Send $Encrypt(OEM_{PUB}; VIN||NONCE||SIGN_{OEM})$ to OEM
    Get $ADR_{VIN}, DS_{DT}$ from OEM
    Decrypt using $REG_{PRI}$, verify using $OEM_{PUB}$
    Send $Sign(REG_{PRI;VIN||NONCE)})||NONCE)$ to $ADR_{VIN}$
    Get encrypted $DS_{VEH}$ from $ADR_{VIN}$
    if $Decrypt(REG_{PRI}; DS_{VEH}) = Decrypt(REG_{PRI}; DS_{DT})$ then
        Return $PASS$
    else
        Return $FAIL$
    end
end
```

4.2.5.2 DS Calculation Algorithm

The ECU corresponding to the root node of the vehicle, our example in Figure 4.5 uses the Telematics Master. This ECU is responsible for communicating with the regulator, directing all ECUs to compute their hashes, building the DS for the entire vehicle, and also managing the encryption and authentication with the regulator.

Algorithm 5: Calculate Digital Signature

```
Input: $SIGNATURE$
Output: $DS$
Known: $REG_{PUB}, Num_{Child}, TM_{KEY}, Num_{Mem}$
begin
    Verify $SIGNATURE$ with $REG_{PUB}$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Send CALCULATE-DS command to child $ECU_i$
    end
    $hash_{TM} \leftarrow HASH(TM(1)||TM(2)|| \ldots ||TM(Num_{Mem}))$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Get $hash_i$ from child $ECU_i$
    end
    $DS \leftarrow HASH(hash_{TM}||hash_1|| \ldots ||hash_{Num_{Child}})$
    Send $Encrypt(REG_{PUB}; DS)$ to Regulator
end
```

4.2.5.3 Branch Calculation Algorithm

Each branch receives the CALCULATE-DS command from its parent and sends it to its children. It computes the hash of its memory space, concatenates the hashes of its children, and sends it to its parent.

Algorithm 6: Build Branch Hash

```
Output: $hash_x$
Known: $Num_{Child}, Num_{Mem}$
begin
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Send CALCULATE-DS command to child $ECU_i$
    end
    $hash_{ECU} \leftarrow HASH(ECU(1)||ECU(2)|| \ldots ||ECU(Num_{Mem}))$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Get $hash_i$ from child $ECU_i$
    end
    $hash_{Node} \leftarrow HASH(hash_{ECU}||hash_1|| \ldots ||hash_{Num_{Child}})$
    Send $hash_{Node}$ to Parent
end
```

4.2.5.4 Leaf Calculation Algorithm

Each ECU calculates the hash of its actively protected memory space. It then sends the hash up to its parent ECU.

Algorithm 7: Build Leaf Hash

```
Output: $hash_x$
Known: $Num_{Mem}$
begin
    $hash_{ECU} \leftarrow HASH(ECU(1)||ECU(2)|| \ldots ||ECU(Num_{Mem}))$
    Send $hash_{ECU}$ to Parent
end
```

4.2.6 Security Analysis

4.2.6.1 Security Robustness

The concept meets the first security criterion, namely we calculate the digital shadow based on the critical ECUs in both the vehicle and the digital twin. With the digital twin's modeling of the vehicle, the hash calculations should be identical if the SW is identical, and different if even a checked single byte in just a single ECU is different.

The DS can also be freely shared, as the one-way calculation nature of the hash function allows the hash to be shared without jeopardizing the content that goes into generating it - satisfying criteria two and four.

The solution supports the fifth criterion, as the on-demand check verifies the current SW against the OEM's desired SW. This eliminates the need for maintaining an elaborate database of SW in each vehicle and ensuring it remains up to date.

This concept fails the third criterion however, as the hash function is well documented and provides repeatable results for anyone who runs the calculation. Any party that can access the ECU and read the SW can consistently generate the correct hash. A skilled adversary would be able to read the memory and calculate the hash or just request the hash from the ECU, record the response, and then simply submit the correct response whenever the request occurs.

An additional weakness is that an adversary with control of the communication bus could monitor the computed correct answer from the OEM and then submit this result as if it were coming from the vehicle.

4.2.6.2 Calculation Time

By running the hash calculations for each ECU in parallel, building the DS takes slightly more time than the slowest ECU. This represents a significant improvement over individually checking each ECU individually.

4.3 Improved Scheme

4.3.1 Algorithm Improvements

The simple concept's main drawback is a lack of robustness against replay attacks. We need to force the ECU to check *each physical memory cell* in the ECU for *each request*. We also need to ensure that each request generates a different correct answer.

We propose three additions to the scheme.

1. **Key-Based Signature:** Instead of using a hash function to compress the data, providing a fingerprint without a secret, we swap the cryptographic primitive and use a MAC instead. A MAC requires each ECU to have a unique key. While it may be prudent to avoid sharing keys outside the vehicle, for this concept to work, this key needs to be stored in both the vehicle and in the DT. This will prevent an adversary from being able to calculate the correct answer without knowing the key.

2. **Random Seed:** Replay protection requires each request to have a different correct answer. A simple way to support this is to vary the starting position for each check. A typical small uC contains one to four megabytes of on-chip memory, providing a million different patterns. With an expected life of 25 years, checking twice per year will only use 50 of these patterns.

3. **Delay Correct Answer:** In order to prevent an adversary from capturing the correct answer from OEM and replaying it, the regulator waits to request the answer from the OEM until it has received the response from the vehicle.

4. **Run Time Measurement:** We include the run time for each ECU. It needs to be authenticated with the ECU's secret key to prevent falsification by an adversary.

5. **Authenticate Requests:** Only perform the calculation if the request can be authenticated from the regulator to prevent an adversary from scanning the memory.

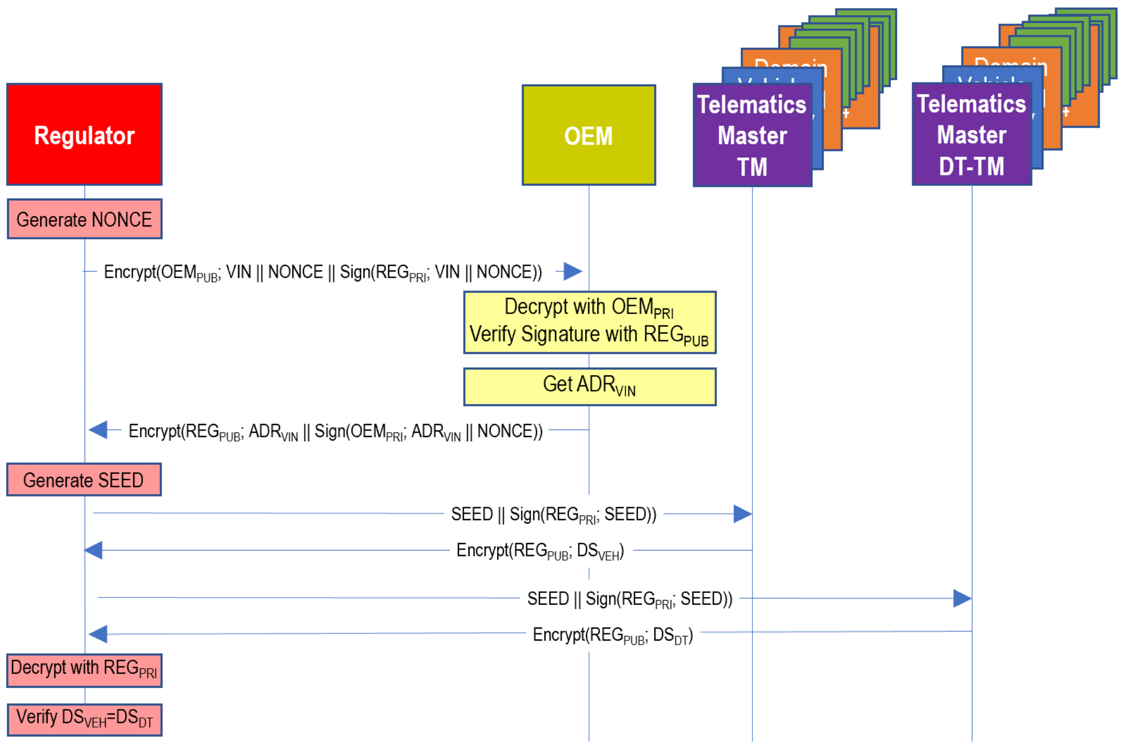

Regulator
OEM
Telematics Master TM
Telematics Master DT-TM
Generate NONCE
Encrypt(OEM_PUB; VIN || NONCE || Sign(REG_PRI; VIN || NONCE))
Decrypt with OEM_PRI
Verify Signature with REG_PUB
Get ADR_VIN
Encrypt(REG_PUB; ADR_VIN || Sign(OEM_PRI; ADR_VIN || NONCE))
Generate SEED
SEED || Sign(REG_PRI; SEED))
Encrypt(REG_PUB; DS_VEH)
SEED || Sign(REG_PRI; SEED))
Encrypt(REG_PUB; DS_DT)
Decrypt with REG_PRI
Verify DS_VEH=DS_DT

Figure 4.6: Regulator Perspective

4.3.2 Regulator Algorithm

Figure 4.6 shows that the regulator initiates and controls the process. Firstly, the regulator generates a $NONCE$ and requests the VIN from OEM similar to the original algorithm. We have the regulator generate a random $SEED$ and also not request the DS from the OEM until it receives it from the vehicle. And lastly, the regulator performs the check to compare that the two digital shadows match.

Algorithm 8: Remote Attestation with Digital Shadow

```
Input: $VIN$
Output: $\{PASS, FAIL\}$
Known: $REG_{PRI}, OEM_{PUB}$
begin
    Generate random $NONCE$
    $SIGN_{OEM} = Sign(REG_{PRI}; VIN||NONCE)$
    Send $Encrypt(OEM_{PUB}; VIN||NONCE||SIGN_{OEM})$ to OEM
    Get $ADR_{VIN}$ from OEM
    Decrypt using $REG_{PRI}$, verify using $OEM_{PUB}$
    Generate random $SEED$
    Send $SEED, Sign(REG_{PRI}; SEED)$ to $ADR_{VIN}$
    Get encrypted $DS_{VEH}$ from $ADR_{VIN}$
    Send $SEED, Sign(REG_{PRI}; SEED)$ to OEM
    Get encrypted $DS_{DT}$ from OEM
    if $Decrypt(REG_{PRI}; DS_{VEH}) = Decrypt(REG_{PRI}; DS_{DT})$ then
        Return $PASS$
    else
        Return $FAIL$
    end
end
```

4.3.3 DS Calculation Algorithm

Figure 4.7 shows the algorithm from the Telematics Master or root's perspective. For the DS calculation, we change from the hash to the MAC cryptographic primitive and include the secret KEY. The telematics root node also sends the $SEED$ to each child, then collects the signature from each of its children and builds the DS for the entire vehicle.

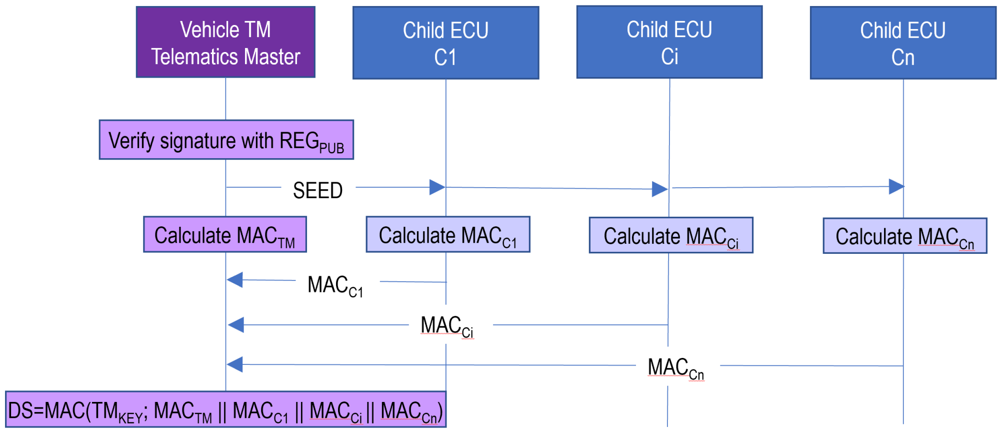


Figure 4.7: Root ECU Perspective

Algorithm 9: Calculate Digital Signature

```
Input: $SEED, SIGNATURE$
Output: $DS$
Known: $REG_{PUB}, Num_{Child}, TM_{KEY}, Num_{Mem}$
begin
    Get $SEED$ from Regulator
    Verify vehicle in safe state
    Verify $SIGNATURE$ with $REG_{PUB}$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Send $SEED$ to child $ECU_i$
    end
    $MAC_{TM} \leftarrow MAC(TM_{KEY}; TM(SEED)||TM(SEED +$
     $1)|| \ldots ||TM(Num_{Mem})||TM(1)|| \ldots ||TM(SEED - 1))$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Get $MAC_i$ from child $ECU_i$
    end
    $DS \leftarrow MAC(TM_{KEY}; MAC_{TM}||MAC_1|| \ldots ||MAC_{Num_{Child}})$
    Send $Encrypt(REG_{PUB}; DS)$ to Regulator
end
```

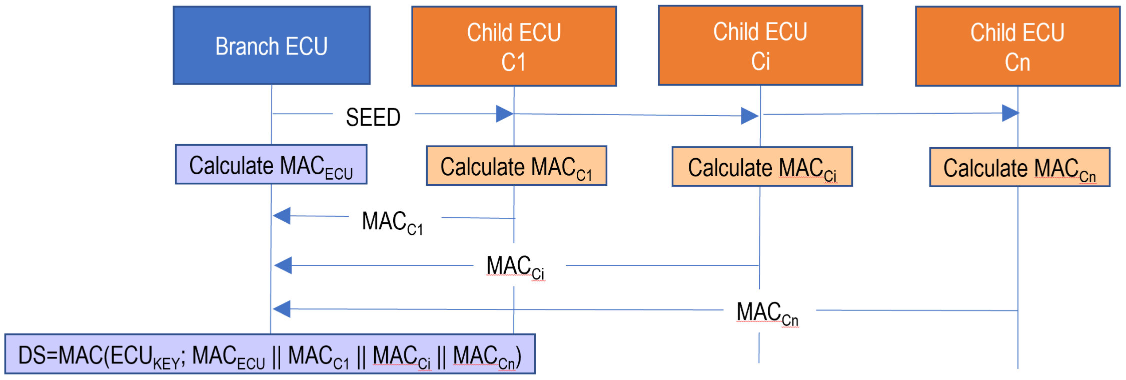


Figure 4.8: Branch ECU Perspective

4.3.4 Branch Calculation Algorithm

Figure 4.8 illustrates the exchange between a branch ECU and its children. Calculating each branch proceeds similar to in the simple algorithm, but using the MAC, ECU_{KEY}, and $SEED$ in its calculation and sending the $SEED$ to all its children. After the children compute their MAC, the branch node combines them together using its secret key.

Algorithm 10: Build Branch MAC

```
Input: $SEED$
Output: $MAC_x$
Known: $Num_{Child}, ECU_{KEY}, Num_{Mem}$
begin
    Get $SEED$ from Parent ECU
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Send $SEED$ to child $ECU_i$
    end
    $MAC_{ECU} \leftarrow MAC(ECU_{KEY}; ECU(SEED)||ECU(SEED+1)|| \ldots$
      $||ECU(Num_{Mem})||ECU(1)|| \ldots ||ECU(SEED-1))$
    for $i \in \{1, \ldots, Num_{Child}\}$ do
        Get $MAC_i$ from child $ECU_i$
    end
    $MAC_{Node} \leftarrow MAC(ECU_{KEY}; MAC_{ECU}||MAC_1|| \ldots ||MAC_{Num_{Child}})$
    Send $MAC_{Node}$ to Parent
end
```

4.3.5 Node Calculation Algorithm

Figure 4.8 also shows the calculation of inside the node ECU. Here, we make two changes: we use the MAC with the secret ECU_{KEY} instead of the hash function. And we also vary the starting location for the memory scan. The MAC calculation starts checking the memory cells at memory location $SEED$ and counts up to Num_{mem}, the last memory cell, then wraps around starting at the beginning and counting up to $SEED - 1$. We want $SEED$ to be a random number up to the memory size of the largest ECU, again to make it more difficult to launch a replay attack with external memory. This would require that we truncate $SEED$ for smaller ECUs.

We did not describe the process of generating or exchanging the ECU_{KEY} between the physical

ECU and the virtual, as this process may vary significantly from OEM to OEM depending on their DT-concept.

Algorithm 11: Build Node MAC

```
Input: $SEED$
Output: $MAC_x$
Known: $ECU_{KEY}, Num_{Mem}$
begin
    Get $SEED$ from Parent ECU
    $MAC_{ECU} \leftarrow MAC(ECU_{KEY}; ECU(SEED)||ECU(SEED+1)||\dots$
      $||ECU(Num_{Mem})||ECU(1)||\dots||ECU(SEED-1))$
    Send $MAC_{ECU}$ to Parent
end
```

4.3.6 Security Analysis

The modifications made in the second approach address the weakness against replay attacks. By using a more appropriate cryptographic primitive, ECU-unique keys, and randomizing the scan starting memory cell, we can ensure that each request will generate a unique DS. Since the regulator initiates the request, having the DT wait to submit the 'correct' response until after the vehicle sends its response also prevents an adversary with control of the network from being able to falsify the correct response as coming from the vehicle.

While the initial scheme did not address our third criterion where an adversary should not be capable of determining the DS even with access to the SW, this implementation does. Without the secret key, an adversary is not able to determine what the correct answer should be. The regulator can ensure that the $SEED$ is not reused, either by maintaining a log or having a space sufficiently large to keep the probability of reuse small enough to make such an attack unlikely to succeed.

By having the regulator sign each request, we also prevent an adversary from being able to pre-compute all possible responses. An adversary with physical access to the vehicle could remove an ECU and scan through the entire memory space, store each result, and then replay them without knowing the key. This would be effective until a SW update occurs. To prevent this attack, we could have critical ECUs also verify the signature from the regulator, rather than relying on the telematics master to authenticate all requests.

4.4 Expanded Algorithm to Address HW Status

4.4.1 Problem Definition

As digital controllers proliferate in number and authority in influencing automobile behavior, people using and regulating these vehicles expect commensurate trust in these advanced systems. What does it mean to verify the integrity of the controllers? Building on the concept of control flow analysis to monitor that the SW is executing in the proper order [57], we want the vehicle to demonstrate that it is capable of making proper, predictable decisions and that is able to execute the intended actions. These objectives all need to be fulfilled without forcing the vehicle or the manufacturer to share sensitive information or exchange large amounts of data. Beyond verifying that the vehicle contains the correct SW, we want to verify that this correct SW is actually controlling the vehicle. We can build the required trust by forcing the system to remotely attest the following properties:

- **SW Integrity:** Does the memory inside the physical ECU $\mathbb{M}$, match the memory that the OEM expects $\mathbb{M}'$? The program memory and the operational run-time memory may need to be considered separately.
- **Current HW Status:** Does the current state of the physical ECU $\mathbb{S}$, match the state that the OEM expects $\mathbb{S}'$?
- **Operation Decisions:** Since automotive ECUs make safety-critical decisions that impact the real world, we want to verify that the vehicle as a whole and the individual advanced systems

are making the correct decisions about how to operate the vehicle. Are the ECUs functioning internally as intended?

- **Decision Execution:** In addition to making correct decisions, we also require the ECUs to demonstrate that they are actually controlling the mechanical systems in the vehicle. Are the ECUs controlling the vehicle as intended?

- **Recent History:** We want to prevent an adversary from keeping the ECU (or a copy of it) in the vehicle to support an attestation request. We want to make sure that the ECU has actually been controlling the vehicle, for the entire time since the last attestation.

The attestation should not divulge sensitive information to the parties involved or to anyone listening. Consider protecting the vehicle SW intellectual property, driver profiles and location data, and the cost of sharing data over a mobile connection.

4.4.2 Concept Approach

What can we measure to force the system to attest these properties? To verify **SW integrity**, we build a DS of the entire memory space of each ECU for the entire vehicle and submit it to the third party verifier. We have the trusted OEM prepare a similar summary for the intended state and submit it to the verifier also. The verifier compares the two, verifing that the memory is in its intended state.

To verify the **state of the HW**, we could think about checking the transient RAM, but the DT does not possess sufficient granularity to accurately model the current running SW with a sub-millisecond resolution. Instead a combination of internal plausibility checks of the fault codes and the cumulative run time, together with an external plausibility can check against the DT to provide a good indication of the current state. The fault codes are the result of the periodic monitoring systems to verify that the system is properly connected. The correct run time without errors can confirm that the ECU is actually controlling the vehicle. With a tight link between vehicle operation and run time, crosschecking the run time across all the ECUs can demonstrate

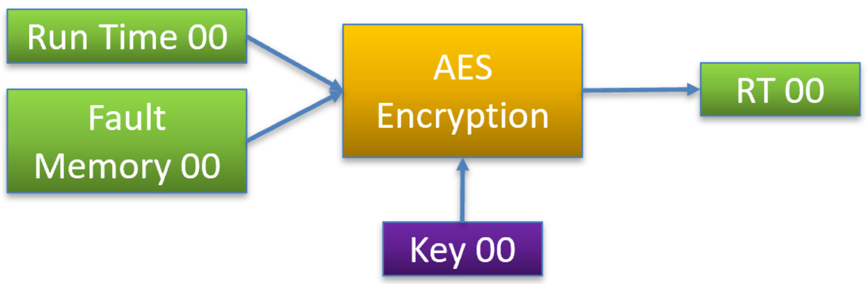


Figure 4.9: How to Build RT (Encrypted Run Time)

that they all have been running together. If no fault codes have been set and the run times match, we can be confident of the current HW status.

The precise times will vary slightly, scrambling them in a hash will not allow reconstruction. To enable this range comparison, we include the run time for each ECU outside of the hash. Figure 4.9 illustrates that we collect the run time and fault memory for each ECU, then encrypt them with an ECU-specific key. Figure 4.10 shows how to build the DS by hashing the memory signatures together, but include each ECU's encrypted run time data. The verifier can check the run time of each ECU and verify that the differences within the vehicle are within an acceptable range. We encrypt the run time for each ECU with its individual symmetric key to prevent an attacker from falsifying the correct value and provide the encrypted run times for each ECU to the DT, who can check the run time of each ECU and test for anomalous behavior against other vehicles.

We build the DS containing the memory fingerprint as before, but now we take the individual ECU encrypted run times and send all of them up to the parent node. This test also attests to the **decision execution** and **recent history**, all with **exchanging a limited amount of data**.

We could measure the ability to make **good decisions** by submitting new problems to the vehicle and ECUs and evaluating their response, but a mature DT model already receives detailed information about what the vehicle is perceiving. A remote service could measure both functional safety and product security aspects of the vehicle.

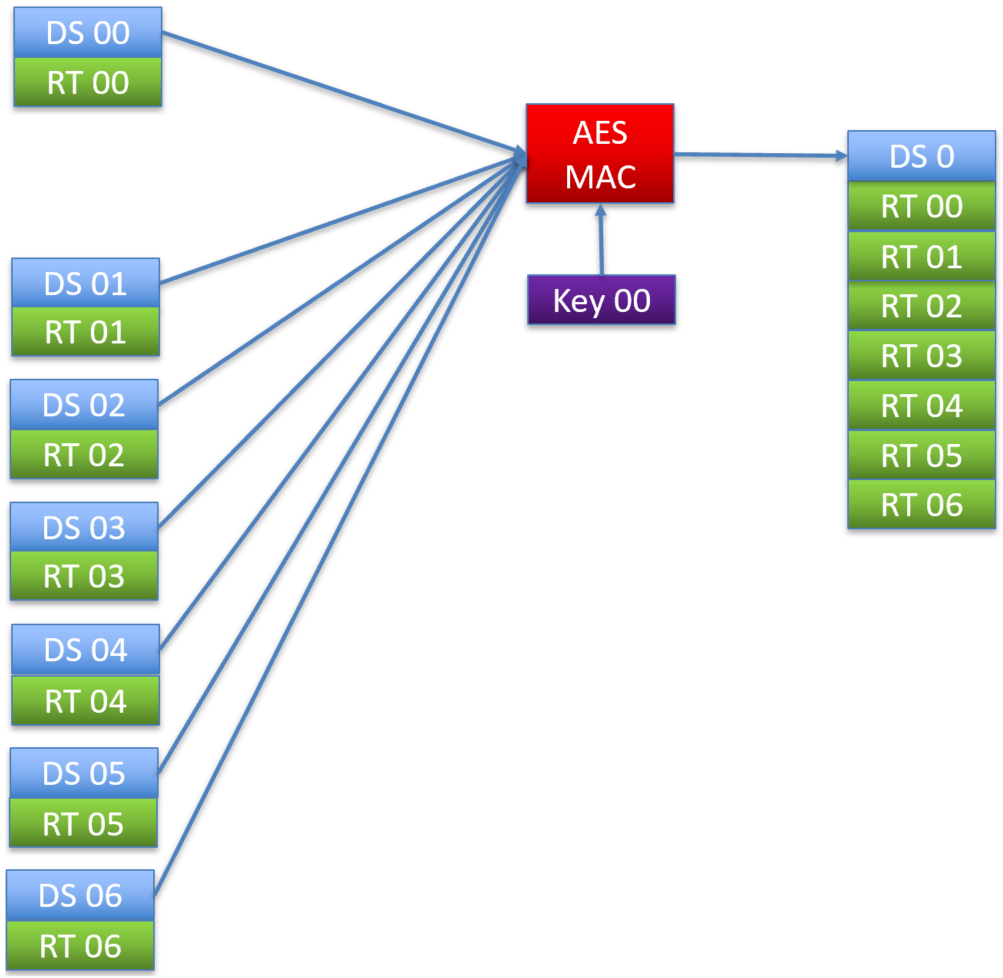


Figure 4.10: How to Build a Digital Signature with Run Time

4.5 Performance Analysis

4.5.1 Memory Requirement

From the regulator's perspective, there are three aspects of memory usage to evaluate: first, the database to keep track of all the OEMs and vehicles, second the data exchanged with the OEM in monitoring the vehicles, and third the data exchanged with the vehicles. We would like to keep the required data storage size to be small enough to fit on a single hard drive, or assume less than a terabyte. Table 4.1 represents an rough estimate of the amount of memory required to support the 284.4 million vehicles in the United States [174] as a sample use case. We assume that the regulator needs to store its keys, and the public keys and IP addresses for each OEM, and a record of the VINs and a date for when the next check is due. With these assumptions, we expect the regulator to store 6.4 GB, which fits easily within our target size.

Data Element	**Size = 6.4 GB**
Private, public keys (RSA 4k)	10 kB
Public keys for each OEM	4 kB * 25 OEMs = 100 kB
IP address for each OEM	8 B * 25 OEMs = 200 B
VIN and date of last check	(17 B + 7 B) * (284.4 *10^6) = 6.4 GB

Table 4.1: Regulator Database

Table 4.2 summarizes the daily data exchange among the regulator and the OEMs with biannual vehicle checks. With a dedicated request per vehicle, we see 3.0 GB / day. If the regulator bundles the requests together, the load reduces to 120 MB / day.

The regulator needs to exchange the seed and the digital shadow with each individual vehicle, this does not lend itself to bundling, as each ECU needs to be addressed individually. Table 4.3 shows that the regulator needs to communicate 1.5 GB / day with the vehicle fleet.

We assume that the OEMs are already maintaining the digital twins for other uses, a record of all VINs they have manufactured, and storing a few cryptographic keys will not present a significant load for them. The vehicle only needs to store the regulator's public key. The vehicle already knows its VIN.

Data Element	**Batch 120 MB**	**Single 3.0 GB**
NONCE	16 B	512 B
VIN	17 B	512 B
IP Address (IP v6)	16 B	512 B
SEED	16 B	512 B
Digital Shadow	16 B	512 B
Vehicles per Day (284.4 10^6 * 2 / 365.24)	1.6 MB	

Table 4.2: Regulator to OEM Daily Communication

Data Element	**Total = 1.5 GB / day**
SEED (16 bytes)	512 B
Digital Shadow (16 bytes)	512 B
Vehicles per Day (284.4 * 10^6 * 2 / 365.24)	1,510,921

Table 4.3: Regulator to Vehicle Daily Communication

4.5.2 Time Requirement

For automotive uCs, we see the MAC calculation taking less time than the hash function. We have focused our measurements accordingly on the time required to calculate a 128-bit CMAC on the entire memory space of the ECU. We either measure this time with an ECU running production SW and executing as a low-priority background run time measurement, or we assume that the uC can dedicate 10% of its run-time to the measurement without impacting operation.

We see in Table 4.4 that the length of time to complete the check is linearly dependent on the memory in the ECU. We also see that different controllers process at radically different speeds. In general, we find that some controllers will be complete in a few minutes, but the larger controllers such as vehicle computers, zone computers, and infotainment systems can take around half an hour with our partial loading assumptions. For systems with off-chip memory, the bottleneck is often the memory used rather than the microprocessor.

Table 4.4: Run-Time Measurements

Manufacturer	Microcontroller	ECU Type	Memory (MB)	Time (seconds)
Infineon	TC38	Engine Controller	8	11
		Engine Controller	12	16
		Engine Controller	16	22
NXP	S32K3x	BMS Controller	4	120
		I/O Aggregator	8	230
	LX2160A	Vehicle Computer	24,576	1,500
Renesas	RH850 ICU-M	Airbag	2	83
		ABS	4	165
		ESP	8	330
	R-Car SOC V3H	Vision	2,048	134
		Vehicle Computer	32,768	2,138
ST	Chorus 4M	Gateway	2	67
		Radar Controller	4	134
Qualcomm	Snapdragon	Infotainment System	32,768	3,277

After each ECU calculates its MAC, the process to build the digital shadow proceeds very quickly. If we assume that each ECU computes a 16 byte MAC and each branch has 8 children, creating the node MAC will need $8 + 16 \times 8 = 136$ bytes. The MAC is built in 16-byte segments, so this will take 9 cycles. The ST Chorus HSM takes 35 microseconds to compute one cycle, so 0.315 ms to build a branch MAC. Sending the signature to it's parent on the CAN bus will strongly depend on bus loading, but 10 ms is a good starting point. Even if we assume 10 layers deep, this will only take about a second to build the digital shadow. The amount of time required to build the DS will be lost in the noise, so we assume that the total time to generate the digital shadow will be roughly the time it takes for the slowest ECU to compute its MAC. Running the calculations in parallel allow a much faster generation than individually querying each ECU.

To accelerate the process, the OEM can begin calculating the DS after the ADR_{VIN} request is received and send them in a batch to the regulator.

4.6 Discussion

Remotely verifying the integrity of an entire vehicle, without access to its target SW is a challenging task. Digital Shadows enable this with a surprisingly lean imposition on industry. A few noteworthy considerations stand out.

- **Dummy ECUs:** It is possible that the attacker leaves the original ECU connected to the communication bus, but inserts a second compromised ECU to control the vehicle. A comprehensive memory check cannot detect this, but the digital twin can be configured to also monitor run time, behavior, and any faults detected.
- **Compliance:** DS can also be used to support compliance testing - demonstrating that the vehicle is operating as intended and has not been tuned or modified with emissions defeat devices.
- **Diagnosis:** DS can also be used to support diagnosis and identification of the modified module by walking the tree and having each parent node submit its signature.
- **HW Plausibility:** Adding monitoring of fault codes and run time needs to be done without being hashed with the memory. It is anticipated that the DT will have a slightly different run time than the actual vehicle. Fault codes may not be accurately tracked in the DT as these represent non-ideal behavior.

4.7 Conclusions

As the industry explores digital twins, digital shadows provide a scalable option for third parties to verify the integrity of automotive SW. With reasonable trust from the users and network, data storage, data exchanged, and time requirements, the scheme provides a framework that provide a trusted authentication of the current vehicle SW. Merging Merkle's tree data structure and vehicle network architecture builds a cryptographically robust fingerprint of the vehicle's current SW without leaking sensitive information about the vehicle.

CHAPTER 5

Zero Trust for SDV - Peer Attestation

5.1 Introduction

Automobiles need a new approach to peer attestation due to new dynamic architectural structures and the probabilistic nature of security intelligence. Autonomous vehicle development is moving at an unprecedented pace, and the vehicles are facing fundamentally new cybersecurity challenges. Today's vehicle can be characterized as a collection of highly-specified ECUs, uniquely entrusted with all associated functionality. For example, the Anti-lock Braking System (ABS) ECU controls the braking system. All functions linked to braking the vehicle find their home in this ECU. The problem of trust within this architecture is solved by sharing symmetric keys to authenticate messages between critical modules [198]. Signed messages provide confidence that these messages are coming from the correct ECU and not an exploiter. The trust anchor can revoke access to keys if it determines an ECU has been compromised, but does not typically interact with the safety system.

These keys are injected during vehicle assembly [36] [175] and require a protected environment if any updates are needed. Keys and operating SW seldom change, and only with great difficulty. The receiver poses the question "Is this message really coming from the ECU that I've been trusting for my entire life?" If the signature matches, then the message is completely trusted. If the signature does not match, then the message is ignored - a response with no gradation. **This trust model is static, binary, and rigid.**

A SDV takes a dramatically different approach, relying on powerful general-purpose computers to control vehicle operation and expecting frequent updates [32] [112] [140]. Autonomous vehicles may use dynamic service oriented protocols such as SOME/IP [26] that allow SW modules to request services and negotiate information sources in real-time. Outside the vehicle, a manufacturer can set up a VSOC team to actively monitor fleets for anomalous behavior, but it needs a means to quickly feed probabilistic intelligence into safety-critical systems. Inside the vehicle, an IDS can detect anomalous behavior in automotive systems, but high false positive rates and AI-hallucinations make it difficult to allow these systems to interact with critical systems with authoritative actions. To take full advantage of these powerful innovations, **a SDV craves a dynamic, nuanced, and adaptive trust model**.

The National Institute for Standards and Technology (NIST) actively promotes ZT to protect against and limit data breaches, and provides a rich field of reference. Older network security strategies followed the analogy of a castle and moat defense. Elaborate defensive measures attempted to keep invaders out, but did not protect against enemy soldiers inside the city. Assuming that an adversary can never breach defenses leaves a false sense of security and the development of new technologies such as planes, parachutes, missiles, and drones renders them ineffective. Classic network security strategies also attempt to keep intruders out, but attackers have been successfully breaching defenses, moving laterally through systems, and gaining access to sensitive information. By assuming that the network cannot be trusted, threats are always present, forcing users to prove their identity, and using fine-grained access rights, ZT forges a new paradigm for defenders.

We propose **changing the focus of ZT from protecting against exposing sensitive data to protecting against taking unsafe actions.** Within a vehicle, individual ECUs do not prioritize keeping secrets from other ECUs. So many data are generated to keep the vehicle running that it is not possible to share everything on a bandwidth limited network. The overarching security risk within a cyber-physical system is not in leaking information, but in taking actions based on compromised information. IT systems manage abstract data, but moving vehicles need to be concerned about their interaction with the physical world. The initial concept considers the problem of introducing IDS into an automobile and giving it the authority to directly influence operation.

Inside the vehicle, an IDS can detect anomalous behavior in automotive systems, but it is difficult to allow IDS to interact with critical systems with authoritative actions, especially when AI-driven systems suffer from hallucinations. Setting the threshold aggressive enough to protect against sophisticated attackers will also trigger on unexpected, but correct behavior.

5.1.1 Contributions

The noteworthy contributions of this chapter are:

- **Dynamic Trust Model:** ZT4SDV provides a scheme that allows a vehicle to dynamically adapt to changing trust. This will be increasingly important as the industry moves to advanced vehicle architectures.
- **Validation in Development Phase:** ZT4SDV allows the designers to verify the impact of the entire range of trust during the development phase.
- **Quick Updates:** Updating the trust policy does not require writing, testing, validating, then distributing new SW, but just distributing a new parameter.
- **Robustly handles Probabilistic and False Positive Data:** Since system operation can be fully validated with a range of trust, the system can be setup and validated with probabilistic intelligence updates and can robustly handle false positive detection from advanced IDS inputs.

5.1.2 Organization

This chapter is organized according to the following structure. Section 5.2 introduces the dynamic trust scheme with a single trust component, an example is proposed then analyzed. Section 5.5 expands this concept with multiple trust components. Section 5.8 concludes the discussion.

5.2 Initial Scheme - Single Trust Component

This section explains our initial scheme to use a dynamic trust model on a single component in an automotive environment and describes a mathematical approach. We use the trust rating of an IDS to balance the weighting of two sensors estimating a value.

5.2.1 Adapting Zero Trust to Automobiles

It is an impressive achievement that vehicles today can circumnavigate the planet four times and last more than a decade without needing anything more than routine maintenance. Each step from development through validation to manufacturing is challenging and demanding. The automobile industry relies on consistency, controllability, and predictability to achieve low cost, high quality, and reliability. Enormous effort goes into making one vehicle work right, then duplicating it as closely as possible; any change introduces safety and quality risks which need to be evaluated, tested, and validated.

Connecting an IDS to a safety critical ECU suffers from several tactical issues. If anomalous or even malicious behavior is detected, how can the vehicle take authoritative actions that do not reduce vehicle safety? How can a system with a high false positive detection rate be allowed to take authoritative actions? What good is a protection system that is not allowed to take authoritative actions?

Zhang et al. [203] explored the potential of using Machine Learning (ML) to detect malicious CAN messages and propose an efficient scheme for achieving this, but did not address the question of what to do with the thousands of false positive detections that even a good system will see every hour.

IT systems effectively use ZT to protect confidentiality using a twofold question of trust. Can the requestor prove he is who he claims to be - identity? Should the requestor access this information - authorization? But when we consider how to apply ZT to enhance security in automobiles, functional safety usurps priority as the primary concern for cyber-physical systems. For our initial concept, we consider that each potential action has an associated safety risk. Inside a vehicle, multiple sources may supply information and within the confines of the vehicle, the current state and functions of the vehicle are not really a secret needing protection. Within the vehicle, the main cybersecurity question is not **"Do I trust the other ECUs enough to honor their request?", but "Do I trust the other ECUs enough to act based on their claim?"** In contrast to ZT in IT systems that protect *abstract access to data*, for cyber-physical systems, the 'asset' we want to protect is *actions in the physical world*.

Table 5.1 provides an overview of the variables we will use in our analysis.

Table 5.1: Variable Definition

Variable	Usage
$\mathbb{S}$	Complete definition of the current vehicle state
$\mathbb{S}'$	Desired next state of vehicle
$\mathbb{S}^*$	Incorrect next state of vehicle
$x_{\mathbb{S}}$	Value of system parameter x in state of $\mathbb{S}$
$V_{\mathbb{S}}$	Estimate of preceding vehicle used by ECUs
$A_{\mathbb{S}}$	Appropriate action to take in state $\mathbb{S}$
$A_{\mathbb{F}}$	False or inappropriate action to take in state $\mathbb{S}$
R_A	Risk of taking action $A_{\mathbb{S}}$
M_i	Message # i
T_M	Trust value in message M
Rel_M	Reliance of estimate on message M
TR_j	Trust Rating of component j
W_j	Weight of component j
k	Scaling constant
Tol_C	Tolerance of camera sensor
Tol_R	Tolerance of radar sensor
V_C	Camera sensor's estimate speed of preceding vehicle
V_R	Radar sensor's estimated speed of preceding vehicle
η	Effectiveness of scheme

5.2.2 Calculating Risk

Traditional cybersecurity priorities consider risk due to loss of confidentiality, integrity, and availability. SDVs may rely on a business model of unlocking features, customers may want to protect vehicle location history; for this paper, we focus our analysis on the unique additional aspect of CPSs - *safety*. Let $\mathbb{S}$ be the complete definition of the state of the vehicle and $\mathbb{S}'$ be the desired state of the vehicle that we want to reach. Furthermore, let $x_{\mathbb{S}}$ represent an individual system parameter such as vehicle speed, coefficient of friction of the right front tire, or the current rotational angle of the motor in the state of $\mathbb{S}$. Each ECU needs to know the subset of $\mathbb{S}$ to determine what action, denoted as $A_{\mathbb{S}}$, it needs to take to reach $\mathbb{S}'$.

$$\mathbb{S} \xrightarrow{A_S} \mathbb{S}' \tag{5.1}$$

Let R_A be the risk associated with executing action $A_{\mathbb{S}}$. This risk considers both the consequences of failure to properly execute $A_{\mathbb{S}}$ (correctly determining $\mathbb{S}'$ but reaching $\mathbb{S}^*$) and failure to properly determine $A_{\mathbb{S}}$ (using $A_{\mathbb{F}}$ to reach the inappropriate state $\mathbb{S}^*$). For many automotive functions, the risk is solely a function of safety. For this paper, we focus our attention on how we can use ZT principles to protect the safety risk.

$$R_A = f(\text{safety}),\ 0 \leq R_A \leq 1 \begin{cases} \mathbb{S} \xrightarrow{A_{\mathbb{S}}} \mathbb{S}^* \\ \mathbb{S} \xrightarrow{A_{\mathbb{F}}} \mathbb{S}^* \end{cases} \tag{5.2}$$

The risk needs to be calculated for each action. Current methods for qualitative risk assessment in automobiles are Failure Mode and Effect Analysis (FMEA) [192] and Automotive Security Integrity Level (ASIL) [77] [149] [137]. ISO 26262 defines risk:

$$Risk_{ISO26262} = Probability \times Severity$$

FMEA ranks risk:

$$Risk_{FMEA} = Severity \times Occurrence \times Detection$$

And ASIL uses a look up table to define risk:

$$Risk_{ASIL} = f(Severity, Exposure, Controllability)$$

The severity ratings from the FMEA or ASIL safety impact can provide a value for R_A. For example, braking the vehicle with an Automatic Emergency Braking maneuver is an ASIL-D function. Properly executed, it can save the life of a child crossing the street, but improper actuation could brake the vehicle when driving on the expressway. ASIL-D functions typically require advanced plausibility detection and redundancy to support safety requirements. Displaying the vehicle speed on the instrument cluster is an ASIL-B function; it is important to know how fast the vehicle is driving, but a significant error is easily detected.

5.2.3 Calculating Trust for each Message

ECUs collect data from sensors and devices in the vehicle to ascertain the true vehicle state $\mathbb{S}$. Based on the estimated state $\mathbb{S}$, the ECUs then determine what actions would be appropriate to control the vehicle. An IDS evaluates the behavior of a device searching for anomalous or suspicious behavior. Often IDSs provide a 'yes' or 'no' output. We want to measure trust as a spectrum. The trust that the IDS has in the source ECU or message should be a graduated estimate of the trust in the message M it sends as T_M.

$$T_M = f(\text{IDS}),\ 0 \leq T_M \leq 1 \tag{5.3}$$

5.2.4 Calculating Reliance

Many vehicle state parameters have multiple sources that can attest to the vehicle's state and concerns other than security play a role in the reliance of a model on a sensor or input. We use Rel_M to describe this reliance on the message under evaluation.

$$Rel_M = f(Trust,\ Risk),\ 0 \leq Rel_M \leq 1 \tag{5.4}$$

We scale the input from each available source *i* by the reliance Rel_{M_i} and if sufficient trust exists to outweigh the risk of misclassifying the vehicle in state $\mathbb{S}$, then we execute action $A_{\mathbb{S}}$.

$$if \sum_i Rel_{M_i} \times T_{M_i} \geq R_A \Rightarrow \text{execute action } A_{\mathbb{S}} \tag{5.5}$$

We also can estimate the value of a parameter $\mathbb{S}_x$ in the vehicle model. The ECUs will use $\mathbb{S}_x$ to determine how it should activate controls in the vehicle. $x_{\mathbb{S}}$ is calculated by combining the available messages attesting to the value of x scaled by their reliance. We include a normalization constant k when combining multiple inputs.

$$x_{\mathbb{S}} = k \times \sum_i Rel_{M_i} \times x_{M_i} \tag{5.6}$$

5.3 Concept Example - Single Trust Component

In this section, we evaluate ZT4SDV using a concrete example with a single trust component.

5.3.1 Example Description

The proof of concept example for integrating a dynamic trust concept consists of two sensors measuring the speed of the vehicle in front of the car, and an IDS that monitors them. The system generates estimates of the speed for two different safety risk settings. The radar and camera sensors have different tolerances, when the system is running normally, the estimate should be more heavily based on the accurate sensor. We simulate a false positive event and an attack on the more accurate sensor and examine the behavior of the estimated speeds.

The radar sensor provides an accurate value with a tolerance of +/- 1 km/h, so $Tol_R = 1.0$. The optical camera provides a less accurate value with a tolerance of +/-5 km/h, so $Tol_C = 5.0$. V_R and

V_C are velocities indicated in the messages received from the radar and camera.

5.3.2 Example Risk

Different consumers use this information within the vehicle and the functions they support have different safety levels. The moderate safety integrity level ASIL-B has a minimum risk level of 0.6. The maximum safety integrity level ASIL-D has a minimum risk rating of 0.9. With different trust thresholds, the two velocity estimates behave differently as trust in the sensors varies.

5.3.3 Example Trust

The IDS examines the messages and provides a trust rating for each of the sensors. The subject of the investigation is the trust in the radar, T_R, which varies throughout the simulation. T_C, the trust in the camera remains at 1.0 for the entire simulation.

Zhang et al.'s ML-based IDS outputs a binary indication if the system is more likely compromised or not [203]. This investigation needs an analog graduated response. One possible strategy takes the outputs from the final hidden layer, sums them, then normalizes to give an output ranging from 0.0 to 1.0, indicating the trust level that the IDS has in each sensor.

5.3.4 Reliance Transfer Function

An ideal dynamic trust model weighs multiple information sources and adapts the dependence during operation as the trust landscape changes. This is difficult for a rigid trust model, but our model allows us to adjust the reliance on different inputs based on the output of an IDS. We also consider that input sources may have different tolerance and accuracy. Our initial concept weights tolerance with a linear function and trust with an exponential. The functions should be continuous to gradually redistribute reliance as trust changes. The signal sources are weighted relative to their accuracy as long as the sources have sufficient trust. When the source is questionable, the reliance reduces dramatically.

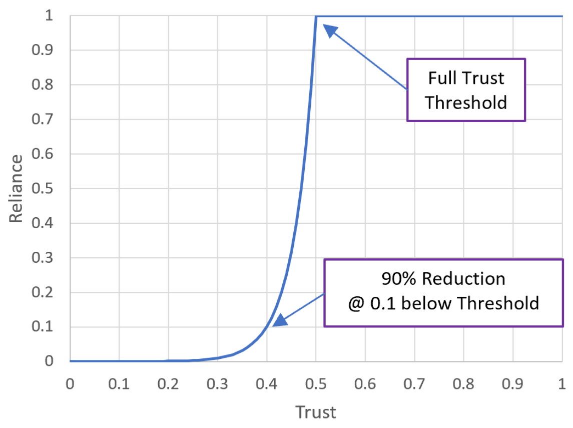


Figure 5.1: Ideal Trust Transfer Function

Figure 5.1 shows the proposed trust weighting function with the trust threshold risk, $R_A = 0.5$. Above this level, we have full reliance in the input. When the trust decreases to 0.1 below the risk threshold, the reliance drops to 0.1. Equation 5.7 shows the formula to enable an exponential transfer in reliance as the trust falls off. The value of 23 was heuristically selected to provide this desired characteristic.

$$Rel = \begin{cases} e^{23(T_M - R_A)} & , 0 \leq T_M < R_A \\ 1 & , R_A \leq T_M \leq 1 \end{cases} \tag{5.7}$$

5.3.5 System Parameter Calculation

The classic approach to fusing the sensor data would combine the two sources based on their accuracy or tolerance. For our discussion, we will calculate the speed of the vehicle based on tolerance and trust. Equation 5.7 calculates the reliance on the radar Rel_R and camera Rel_C.

When $T_M \geq R_A$, we want to weight the estimation of the system parameter $\mathbb{S}_v$ inversely proportional to the tolerance of the input. The normalization factor k is calculated in (5.8).

$$k = \frac{Tol_C \times Tol_R}{Tol_R \times Rel_R + Tol_C \times Rel_C} \tag{5.8}$$

$V_{\mathbb{S}}$, the system estimate of the preceding vehicle velocity is built in (5.9).

$$V_{\mathbb{S}} = k \times \left(\frac{Rel_R \times V_R}{Tol_R} + \frac{Rel_C \times V_C}{Tol_C} \right) \tag{5.9}$$

5.4 Performance Analysis - Single Trust Component

This section evaluates the performance of the single component dynamic trust algorithm.

5.4.1 Adversary Objectives

To evaluate the performance of this scheme, we do not address the general problem of IDS detection effectiveness but focus on the impact on the physical system. The adversary has two objectives - to compromise the system in a way that allows a substantial impact to the vehicle behavior and avoids detection for a significant length of time.

Our system has the following principal objectives that lend themselves to measurement.

- While allowing the IDS to affect system performance, the impact of false positive detections should be within allowable limits.
- Testing for the variable IDS influence should be safely, completely, and efficiently validated in the development phase.
- A compromised system should be quickly detected.
- A small change in the transmitted values should be quickly detected.

5.4.2 False Detection Robustness

The vehicle speed starts at 80 km/h and then varies by up to 1 km/h every 100 ms. The radar and camera read the vehicle speed and each adds a random error based on its tolerance. At 5 seconds, the IDS erroneously detects suspicious behavior on the radar, reducing T_R down to 0.78 and back to 1.0 at 6 seconds. Figure 5.2 does not show a change in the calculated ASIL-B or ASIL-D speed.

Figure 5.4 however shows that the error for the ASIL-D signal increases above the error for the ASIL-B signal. Figure 5.3 shows that between 5 and 6 seconds, the ASIL-D value transfers its reliance from the radar to the vision camera based on the input from the IDS, but the ASIL-B signal, with a lower risk rating and requisite trust, does not. If the accuracy of the camera sensor is sufficient to allow safe operation, then the system operates properly even when IDS misdetection occurs.

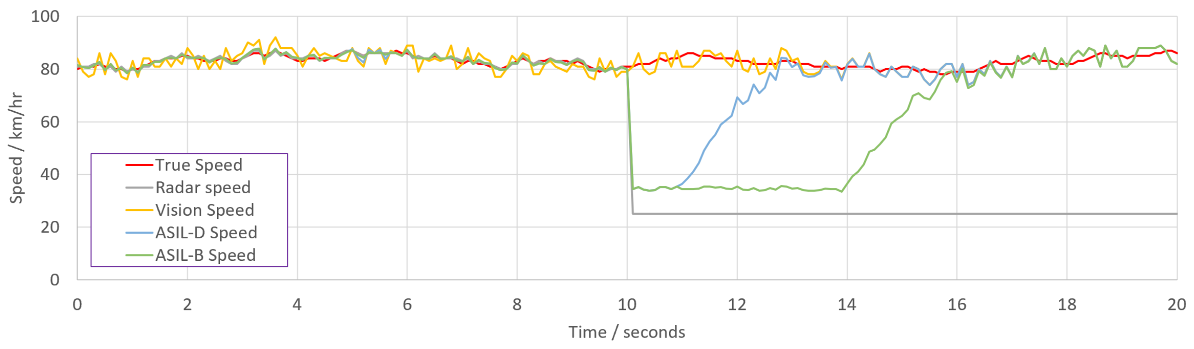


Figure 5.2: Vehicle Speed Estimates

5.4.3 Attack Detection Reliance Tradeoff

At 10 seconds, the attacker changes the radar speed to 25 km/h. Figure 5.3 shows the output of the IDS trust in the radar starts at 100% when the attack happens and drops off 1% each 100 ms. This is not how a real IDS would behave, but it provides a good illustration of how the reliance transfers as the trust drops off. Until 11 seconds, $T_R \geq R_{ASIL-D}$ so there's no change in the reliance on the sensors.

At 11 seconds, the IDS degrades the trust in the radar to the threshold $T_R = R_{ASIL-D}$, Figure 5.3 shows that the reliance of the $V_{\mathbb{S}-ASIL-D}$ on V_R drops off exponentially and begins relying on V_C instead. The attacker successfully modifies $V_{\mathbb{S}-ASIL-D}$ until the IDS reduces the trust at $T_R = 0.9$.

At 14 seconds, the IDS degrades the trust in the radar to $T_R = R_{ASIL-B}$. Figure 5.3 shows that the reliance of the ASIL-B estimate then begins to drop as well.

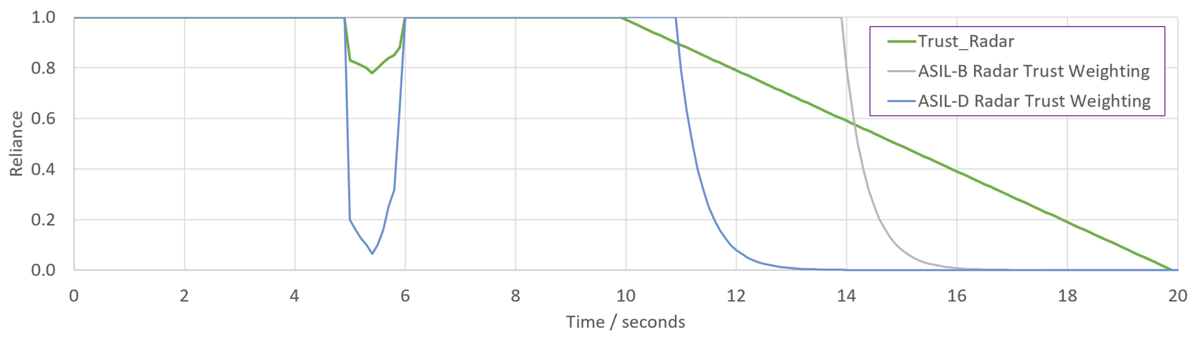


Figure 5.3: Reliance Dependence on Trust

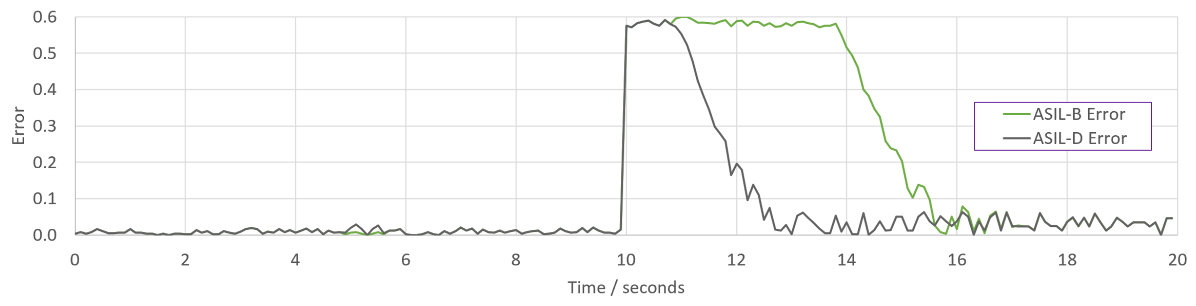


Figure 5.4: Error in Vehicle Speed Estimates

5.4.4 Performance Analysis

Figure 5.5 illustrates our first proposal to measure the impact of the attack by integrating the difference between the value that the algorithm estimates for this parameter during the compromise and the value that the algorithm would estimate based on the uncompromised sensor data. We then integrate the difference over the time that the attack is active to calculate the effectiveness η. The quicker the algorithm detects a smaller error, the better the performance.

$$\eta = \int_0^{t_{DET}} |V_{S-ASIL-D} - V_C| \tag{5.10}$$

5.4.5 Validation

Abstracting the IDS input and characterizing it as a risk-based measurement allows validation to occur during the development process. It requires in-vehicle testing for several critical values of

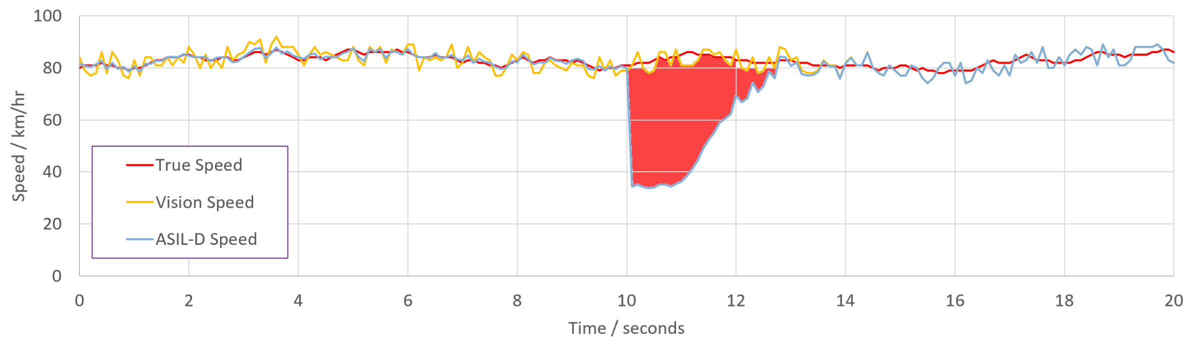


Figure 5.5: Estimating Total Impact

T_R and T_C, supplemented by simulations sweeping their entire ranges. This process can be safely integrated into the vehicle allowing robust validation and direct interaction of the IDS in critical operation. This supports a gradual shifting of reliance rather than an abrupt change in the system operation.

5.5 Complex Scheme - Multiple Trust Components

This section expands the dynamic trust concept to consider multiple components of trust.

5.5.1 Calculating Risk

We now want to expand the concept to evaluate multiple components of trust or factors that contribute to how much the ECU should trust the incoming message. It will use the same strategy for evaluating $\mathbb{S}$, $x_{\mathbb{S}}$, and R_A as previously discussed.

5.5.2 Calculating Trust for each Message

The ECU collects data from sensors and devices to ascertain the true vehicle state $\mathbb{S}$. We now want to consider measuring trust by evaluating multiple components. Did the message travel a short distance, e.g., from another ECU on a dedicated CAN bus? Is there a long history of communicating with that same ECU or is this a new communication partner? Is there robust attestation of the message's authenticity? Has the VSOC downgraded the trust in the source ECU based on intelligence gathered from monitoring vehicles in the field? Has the IDS downgraded the trust in the source ECU or message based on internal plausibility? We refer to the trust in message M as T_M :

$$T_M = f(\text{distance, history, authentication, VSOC, IDS}) \tag{5.11}$$

We calculate the trust in Equation 5.11 by evaluating two factors for each component. Selecting the j^{th} element from the set of components C, let W_j be the weight of component C_j relative to the other components. Let TR_j be the Trust Rating for component j ranging from 0 to 1. We compute the trust for each message M by Equation 5.12.

$$T_M = \frac{\sum_j W_j * TR_j}{\sum_j W_j} \tag{5.12}$$

5.5.3 Reliance Calculus

For the expanded case with multiple trust components, we calculate R_A, the risk associated with taking action A_S which would be appropriate if the vehicle is in state $\mathbb{S}$. We use the FMEA or ASIL ratings to provide a numerical value for the risk as shown in (5.2).

We calculate the trust in the messages informing the ECU that the vehicle is in state $\mathbb{S}$ using (5.12).

Many vehicle state parameters have multiple sources that can attest to the vehicle's state and concerns other than security play a role in the reliance of a model on a sensor or input. The reliance on message M, Rel_M is a function of trust and risk as in (5.4).

With multiple sources providing messages with information attesting to the vehicle state, we scale the input from each available source i by the reliance Rel_{M_i} and if sufficient trust exists to outweigh the risk of misclassifying the vehicle in state $\mathbb{S}$, then execute action $A_{\mathbb{S}}$ as in (5.5).

The value of an individual system parameter $V_{\mathbb{S}}$ can also be estimated by combining the available messages attesting to the value of $V_{\mathbb{S}}$ scaled by their reliance on the trust and accuracy of the source as in (5.6)

5.5.4 Potential Use Cases

An ideal dynamic trust model weighs multiple information sources and adjusts the dependence in the field. The following objectives and use cases are difficult for a rigid trust model, but well suited to a dynamic one.

- **Validation during Development Process:** By varying the trust rating during the development process, engineers can evaluate the impact on system performance and validate resulting behavior. After development is complete, revalidating vehicle operation becomes onerous, expensive, and lengthy. During the validation process, the concept validates that the system operates properly while varying TR_j for each input source.
- **Scalability:** Since vehicles are inherently mobile, they may receive their updates over slow, unreliable connections. It is much easier to update a trust policy parameter than to deliver a complete SW package as required by today's model.
- **Time to Deploy:** Performing the validation during the development cycle and supporting lean updates shortens the time to develop, test, and rollout a fix from months to minutes.
- **Maintainability:** Maintaining a comprehensive test fleet to validate SW updates is complicated, expensive, and time intensive, especially as the target population ages.
- **VSOC:** A manufacturer's VSOC with a rigid trust structure cannot easily implement updates. Reworking the SW may require in-vehicle testing. However a fully validated system that accepts a full range of trust ratings allows the VSOC to make fine-grained changes reflecting current threat intelligence. The VSOC can make authoritative changes that directly influence vehicle control, as the parameters have already been validated. The VSOC can also quickly

roll out these changes as a change to the trust policy is just a few bytes rather than megabytes for a new SW package.

- **IDS:** Today's vehicles are reluctant to place too much authority on IDS warnings, as safety must be maintained and false positive detection rates limit the usability of the systems. A fine-grained trust rating enables an IDS to engage by gradually influencing vehicle operation.

- **Right to Repair:** Industry, advocates, and lawmakers are struggling with balancing right to repair concerns with security risk. The flexibility from the dynamic trust strategy allows fine-grained and flexible control of service and diagnostics, perhaps opening up certain vehicles to more third-party service options or allowing classic cars to be enjoyed after security updates are discontinued.

- **Moving Target Defense (MTD):** A MTD strategy increases variation in system configuration to make it difficult for adversaries to develop a successful, long-term, scalable attack. For SDVs, MTD could have each vehicle develop a custom reliance pattern or a time-limited reliance pattern; adaptability and diversification serve to both harden individual vehicles and increase detection of suspicious behavior in the fleet. The worst-case cybersecurity scenario is where attackers develop an attack on a single vehicle, then deploy this to the entire fleet of identical vehicles with identical systems and identical security measures. A MTD can reduce the scalability risk of successful attacks either by limiting the similarity between vehicles or the length of time that a specific configuration is active.

- **Health Check:** When a vehicle is isolated from the network, either with malicious or benign intent, it is not aware of available security updates from the VSOC or safety updates from the OEM. The dynamic model could include a trust component for health whose value decreases at regular intervals and gets refreshed by checking for updates. ZT4SDV enables a graceful degradation of functionality until the system can check for updates. Functions and sensors could degrade at different rates depending on their susceptibility to malicious attack and sensitivity to time-critical updates. These rates could also be adjusted by the VSOC to provide granular control.

5.6 Concept Example - Multiple Trust Components

This section demonstrates the proof of concept example for integrating a dynamic trust concept retains the two radar and camera sensors that send regular messages with estimates of the speed of the vehicle in front of the car.

5.6.1 Trust and Risk Calculation

We start each component's trust rating at 1.0, assuming the sensors have already built sufficient trust. The single exception is the distance component, the hypothetical vehicle has other ECUs on the same bus and anticipates potential malicious messages. We rank W_j for each component according to a plausible relative rating for the importance of the different components.

Table 5.2: Initial Component Trust Rating Values

C_j	W_j	Initial TR_j
IDS	5	1.0
VSOC	5	1.0
Distance	4	0.4
Authentication	3	1.0
History	1	1.0

Based on Equation 5.12, the calculated trust for the camera messages, T_{camera} is 0.87 for the entire event. The calculated trust for the radar, T_{radar} also starts at 0.87 but changes over the course of the simulation.

We assume that there are multiple functions using this information with different safety ratings. We assigned the minimum trust level of 0.6 for an ASIL-B function and 0.8 for one rated ASIL-D.

5.6.2 Reliance Calculation

We retain the tolerances and variable names from (5.7), (5.8), and (5.9). Again, we calculate an estimate of the vehicle speed based on the risk, trust, and tolerance. The ASIL-D value has a higher risk threshold and will be more sensitive to reductions from any of the components that we are using to calculate trust. As the components contribute to trust, they will also impact the reliance of the system parameter on the different sources.

5.6.3 Example Operation

Figure 5.6 illustrates the concept in action. The true speed of the preceding vehicle is blue, starting at 80 km/h randomly changing by up to 1 km/h every 100 ms. The radar sensor estimate is in red, the optical camera's estimate is the brown trace. At time 8 seconds, the trust engine receives input from the VSOC that there other vehicles have reported suspicious activity on the radar sensor and downgrades the trust to $TR_{VSOC} = 0.8$. At 10 seconds, the adversary launches the attack and changes the radar speed to 25 km/h. Based on the current trust settings, the ASIL-B and ASIL-D estimates drop to 33.3 km/h.

Figure 5.7 illustrates that the onboard IDS quickly detects the suspicious behavior and begins gradually downgrading the trust in the radar sensor TR_{IDS} in increments of 0.2. As the trust engine decreases the trust, the system gradually transfers its reliance on the vehicle speed from the more accurate but compromised radar to the uncompromised camera. Since the vehicle-level impact of the ASIL-D action is higher than the ASIL-B, it is more sensitive to a loss in trust. The ASIL-D value becomes totally dependent on the camera by 14 seconds, whereas the ASIL-B value is just starting to respond. At 18 seconds, the VSOC provides another update to the vehicle to further downgrade the trust in the radar sensor with $TR_{VSOC} = 0.6$ and the final value of $T_{radar} = 0.48$.

5.7 Performance Analysis - Multiple Trust Components

This section evaluates performance of the multiple component ZT4SDV concept.

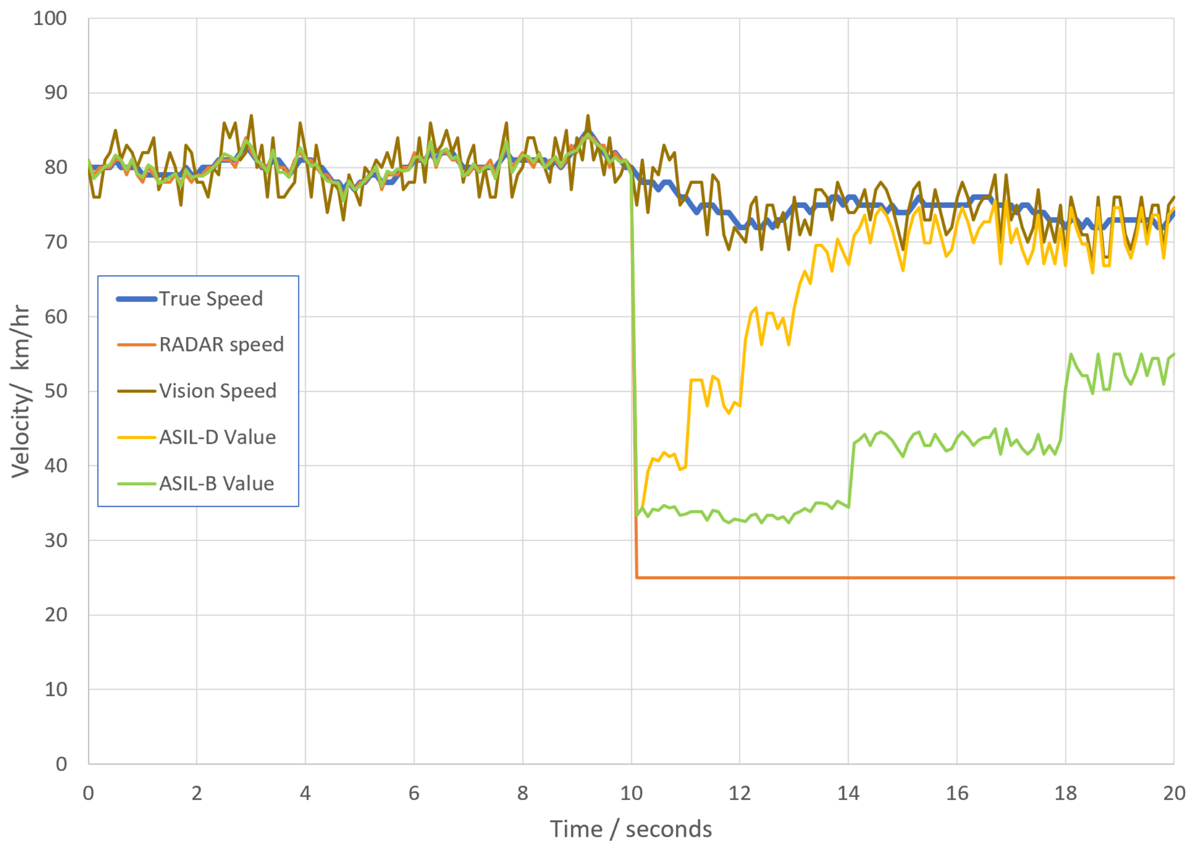


Figure 5.6: Dynamic Evaluation of Vehicle Speed based on Trust

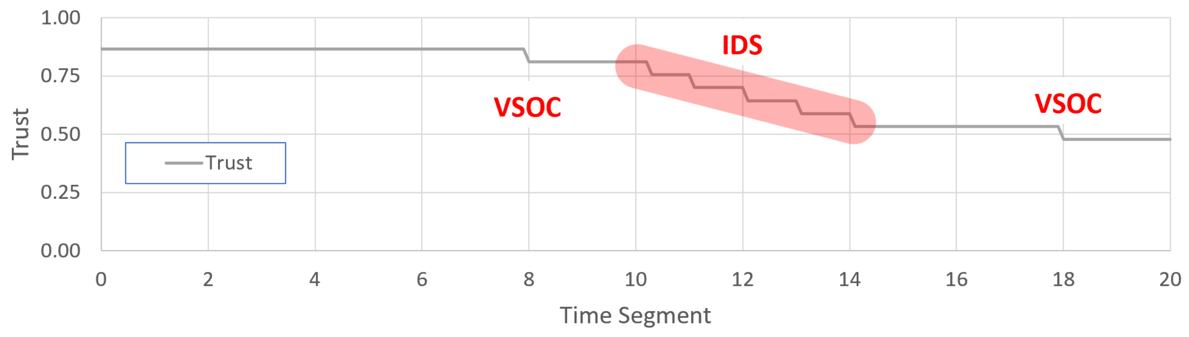


Figure 5.7: IDS and VSOC Impact on Trust

5.7.1 Adversary Objectives

We assume the adversaries attacking a SDV want to maintain a higher level of trust than is warranted. They want to use this trust to achieve two things:

1. Modify the vehicle behavior by a significant amount.

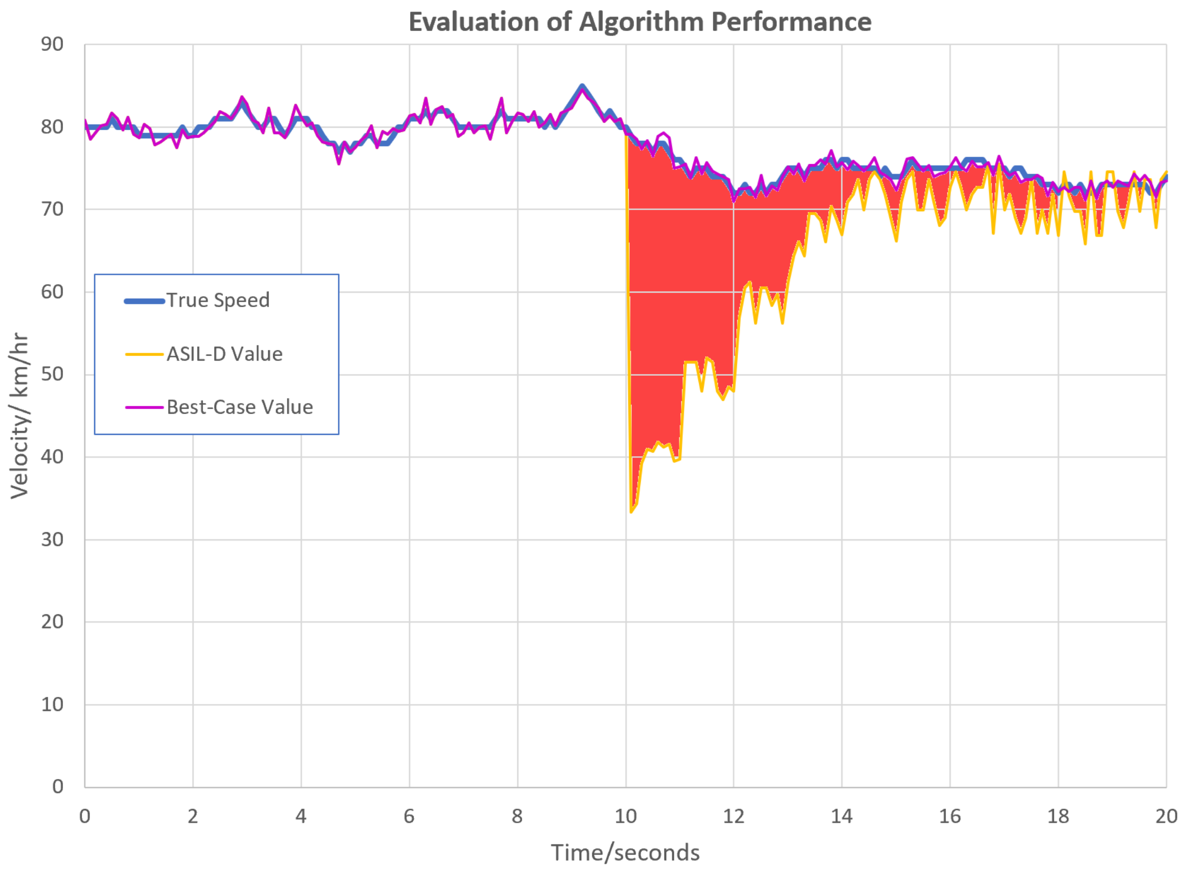


Figure 5.8: Algorithm Efficiency

2. Avoid detection for a long time.

The performance of this concept is demonstrated by evaluating the ASIL-D calculated speed, $V_{\mathbb{S}-ASIL-D}$. We measure the difference between the value that the algorithm estimates for this parameter during the compromise and the value that the algorithm would estimate if it had access to an omniscient oracle with access to these same data sources without any compromise, $V_{\mathbb{S}-ASIL-D}*$. We then integrate the difference over the time that the attack is detected to calculate the effectiveness η. The quicker the algorithm detects a smaller error, the better the performance.

$$\eta = \int_0^{t_{DET}} |V_{\mathbb{S}-ASIL-D} - V_{\mathbb{S}-ASIL-D} * | \tag{5.13}$$

For this example, Figure 5.8 illustrates the proposal to measure the total error, namely add up area in the red shaded region of this difference. When the attack starts at 10 seconds, the initial error is significant, the error is 45.8 km/h. As the IDS detects the compromise and the trust degrades, the error drops. At 12.1 seconds, the error is only 15.6 km/h, and at 14.1 seconds, the error is 4.0 km/h, which is within the tolerance of the camera. The integral of this error over the time of the attack totals to 103.7 km/h-seconds. The small continuing contribution to the error after conclusive detection is due to the reliance on the less accurate sensor providing an error greater than if the compromise had not occurred.

This evaluation scheme provides a method of rating different IDS implementations and parameter settings while interacting with a vehicle.

5.7.2 VSOC Operation

In this example, the VSOC was able to directly interact with the safety systems in real time by allowing a nuanced input to reflect the fuzzy nature of security intelligence. A full SW update requires significant effort to develop the new models, test them in a virtual environment, test them in physical vehicles, and then roll it out to the fleet. Even if the vehicles support FOTA updates, the process of sending new updates often requires a vehicle to be parked near a trusted internet connection for a significant period of time. Sending a small update to the trust engine of a parameter is far easier. It also allows system developers to validate adjusting a parameter during the pre-production validation process.

5.8 Discussion

This paper is an illustrative demonstration of a strategy to enable probabilistic security measures to interact with a SDV. The main limitation of this work is its conceptual nature - it has not been developed in sufficient detail to test in a vehicle or evaluate with explicit functions. We are working to submit a proposal to CCAT to continue investigating the following steps of ZT for SDV:

- Build a trust concept compatible with functional safety.

- Investigate how to build a signal from different sources with different trust and reliance levels.
- Investigate if it is compatible with functional safety requirements to adapt reliance weightings on a functioning vehicle.
- Evaluate if validating the trust parameters in the development phase is feasible and efficient.
- Develop a concept for separating the control plane from the data plane for a SDV. Should the trust engine reside in a central computer evaluating the values from different sources or in the actuator?
- Evaluate the system from the attacker's perspective.
- Develop a proof of concept MTD automotive model. Is there sufficient breadth of signal reliance to build sufficient uniqueness to provide an advantage to defenders? Does a time limited reliance pattern provide an advantage to defenders? Is there another strategy that could be used to implement MTD in a vehicle environment?
- Include time of last update as a trust component with a timer measuring how long it has been since the last update.
- Finally, evaluate the concept with real-world data.

5.9 Conclusions

Adapting the ZT paradigm to a cyber-physical automobile requires refocusing the key objective to link trust and safety impact. Actions cannot be taken unless the actuator has sufficient trust in the signals. The dynamic, nuanced, and adaptive concept enables SDVs to grant security mechanisms direct authority in operation and verify the behavior during the development phase.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

The underlying concern tying these problems together is that tomorrow's automobiles will be characterized by increased connectivity and increased authority of the electronics. Adopting proven cybersecurity techniques to address these new risks is complicated by the real-time, safety-critical, CPS, low power microcontrollers driving them. Chapter 3 explains our solutions for how these devices can verify their SW before starting. Chapter 4 details our solution for how government regulators and other interested third parties can remotely verify the SW while the vehicles are deployed. Chapter 5 explains our strategy to allow software defined vehicles to have SW modules dynamically adjust the trust and reliance on their peers.

Sliced Secure Boot reduces the time required to verify the ECU's SW to make secure boot feasible for automotive ECUs while avoiding interference with the safety concept. Carefully defining the adversary objectives allows the sampled check to provide an acceptable level of security to protect automotive controllers. Each ECU can use a unique key to protect the fingerprints and a unique pattern to generate them. Our formula defining the behavior, the simulations verifying them, and the real-world measurements exhibit significant benefits from using SSB as a run-time detection algorithm, where we build trust incrementally. The critical question facing automotive security engineers is not "Is a sampled secure boot as good as a full one?" but the pragmatic one "Since we must meet our functional safety and start up time requirements, do we implement SSB protecting *this* ignition cycle or a full image background check protecting *the next*?"

Digital shadows provide a scalable option for third parties to verify the integrity of automotive SW using digital twins. With reasonable trust from the users and network, data storage, data exchanged, and time requirements, the scheme provides a framework that provide a trusted authentication of the current vehicle SW without leaking sensitive information about the vehicle. The MAC-based algorithm with a random starting address together with information about the run-time and fault codes demonstrate to a third party that the desired SW is actually in control of the vehicle. Our measurements demonstrate that concept works within reasonable limits with memory and time.

Dynamic Trust for SDVs adopts the ZT paradigm to a cyber-physical automobile by linking trust and safety impact. Actions cannot be taken unless the actuator has sufficient trust in the signals. Our simulation shows that vehicle state variables can be built by including trust as a factor in sensor fusion. The dynamic, nuanced, and adaptive concept enables SDVs to grant probablistic security intelligence mechanisms like IDS and VSOC direct authority in operation and verify the behavior during the development phase.

6.2 Future Work

As is the case for most doctoral research, this paper reflects a significant amount of thinking, testing, arguing, and hopefully a few good ideas. The secure boot project is at a state where it could be implemented and is being considered to protect automotive ECUs. The remote attestation project has met moderate interest from several DT adopters, but it needs broader adoption of digital twins before it would make sense to consider. The dynamic trust concept provides an interesting approach to a problem that is still quite fresh. There is significant interest from government and industry to introduce principles of ZT in the automotive industry and SDVs need an adaptive trust model. I plan to continue researching this topic at work and hope to align with other collaborators.

Appendix A

Modulo 16 Multiplication Table

Table A.1 shows the shuffling patterns with modulo 16 multiplication. The rows represent the offset that is chosen for a block. The columns represent the memory cells that are selected. The mapping proceeds as follows. Assume $Offset = 5$. The first fingerprint samples cell 5, the second samples cell 10, the ninth cell 13.

Modulo multiplication requires careful selection of the base. Using a composite number with common factors causes the patterns to collapse. The even offsets create a degenerate state where the odd cells are not protected by any fingerprints. Cell 8 maps always to either fingerprint 0 or 8, fingerprint 16 (or 0 in Modulo 16) always maps to fingerprint 0. Any predictability in these patterns provides a foothold for the attacker.

The odd offsets are relatively prime to the modulo, so these 8 offsets provide full coverage patterns ensuring that each cell will be protected by a fingerprint. The fact that cell 16 always maps to fingerprint 0 provides a bit of shadowing that a creative adversary may be able to exploit.

Table A.1: Modulo 16 Multiplication

Offset	Memory Cell Column															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2	2	4	6	8	10	12	14	0	2	4	6	8	10	12	14	0
3	3	6	9	12	15	2	5	8	11	14	1	4	7	10	13	0
4	4	8	12	0	4	8	12	0	4	8	12	0	4	8	12	0
5	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11	0
6	6	12	2	8	14	4	10	0	6	12	2	8	14	4	10	0
7	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9	0
8	8	0	8	0	8	0	8	0	8	0	8	0	8	0	8	0
9	9	2	11	4	13	6	15	8	1	10	3	12	5	14	7	0
10	10	4	14	8	2	12	6	0	10	4	14	8	2	12	6	0
11	11	6	1	12	7	2	13	8	3	14	9	4	15	10	5	0
12	12	8	4	0	12	8	4	0	12	8	4	0	12	8	4	0
13	13	10	7	4	1	14	11	8	5	2	15	12	9	6	3	0
14	14	12	10	8	6	4	2	0	14	12	10	8	6	4	2	0
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B

Modulo 17 Multiplication Table

Table B.1 shows the shuffling patterns with modulo 17 multiplication. The degenerate factor 0 was excluded leaving 16 unique columns and combinations. We also see these desirable qualities.

- **Unique scattering patterns:** We see that each offset provides a unique scattering pattern.
- **Full coverage:** We see that each offset supports each cell being protected by a single fingerprint.
- **Full variation:** We see that each offset maps a cell to a different fingerprint.
- **Unbiased distribution:** We see there is no bias in the mapping. E.g., as Table A.1 with cell 8 mapping to fingerprint 8, or cell 2 mapping to fingerprint 2 twice, but never to fingerprint 1.
- **Different jump sizes:** The offsets correspond to how many cells are jumped between fingerprints. Having different jump sizes makes it difficult to conceal contiguous code blocks.

Table B.1: Modulo 17 Multiplication

Offset	Memory Cell Column															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	2	4	6	8	10	12	14	16	1	3	5	7	9	11	13	15
3	3	6	9	12	15	1	4	7	10	13	16	2	5	8	11	14
4	4	8	12	16	3	7	11	15	2	6	10	14	1	5	9	13
5	5	10	15	3	8	13	1	6	11	16	4	9	14	2	7	12
6	6	12	1	7	13	2	8	14	3	9	15	4	10	16	5	11
7	7	14	4	11	1	8	15	5	12	2	9	16	6	13	3	10
8	8	16	7	15	6	14	5	13	4	12	3	11	2	10	1	9
9	9	1	10	2	11	3	12	4	13	5	14	6	15	7	16	8
10	10	3	13	6	16	9	2	12	5	15	8	1	11	4	14	7
11	11	5	16	10	4	15	9	3	14	8	2	13	7	1	12	6
12	12	7	2	14	9	4	16	11	6	1	13	8	3	15	10	5
13	13	9	5	1	14	10	6	2	15	11	7	3	16	12	8	4
14	14	11	8	5	2	16	13	10	7	4	1	15	12	9	6	3
15	15	13	11	9	7	5	3	1	16	14	12	10	8	6	4	2
16	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Appendix C

SSB Theoretical Derivation

I spent a lot of energy trying to develop a closed form equation describing the robustness of the fingerprints and the escape rate probability. Estimating the most likely escape rate with random distribution of the compromised cells was modeled and reflected in Chapter 2. Trying to determine how likely the adversary was to improve his escape rate beyond this proved difficult to thoroughly model. It did provide the advantage of helping to define an attacker strategy, namely where the attacker would attempt to ”shadow” the compromised cells from each block to fall into the same fingerprints. This appendix is included to describe a technique for calculating a more effective attack, but was not included in Chapter 2, as we anticipate skilled attackers would use their intelligence to maximize shadowing. Using ECU-specific unique patterns makes it very difficult to transfer a successful attack pattern from one ECU to the next. Limiting the return on an attacker's investment to a single ECU is an effective protection measure. This appendix is included to address questions about modeling the escape rate for randomly placed modifications in greater detail, but a skilled attacker will not randomly place modifications.

C.1 Bayesian Learning Improving Escape Rate

We do know that the escape rate is bounded. Ideally, we would like each fingerprint to sample only one modified code cell, and the adversary would like v modified cells to be sampled by each fingerprint. We can think of this as shadowing, or the portion of the compromised code segments from different rows that are covered by the multiple fingerprints. We can also consider this as how effective the adversary is at predicting the sampling pattern. If we have no shadowing, or the adversary fails at concealing any compromised code segments to being checked by the same fingerprint, the minimum escape rate will be:

$$Pr(Escape_m) \geq \left(1 - \frac{w \times v}{b}\right)^m \tag{C.1}$$

If the adversary is fully successful at shadowing the compromised segments, the fingerprint patterns will effectively only see a single compromised segment, so the maximum escape rate will be:

$$Pr(Escape_m) \leq \left(1 - \frac{w}{b}\right)^m \tag{C.2}$$

Rather than considering the coverage rate by looking at the individual compromised blocks, we can map the compromised cells onto the fingerprint patterns and see how many fingerprints contain compromised cells. If we consider that the v segments map to an effective number of cells covered in a single block w^*, we calculate the most likely value of the coverage to be:

$$Pr(Escape_m) = \left(1 - \frac{w}{b}\right)^{v \times m} = \left(1 - \frac{w^*}{b}\right)^m \tag{C.3}$$

$$w^* = b\left(1 - \left(1 - \frac{w}{b}\right)^v\right) \tag{C.4}$$

To address this theoretically, I think what we would need is to be able to calculate the amount of shadowing given that we have escaped m events previously. I considered using the combinatorial formula.

$$Number_{issues} = nCv \times w(s)^m(1-s)^{n-m} \tag{C.5}$$

But what we really want, is what is the maximum likelihood value of w_{eff_m}, given that we have successfully evaded m cycles. We know that $w \leq w_{eff} \leq w \times min(v, b)$. When $m = 1, w_{eff} = w^*$.

For the first $\frac{b}{w^*}$ ignition cycles, we actually don't learn anything, since we expect our first detection to occur after $1/\frac{w^*}{b}$ cycles. After we pass twice this amount, then it is pretty likely that

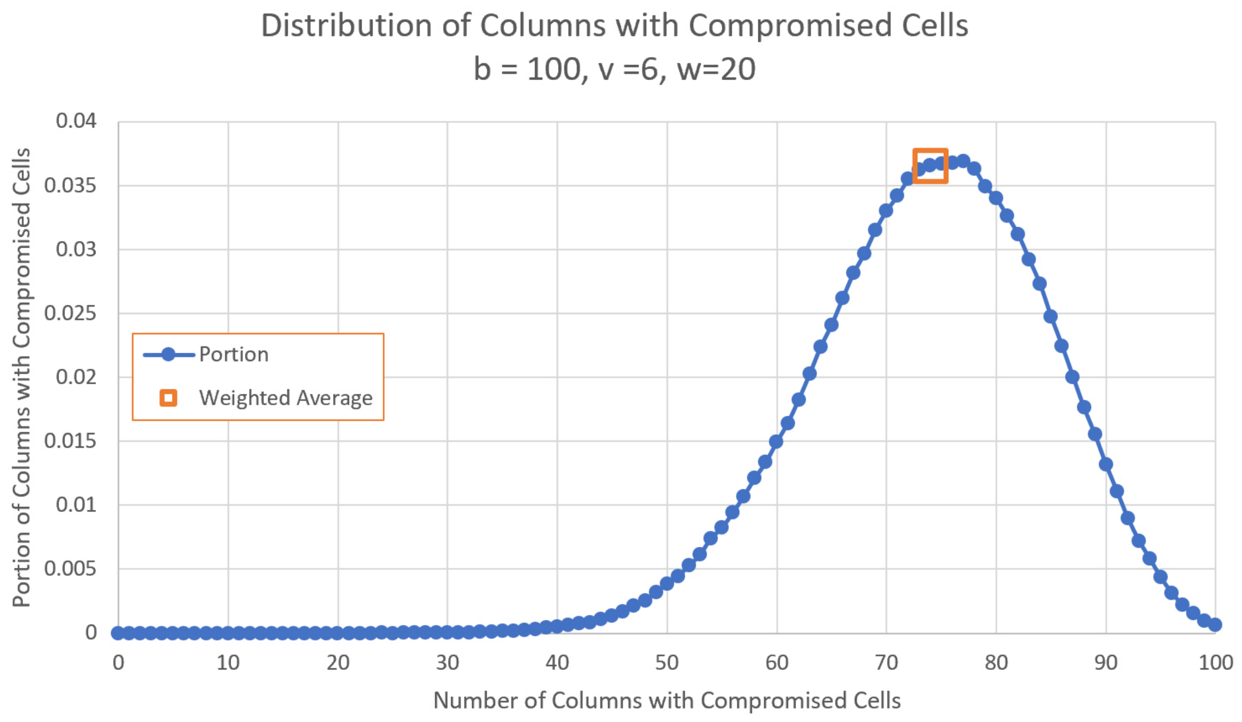


Figure C.1: Simulated Escape Rate for SSB

the adversary has effectively shadowed some of the cells. If $v > \frac{w}{b}$, then the distribution will start to look Gaussian due to the law of large numbers. In Figure C.1, we see the weighted average as w^*. I think if we run a simulation with $w_{eff} \in (-1\sigma, v \times w)$, we would get an escape rate roughly equal to $w_{eff} = w^*$. If we run a simulation with $w_{eff} \in (-2\sigma, -1\sigma)$, we would get an escape rate roughly equal to $w_{eff} = w^* - \sigma$. If we calculate the cumulative distribution function for the value of w_{eff} and use this as a basis for the fall off rate, I think this may be the best way to determine this theoretically. Maybe do a Taylor series expansion for the CDF and then see how it behaves as we decrease w_{eff}.

Consider the case when w/b is small and v is large, so that the estimated value of $w^* \approx b$. After each successive sample, we learn that the shadowing is significant. But, if we have v small, we do not learn much about the shadowing.

If we wanted to define a value of shadowing, I'd say $s = 1$, when $w_{eff} = w$. And $s = 0$, when $w_{eff} = v \times w$. This does not play a role with single-use fingerprints, as the shadowing will change with each fingerprint, and on average $w_{eff} = w^*$.

C.2 Linked Scanning

If we return to the idea of shadowing, another perspective could be to think of the linkage in the fingerprint patterns. If we have a fully linked, worst-case from the detection perspective, we would have $w_{eff} = w$.

If we have unlinked or randomly linked, we would have $w_{eff} = w^*$. This is the value that we have for batch processing - where we have the possibility of having multiple fingerprints scan the same cells.

If we look at the unlinked case, we see that we are not taking advantage of the $w \times v - w^*$ compromised cells - they are not contributing to the detection. Could we develop a best-case linked fingerprint scenario? I think for this, we would need to take advantage of the contiguous segment characteristic.

If we assume column-wise selection, the likelihood that the second compromised block will have full shadowing is $\frac{1}{b}$ - of the b possible columns for the compromised segment to start, if the same one is chosen, then we get full shadowing. For all the compromised segments, $Pr(Full\ Shadowing) = \left(\frac{1}{b}\right)^{v-1}$. This obviously does not consider the possibility of the adversary crafting her attack to take advantage of column-wise sampling. If the adversary knows the value of b and is able to modified cells without impacting critical code locations, which may not present a substantial problem if d presents a wide enough range to select from.

If we use modulo multiplication to build the fingerprints, we have b possible starting points for the code and $b - 1$ patterns so the probability of getting full shadowing is $Pr(Full\ Shadowing) = \left(\frac{1}{b(b-1)}\right)^{v-1}$.

If we use the sampling without replacement to build the fingerprints, then the probability that the second fingerprint will match the first is calculated by how likely it is for each cell from the compromised cell to match. It is not important that the first cell from the first sample is the first cell from the second sample, but that it picks one of them. The first segment covers w out of b cells. To obtain full shadowing, the first cell chosen from the second segment must fit into one of the w fingerprints. In order to get full shadowing, the second fingerprint will select one of the $w - 1$

remaining compromised cells, out of the $b-1$ available cells. This continues until the last of the w compromised cells are put into the remaining fingerprint that already covers a compromised cell.

$$\begin{aligned} Pr(Full\ Shadowing_2) &= \frac{w}{b} \times \frac{w-1}{b-1} \dots \frac{1}{b-w} \\ &= \frac{w!(b-w-1)!}{b!} \end{aligned} \tag{C.6}$$

For the remaining $v-2$ compromised segments, the probability of catching full shadowing is the same, and independent for each.

$$Pr(Full\ Shadowing_n) = \left(\frac{w!(b-w-1)!}{b!}\right)^{v-1} \tag{C.7}$$

C.3 Fingerprint Robustness

C.3.1 How to Define Robustness for Sampled Authentication

When we think about using a sampled rather than an exhaustive basis to ascertain that the SW image is intact, there are a few questions to consider.

1. **New SW or current SW:** When an ECU receives new SW, it needs to verify the origin, freshness, and applicability of the SW. DSAs and asymmetric cryptography work well to establish sufficient levels of trust, since this happens relatively infrequently and SW updates can be postponed until the vehicle is in a safe state, the confidence gained by using these established algorithms warrants the time it takes to execute the computation-intensive algorithms on slow processors. When starting the vehicle, most ECUs do not have sufficient time to verify the memory space with asymmetric algorithms. Symmetric algorithms are faster, but require significant infrastructure to securely distribute keys. With this in mind, an ECU-specific symmetric key is able to provide robust detection of changes between an initial state and the current state in the ECU. A HTA with protected memory access can be used to build a robust strategy.

2. **Primary Detection:** We would like to know how effective the scheme is at detecting compromised code. Exhaustive checks provide guaranteed detection, but all sampling schemes, since they do not check every single cell, must evaluate the detection effectiveness in terms of probability of escaping detection. How effectively can the scheme detect the compromise on the first ignition cycle after introduction?

3. **Secondary Detection:** Considering the case where the attacker is able to evade detection on the initial boot cycle, will it eventually detect the modification? Does the probability of escaping detection decrease on each subsequent ignition cycle?

4. **Sustainable Detection:** Let us assume that the adversary has studied the behavior of a specific module and is able to build a profile of the fingerprints. Can he determine a strategy to avoid detection altogether? How effectively can he shadow his modifications to reduce the detection rate?

5. **Scalability:** Let us assume that the adversary invests significant effort into successfully compromising a single ECU. Is it possible for an adversary to use this to easily compromise additional devices?

C.3.2 SSB Variants

There are a few different schemes for implementing the slides from SSB. The simplest is to use column-wise slices, so slice j is constructed by building a fingerprint from the j^{th} cell from each row. The second option is to use modulo addition with each row having it's own unique offset. This is computationally simple to implement when $b = 2^i$. Rather than storing the value for each row's offset, it is possible to generate them with a function. This provides b different offsets. The third option is to use modulo multiplication, but either $b \in \mathbb{P}$, or GCD(b,index)=1, with $b - 1$ different patterns (index=0 needs to be avoided). The last option is to have a random selection without replacement, giving $b!$ possible patterns. SSB needs to protect each cell with one fingerprint, so it is important to sample without replacement.

Table C.1: Effective Coverage Rate

Scheme	Maximum	Average	Minimum	Pr(Full Shadow)
Random with Replacement	1	w^*	0	$\left(\sum_{i=0}^{w}\binom{b}{i}\left(\frac{w}{b}\right)^i\left(1-\frac{w}{b}\right)^{b-i}\right)^v$
SSB column-wise	$v \times w$ w	w^* w	w w	$\left(\frac{1}{b}\right)^{v-1}$ if algorithm secret 1 if algorithm known
SSB Modulo +	$v \times w$	w^*	w	$\left(\frac{1}{b}\right)^{v-1}$ if algorithm known
SSB Modulo *	$v \times w$	w^*	w	$\left(\frac{1}{b(b-1)}\right)^{v-1}$
SSB Random	$v \times w$	w^*	w	$\left(\frac{w!(b-w-1)!}{b!}\right)^{v-1}$

C.3.3 Initial Detection

For each of these instances, we guarantee that at least w of the fingerprints will detect compromised memory cells. For the SSB column-wise, an adversary with knowledge of the algorithm with attempt to construct the attack with shadowing. For the other SSB variants, the pattern would be unique per ECU and difficult to predict beforehand, and on average w^* fingerprints would detect compromised memory cells - the same as with batch fingerprints.

C.3.4 Long-Term Detection

If we assume that the adversary has the ability to monitor the complete fingerprint patterns, what is the best that he could do? For batch, assuming that we re-use the fingerprints, the adversary could measure the bias in the fingerprint coverage. On average, we are assuming that each cell has a probability of being checked = $\frac{1}{b}$, and while that may be true for the next unknown fingerprint, an adversary could determine that there are cells which are protected infrequently or perhaps not at all. This would make the long-term protection decrease to nCw (1/b)cells (1-1/b)f.

For SSB Column-wise, the adversary does not have control over the selection of which fingerprint to use, so the detection rate will still be at least based on w. For SSB Modulo*, I think that the long-term will still be w*, as he cannot find contiguous cells to compromise.

BIBLIOGRAPHY

[1] Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea, and Adam Stubblefield. *Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems*. O'Reilly, 2020.

[2] National Highway Traffic Safety Administration. Automated vehicles for safety. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

[3] National Highway Traffic Safety Administration. U.s. dot announces 2017 roadway fatalities down — nhtsa. https://www.nhtsa.gov/press-releases/us-dot-announces-2017-roadway-fatalities-down.

[4] National Highway Traffic Safety Administration. Cybersecurity best practices for the safety of modern vehicles, updated 2022. https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-09/cybersecurity-best-practices-safety-modern-vehicles-2022-tag.pdf, 9 2022.

[5] National Highway Traffic Safety Administration. Nhtsa's 2021 estimate of traffic deaths shows 16-year high. https://www.nhtsa.gov/press-releases/early-estimate-2021-traffic-fatalities, 5 2022.

[6] Shohin Aheleroff, Xun Xu, Ray Y. Zhong, and Yuqian Lu. Digital twin as a service (dtaas) in industry 4.0: An architecture reference model. *Advanced Engineering Informatics*, 47:101225, 1 2021.

[7] Masoom Alam, Tamleek Ali, Sanaullah Khan, Shahbaz Khan, Muhammad Ali, Mohammad Nauman, Amir Hayat, Muhammad Khurram Khan, and Khaled Alghathbar. Analysis of existing remote attestation techniques. *Security and Communication Networks*, 5:1062–1082, 2012.

[8] Gunnar Alendal, Geir Olav Dyrkolbotn, and Stefan Axelsson. Forensics acquisition analysis and circumvention of samsung secure boot enforced common criteria mode. https://doi.org/10.1016/j.diin.2018.01.008, 2018.

[9] Amazon. Software-defined vehicle — automotive cloud solutions. https://aws.amazon.com/automotive/software-defined-vehicle/.

[10] Sigurd Frej Joel Jørgensen Ankergård, Edlira Dushku, and Nicola Dragoni. State-of-the-art software-based remote attestation: Opportunities and open issues for internet of things. *Sensors*, 21:1–23, 3 2021.

[11] Anuradha Annaswamy, Samarjit Chakraborty, Dip Goswami, S. Ramesh, and Marilyn Wolf. Trustworthy cyber physical systems. *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, 2016.

[12] Apple. Hardware security overview. https://support.apple.com/guide/security/hardware-security-overview-secf020d1074/1/web/1.

[13] Apple. ios and ipados secure boot chain. https://support.apple.com/guide/security/ios-and-ipados-secure-boot-chain-secb3000f149/web.

[14] Apple. Startup security utility on a mac with an apple t2 security chip - apple support. https://support.apple.com/guide/security/startup-security-utility-secc7b34e5b5/1/web/1, 2 2021.

[15] Apple. Apple platform security - apple support. https://support.apple.com/guide/security/welcome/web, 5 2022.

[16] Aptive. What is a software-defined vehicle? https://www.aptiv.com/en/insights/article/what-is-a-software-defined-vehicle.

[17] William A. Arbaugh, David J. Farber, and Jonathan M. Smith. Secure and reliable bootstrap architecture. *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 65–71, 1997.

[18] Archlinux. Secure boot. https://wiki.archlinux.org/index.php/Secure_Boot.

[19] Archlinux. Unified extensible firmware interface/secure boot - archwiki. https://wiki.archlinux.org/title/Unified_Extensible_Firmware_Interface/Secure_Boot, 10 2022.

[20] Orlando Arias, Fahim Rahman, Mark Tehranipoor, and Yier Jin. Device attestation: Past, present, and future. *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, 2018-January:473–478, 4 2018.

[21] Orlando Arias, Dean Sullivan, Haoqi Shan, and Yier Jin. Lahel: Lightweight attestation hardening embedded devices using macrocells. *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020*, pages 305–315, 12 2020.

[22] ACEA European Automobile Manufacturers' Association. Automobile industry pocket guide 2022-2023. https://www.acea.auto/publication/automobile-industry-pocket-guide-2022-2023/.

[23] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 14, 6 2011.

[24] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 598–610, 2007.

[25] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. *SecureComm*, 2008.

[26] AUTOSAR. Some/ip protocol specification. https://www.autosar.org/fileadmin/standards/R1-3/FO/AUTOSAR_PRS_SOMEIPProtocol.pdf, 2017.

[27] Kwang-Hyun Baek, Sergey Bratus, Sara Sinclair, and Sean W Smith. Attacking and defending networked embedded devices. https://cs.dartmouth.edu/~sws/pubs/bbss07.pdf.

[28] Network Bandwidth and Qijun Gu. Phenotyping. *Encyclopedia of Cryptography and Security*, pages 927–927, 2011.

[29] Michael Batty. Digital twins. *Environment and Planning B: Urban Analytics and City Science*, 45:817–820, 9 2018.

[30] Marco Bertoni and Alessandro Bertoni. Designing solutions with the product-service systems digital twin: What is now and what is next? *Computers in Industry*, 138:103629, 6 2022.

[31] Oleksiy Bondarenko and Tetsugo Fukuda. Development of a diesel engine's digital twin for predicting propulsion system dynamics. *Energy*, 196:117126, 4 2020.

[32] Bosch. Bosch software-defined vehicle. https://www.bosch-mobility-solutions.com/en/mobility-topics/software-defined-vehicle/.

[33] Bosch. Digital twin — bosch iot things documentation. https://docs.bosch-iot-suite.com/things/getting-started/twin/.

[34] Ferdinand Brasser, Kasper B. Rasmussen, Ahmad Reza Sadeghi, and Gene Tsudik. Remote attestation for low-end embedded devices: The prover's perspective. *Proceedings - Design Automation Conference*, 05-09-June-2016, 6 2016.

[35] Oliver Bubek, Jens Gramm, and Markus Ihle. A hardware security module for engine control units. 2011.

[36] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 950:275–286, 1995.

[37] Levente Buttyan and Jean-Pierre Hubaux. *Security and Cooperation in Wireless Networks Thwarting Malicious and Selfish Behavior in the Age of Ubiquitous Computing*. Cambridge University Press, 1.5.1 edition, 6 2012.

[38] Xavier Carpent, Karim Eldefrawy, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. Invited: Reconciling remote attestation and safety-critical operation on simple iot devices. 2018.

[39] Xavier Carpent, Norrathep Rattanavipanon, and Gene Tsudik. Remote attestation via self-measurement. *ACM Transactions on Design Automation of Electronic Systems*, 24, 1 2019.

[40] Claude Castelluccia, Aurélien Francillion, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. *Proceedings of the 16th ACM conference on Computer and communications security*, pages 400–409, 11 2009.

[41] Samarjit Chakraborty, Mohammad Abdullah Al Faruque, Wanli Chang, Dip Goswami, Marilyn Wolf, and Qi Zhu. Automotive cyber-physical systems: A tutorial introduction. *IEEE Design and Test*, 2016.

[42] Fotios Chantzis, Ioannis Stais, Paulino Calderon, Evangelos Deirmentzoglou, and Beau Woods. *Practical IOT Hacking: The Definitive Guide to Attacking the Internet of Things*. No Starch Press, 2021.

[43] Anupam Chattopadhyay and Kwok Yan Lam. Security of autonomous vehicle as a cyber-physical system. *2017 7th International Symposium on Embedded Computing and System Design, ISED 2017*, 2018.

[44] Stephen Checkoway, Damon Mccoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. *Proceedings of the 20th USENIX Conference on Security*, 2011.

[45] Kyong-Tak Cho and Kang Shin. Viden: Attacker identification on in-vehicle networks. 2017.

[46] Kyong-Tak Cho and Kang G. Shin. Error handling of in-vehicle networks makes them vulnerable. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 2016.

[47] Kyong Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. *Proceedings of the 25th USENIX Security Symposium*, pages 911–927, 2016.

[48] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10:63–81, 6 2011.

[49] Vehicle Electrical System Security Committee. J3101_202002: Hardware protected security for ground vehicles. https://www.sae.org/standards/content/j3101_202002/, 2 2020.

[50] Marco Conti, Sajal K. Das, Chatschik Bisdikian, Mohan Kumar, Lionel M. Ni, Andrea Passarella, George Roussos, Gerhard Tröster, Gene Tsudik, and Franco Zambonelli. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyberphysical convergence. *Pervasive and Mobile Computing*, 8:2–21, 2012.

[51] Philip Cooke. Image and reality: 'digital twins' in smart factory automotive process innovation – critical issues. https://0-doi-org.wizard.umd.umich.edu/10.1080/00343404.2021.1959544, 2021.

[52] Mauro Cordella, Felice Alfieri, Christian Clemm, and Anton Berwald. Durability of smartphones: A technical analysis of reliability and repairability aspects. *Journal of Cleaner Production*, 286, 3 2020.

[53] Renesas Electronics Corporation. Security-conscious debugging methods for rh850 devices (main-core debugging). https://www.renesas.com/us/en/document/apn/security-conscious-debugging-methods-rh850-devices-main-core-debugging?language=en, 5 2020.

[54] Gianpiero Costantino, Antonio La Marra, Fabio Martinelli, and Ilaria Matteucci. Candy: A social engineering attack to leak information from infotainment system. *IEEE Vehicular Technology Conference*, 2018-June, 2018.

[55] Cybersecurity and Infrastructure Security Agency. Cisa, fbi, and cnmf release advisory on multiple nation-state threat actors exploit cve-2022-47966 and cve-2022-42475 — cisa. https://www.cisa.gov/news-events/alerts/2023/09/07/, 9 2023.

[56] Alvaro A Cárdenas, Saurabh Amin, Bruno Sinopoli, Annarita Giani, Adrian Perrig, and Shankar Sastry. Challenges for securing cyber physical systems. https://pdfs.semanticscholar.org/d514/97e5827cc00d9d00c26e27a769d42284cfba.pdf.

[57] Heini Bergsson Debes, Edlira Dushku, Ubitech Ltd agiannetsos, ubitecheu Ali Marandi, Thanassis Giannetsos, and Ali Marandi. Zekra: Zero-knowledge control-flow attestation. https://doi.org/10.1145/3579856.3582833.

[58] Jayme Deerwester. The average age of a car in the us is 12.2 years, a new record. https://www.usatoday.com/story/money/cars/2022/05/24/average-american-car-12-years-old/9907901002/, 5 2022.

[59] Karim El Defrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik. Smart: Secure and minimal architecture for (establishing a dynamic) root of trust.

[60] Kurt Dietrich and Johannes Winter. Secure boot revisited. *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008*, pages 2360–2365, 2008.

[61] Xuhua Ding and Gene Tsudik. Initializing trust in smart devices via presence attestation. *Computer Communications*, 131:35–38, 10 2018.

[62] Cuong T Do, Nguyen H Tran, Choongseon Hong, Charles A Kamhoua, Kevin A Kwiat, Erik Blasch, Shaolei Ren, Niki Pissinou, and Sundaraja Sitharama. 30 game theory for cyber security and privacy. *ACM Comput. Surv*, 50, 2017.

[63] Andreas Dolezal and Ulrich Bayer. Manipulationsschutz fuer antriebssteuergeraete vernetzter, autonomer fahrzeuge in der automobilindustrie. 9 2020.

[64] Andreas Dolezal and Martin Schmiedecker. Identifikation von angriffspfaden fuer threat-analysis und risk-assessment durch design und systems thinking. 8 2023.

[65] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V. Mancini, and Silvio Ranise. Sara: Secure asynchronous remote attestation for iot systems. *IEEE Transactions on Information Forensics and Security*, 15:3123–3136, 2020.

[66] Edlira Dushku, Md Masoom Rabbani, Jo Vliegen, An Braeken, and Nele Mentens. Prove: Provable remote attestation for public verifiability. *Journal of Information Security and Applications*, 75:103448, 6 2023.

[67] Mahmoud Hashem Eiza and Qiang Ni. Driving with sharks: Rethinking connected vehicles with vehicle cybersecurity. *IEEE Vehicular Technology Magazine*, 12, 2017.

[68] Karim Eldefrawy, Sky Faber, and Tyler Kaczmarek. Proactively secure cloud-enabled storage. *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1499–1509, 6 2017.

[69] Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Hy-dra: Hybrid design for remote attestation (using a formally verified microkernel). 12, 2017.

[70] Karim Eldefrawy and Gene Tsudik. Advancing remote attestation via computer-aided formal verification of designs and synthesis of executables. *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks - WiSec '19*, pages 45–48, 2019.

[71] Elektrobit. Automotive embedded security with eb zentur. https://www.elektrobit.com/products/security/.

[72] Nikolay Elenkov. *Android Security Internals: An in-Depth Guide to Android's Security Architecture*. No Starch Press, 2014.

[73] Elsevier. Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes. https://reader.elsevier.com/reader/sd/pii/S1383762121001661.

[74] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 213–222, 2009.

[75] Escrypt. Cycurhsm. https://www.escrypt.com/en/products/cycurhsm.

[76] Neil Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010.

[77] International Organization for Standardization. Iso 26262-1:2018(en), road vehicles — functional safety — part 1: Vocabulary. https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en, 2018.

[78] Daniel S. Fowler, Madeline Cheah, Siraj Ahmed Shaikh, and Jeremy Bryans. Towards a testbed for automotive cybersecurity. *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017*, 2017.

[79] Aurelien Francillon, Quan Nguyen, Kasper B. Rasmussen, and Gene Tsudik. A minimalist approach to remote attestation. *2014 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, 4 2014.

[80] Evan Gilman and Doug Barth. *Zero Trust Networks*. O'Reilly, 7 2017.

[81] Vanessa Gratzer and David Naccache. Alien vs. quine, the vanishing circuit and other tales from the industry's crypt. 2006.

[82] Michael W Grieves. Plm–beyond lean manufacturing. *Manufacturing Engineering*, 130:23, 3 2003.

[83] Bogdan Groza and Pal Stefan Murvay. Efficient intrusion detection with bloom filtering in controller area networks. *IEEE Transactions on Information Forensics and Security*, 14, 2019.

[84] Badis Hammi, Jean Philippe Monteuuis, Houda Labiod, Rida Khatoun, and Ahmed Serhrouchni. Using butterfly keys: A performance study of pseudonym certificates requests in c-its. *2017 1st Cyber Security in Networking Conference (CSNet)*, pages 1–6, 10 2017.

[85] Hedges. How many cars are there in the world in 2023? statistics by country. https://hedgescompany.com/blog/2021/06/how-many-cars-are-there-in-the-world/.

[86] Greg Hogan. Tesla ap1 epas and 2018 honda accord eps both have some sort of can terminal which allows arbitrary memory read write. https://twitter.com/gregjhogan, 3 2020.

[87] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks - practical examples and selected short-term countermeasures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5219 LNCS:235–248, 2008.

[88] Zhe Hou, Qinyi Li, Ernest Foo, Jin Song Dong, and Paulo De Souza. A digital twin runtime verification framework for protecting satellites systems from cyber attacks. *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 2022-March:117–122, 2022.

[89] Dijang Huang, Ankur Chowdhary, and Sandeep Pisharody. *Software-Define Networking and Security (Data-Enabled Engineering)*. CRC Press, 2019.

[90] Infineon. Aurix™ security solutions. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-security-solutions/.

[91] Infineon. Aurix™ security solutions. https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/aurix-security-solutions/, 2020.

[92] Michael Jacoby and Thomas Usländer. Digital twin and internet of things-current standards landscape. *Applied Sciences (Switzerland)*, 10, 9 2020.

[93] Michael Janosko, Hunter King, Betsy (Adrienne Elizabeth) Beyer, and Max Saltonstall. Beyondcorp 6: Building a healthy fleet. https://research.google/pubs/pub47356/, 2018.

[94] Hang Jiang, Rui Chang, Lu Ren, and Weiyu Dong. Implementing a arm-based secure boot scheme for the isolated execution environment. *2017 13th International Conference on Computational Intelligence and Security (CIS)*, pages 336–340, 2017.

[95] Nesrine Kaaniche, Ethmane El Moustaine, and Maryline Laurent. A novel zero-knowledge scheme for proof of data possession in cloud storage applications. *Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014*, pages 522–531, 2014.

[96] Min Joo Kang and Je Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PLoS ONE*, 11, 6 2016.

[97] Robert Kaster and Di Ma. Secure sampled boot in automotive controllers. https://www.escar.info/images/Datastore/2021_escar_USA/papers/Secure_Sampled_Boot_in_Automotive_Controllers_Paper.pdf, 2021.

[98] Robert Kaster and Di Ma. Digital shadows for remote automotive sw verification. *ACM International Conference Proceeding Series*, pages 25–30, 5 2023.

[99] Robert Kaster and Di Ma. Sliced secure boot: An accelerated secure boot concept compatible with automotive safety controllers. *SAE International Journal of Connected and Automated Vehicles*, 6:365–382, 6 2023.

[100] Robert Kaster, Di Ma, Ashish Behl, and Bartosz Bakalarczyk. Sliced secure boot-sampled secure boot with re-usable fingerprints sliced secure boot-sampled secure boot with reusable fingerprints. https://www.escar.info/images/Datastore/2021_escar_EU/Papers/210920_escar2021_Konferenzband_A4_01_paper10.pdf, 2021.

[101] Ron S Kenett and Jacob Bortman. The digital twin in industry 4.0: A wide-angle perspective. *Quality Reliability Engineering International*, 38:1357–1366, 2022.

[102] El Khayari. Secure automotive on-board electronics network architecture. http://www.eurecom.fr/fr/publication/3132/download/rs-publi-3132.pdf.

[103] Alissa Knight. *Hacking Connected Cars Tactics, Techniques, and Procedures*. Wiley, 2020.

[104] Boyu Kuang, Anmin Fu, Willy Susilo, Shui Yu, and Yansong Gao. A survey of remote attestation in internet of things: Attacks, countermeasures, and prospects. *Computers & Security*, 112:102498, 1 2022.

[105] Boyu Kuang, Anmin Fu, Shui Yu, Guomin Yang, Mang Su, and Yuqing Zhang. Esdra: An efficient and secure distributed remote attestation scheme for iot swarms. *IEEE Internet of Things Journal*, 6:8372–8383, 10 2019.

[106] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. *Proceedings - 2017 15th Annual Conference on Privacy, Security and Trust, PST 2017*, 2018.

[107] Andrew J Leslie, Raymond J Kiefer, Michael R Meitzner, and Carol A Flannagan. Analysis of the field effectiveness of general motors production active safety and advanced headlighting systems. 2019.

[108] Matan Levi, Yair Allouche, and Aryeh Kontorovich. Advanced analytics for connected car cybersecurity. *IEEE Vehicular Technology Conference*, 2018-June, 2018.

[109] Chung Wei Lin, Qi Zhu, and Alberto Sangiovanni-Vincentelli. Security-aware modeling and efficient mapping for can-based real-time distributed automotive systems. *IEEE Embedded Systems Letters*, 7, 2015.

[110] Zhen Ling, Huaiyu Yan, Xinhui Shao, Junzhou Luo, Yiling Xu, Bryan Pearson, and Xinwen Fu. Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes. *Journal of Systems Architecture*, 119:102240, 10 2021.

[111] Jiajia Liu, Shubin Zhang, Wen Sun, and Yongpeng Shi. In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, 31, 2017.

[112] Sidi Lu and Weisong Shi. Vehicle as a mobile computing platform: Opportunities and challenges. 2023.

[113] Tianbo Lu, Jinyang Zhao, Lingling Zhao, Yang Li, and Xiaoyan Zhang. Security objectives of cyber physical systems. *Proceedings - 7th International Conference on Security Technology, SecTech 2014*, 2015.

[114] Tianbo Lu, Jinyang Zhao, Lingling Zhao, Yang Li, and Xiaoyan Zhang. Towards a framework for assuring cyber physical system security. *International Journal of Security and its Applications*, 2015.

[115] Hans Löhr, Ahmad Reza Sadeghi, and Marcel Winandy. Patterns for secure boot and secure storage in computer systems. *ARES 2010 - 5th International Conference on Availability, Reliability, and Security*, pages 569–573, 2010.

[116] Di Ma and Nitesh Saxena. A context-aware approach to defend against unauthorized reading and relay attacks in rfid systems. *Security and Communication Networks*, 7:2684–2695, 12 2014.

[117] Di Ma and Gene Tsudik. A new approach to secure logging. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008.

[118] Georg Macher, Eric Armengaud, Eugen Brenner, and Christian Kreiner. Threat and risk assessment methodologies in the automotive domain. *Procedia Computer Science*, 83, 2016.

[119] Tobias Madl, Hans-Joachim Hof, Jasmin Brückmann, and Dr-Ing Hans-Joachim Hof. Reference architecture for the design of secure automotive cyber systems automotive security view project security for smart homes. https://www.researchgate.net/publication/318722901, 2017.

[120] Mirco Marchetti and Dario Stabili. Anomaly detection of can bus messages through analysis of id sequences. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2017.

[121] Mirco Marchetti, Dario Stabili, Alessandro Guido, and Michele Colajanni. Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms. *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow, RTSI 2016*, 2016.

[122] Maria Markstedter. *Arm Assembly: Internals and Reverse Engineering*. Wiley, 2023.

[123] Stefan Marksteiner, Slava Bronfman, Markus Wolf, and Eddie Lazebnik. Using cyber digital twins for automated automotive cybersecurity testing. https://arxiv.org/abs/2107.07355, 7 2021.

[124] Willem Melching. Hacking a vw golf power steering ecu - part 4. https://icanhack.nl/blog/vw-part4/, 2022.

[125] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[126] Ralph Merkle. Method of providing digital signatures. https://patents.google.com/patent/US4309569A/en, 1982.

[127] ST Microelectronics. Spc58nh92c5 - 32-bit power architecture mcu for automotive general purpose applications - chorus family. https://www.st.com/en/automotive-microcontrollers/spc58nh92c5.html.

[128] Microsoft. Boot and uefi - windows drivers. https://docs.microsoft.com/en-us/windows-hardware/drivers/bringup/boot-and-uefi.

[129] Microsoft. Secure boot. https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-secure-boot.

[130] Roberto Minerva, Gyu Myoung Lee, and Noel Crespi. Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. *Proceedings of the IEEE*, 108:1785–1824, 10 2020.

[131] MITRE. Cve - cve. https://cve.mitre.org/.

[132] Arslan Munir and Farinaz Koushanfar. Design and analysis of secure and dependable automotive cps: A steer-by-wire case study. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[133] Thomas Nadeau and Ken Gray. *SDN: Software Defined Networks*. O'Reilly, 2013.

[134] K N Sridharan Namboodiri. An efficient batch verification scheme for secure vehicular communication using bilinear pairings. https://doi.org/10.1007/978-981-15-2475-2_65.

[135] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. Obd-securealert: An anomaly detection system for vehicles. *2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016*, 2016.

[136] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. Using semantic technologies to mine vehicular context for security. *37th IEEE Sarnoff Symposium, Sarnoff 2016*, 2017.

[137] Ahmad MK Nasser. *Automotive Cybersecurity Engineering Handbook The automotive engineer's roadmap to cyber-resilient vehicles*. Packt, 10 2023.

[138] Ahmad MK Nasser, Wonder Gumise, and Di Ma. Accelerated secure boot for real-time embedded safety systems. *SAE International Journal of Transportation Cybersecurity and Privacy*, 2:11–02–01–0003, 7 2019.

[139] Ahmad MK Nasser, Di Ma, and Priya Muralidharan. An approach for building security resilience in autosar based safety critical systems. https://www.riverpublishers.com/journal/journal_articles/RP_Journal_2245-1439_633.pdf, 2017.

[140] Nicolas Navet, Aurélien Monot, Bernard Bavouxt, and Françoise Simonot-Lion. Multi-source and multicore automotive ecus - os protection mechanisms and scheduling. *IEEE International Symposium on Industrial Electronics*, 2010.

[141] Tu N. Nguyen, Sherali Zeadally, and Abhijith B. Vuduthala. Cyber-physical cloud manufacturing systems with digital twins. *IEEE Internet Computing*, 26:15–21, 2022.

[142] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, Michael Steiner, and Gene Tsudik. Vrased: A verified hardware/software co-design for remote attestation. *28th USENIX Security Symposium*, 8 2019.

[143] Ivan De Oliveira Nunes, Karim Eldefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Pure: Using verified remote attestation to obtain proofs of update, reset and erasure in low-end embedded systems. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers ICCAD*, 11 2019.

[144] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, Norrathep Rattanavipanon, and Gene Tsudik. On the toctou problem in remote attestation. *CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2921–2936, 11 2021.

[145] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, Norrathep Rattanavipanon, and Gene Tsudik. Towards remotely verifiable software integrity in resource-constrained iot devices. *IEEE Communications Magazine*, pages 1–7, 2024.

[146] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, and Gene Tsudik. Dialed: Data integrity attestation for low-end embedded devices. *Proceedings - Design Automation Conference*, 2021-December:313–318, 12 2021.

[147] NXP. Hsm and she on i.mx 8qxp and i.mx 8dxl. https://www.nxp.com/docs/en/application-note/AN12906.pdf, 4 2021.

[148] Society of Automotive Engineers. Sae levels of driving automation™ refined for clarity and international audience. https://www.sae.org/blog/sae-j3016-update.

[149] Society of Automotive Engineers. Sae j2980. https://www.sae.org/standards/content/j2980_202310/, 10 2023.

[150] Surjya Kanta Pal, Debasish Mishra, Arpan Pal, Samik Dutta, Debashish Chakravarty, and Srikanta Pal. Digital twin – fundamental concepts to applications in advanced manufacturing. 2022.

[151] Simon Parkinson, Paul Ward, Kyle Wilson, and Jonathan Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 18:2898–2915, 11 2017.

[152] Jonathan Petit and Steven E. Shladover. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11, 2014.

[153] G Pocklassery, W Che, F Saqib, M Areno, and J Plusquellic. Self-authenticating secure boot for fpgas. *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 221–226, 2018.

[154] Enrico Pozzobon, Nils Weiss, Jürgen Mottok, and Václav Matoušek. Fuzzy fault injection attacks against secure automotive bootloaders. *Escar*, 2023.

[155] R. V. Rashmi and A. Karthikeyan. Secure boot of embedded applications - a review. *Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology, ICECA 2018*, pages 291–298, 9 2018.

[156] Roland Rieke, Marc Seidemann, Elise Kengni Talla, Daniel Zelle, and Bernhard Seeger. Behavior analysis for safety and security in automotive systems. *Proceedings - 2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2017*, 2017.

[157] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Nist special publication 800-207 zero trust architecture. https://doi.org/10.6028/NIST.SP.800-207, 2020.

[158] Jerome Saltzer and M Schroeder. The protection of information in computer systems. http://ieeexplore.ieee.org/document/1451869/, 1975.

[159] Steffen Sanwald, Liron Kaneti, Marc Stöttinger, and Martin Böhner. Secure boot revisited: Challenges for secure implementations in the automotive domain. *SAE International Journal of Transportation Cybersecurity and Privacy*, 2, 8 2020.

[160] Michael Schluse, Marc Priggemeyer, Linus Atorf, and Juergen Rossmann. Experimentable digital twins-streamlining simulation-based systems engineering for industry 4.0. *IEEE Transactions on Industrial Informatics*, 14:1722–1731, 4 2018.

[161] Steve Schmidt. What is zero trust on aws? – amazon web services (aws). https://aws.amazon.com/security/zero-trust/.

[162] Offensive Security. Exploit database shellcodes. https://www.exploit-db.com/shellcodes, 2023.

[163] NXP Semiconductors. S32g2 safe and secure vehicle network processor. https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32g-vehicle-network-processors/s32g2-processors-for-vehicle-networking:S32G2, 5 2022.

[164] Ioannis Sfyrakis and Thomas Gross. A survey on hardware approaches for remote attestation in network infrastructures. http://arxiv.org/abs/2005.12453, 5 2020.

[165] Barry Sheehan, Finbarr Murphy, Martin Mullins, and Cian Ryan. Connected and autonomous vehicles: A cyber-risk classification framework. *Transportation Research Part A: Policy and Practice*, 11 2018.

[166] Ali Shoker, Vincent Rahli, Jérémie Decouchant, and Paulo Esteves-Verissimo. Intrusion resilience systems for modern vehicles. *IEEE Vehicular Technology Conference*, 2023-June, 2023.

[167] Bluetooth SIG. Bluetooth market update 2020 white paper. https://www.bluetooth.com/wp-content/uploads/2020/03/2020_Market_Update-EN.pdf, 2020.

[168] Purnendu Sinha. Dynamic task-level reconfiguration in automotive software architectures. *Proceedings of the 6th India Software Engineering Conference on - ISEC '13*, 2013.

[169] Dirk Slama, Achim Nonnenmacher, and Thomas Irawan. *The Software-Defined Vehicle: A Digital-First Approach to Creating Next-Generation Experiences*. O'Reilly Media, 2023.

[170] Craig Smith. *The Car Hacker's Handbook A Guide for the Penetration Tester*. No Starch Press, 2016.

[171] Yoo Ho Son, Kyu Tae Park, Donggun Lee, Seung Woo Jeon, and Sang Do Noh. Digital twin-based cyber-physical system for automotive body production lines. *The International Journal of Advanced Manufacturing Technology*, 115:291–310, 2021.

[172] Dario Stabili, Luca Ferretti, and Mirco Marchetti. Analyses of secure automotive communication protocols and their impact on vehicles life-cycle. *Proceedings - 2018 IEEE International Conference on Smart Computing, SMARTCOMP 2018*, 2018.

[173] Statista. U.s.: smartphones replacement cycle 2013-2027. https://www.statista.com/statistics/619788/average-smartphone-life/.

[174] Statista. U.s.: vehicles in operation 2022. https://www.statista.com/statistics/859950/vehicles-in-operation-by-quarter-united-states/.

[175] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-hellman key distribution extended to group communication. *Proceedings of the ACM Conference on Computer and Communications Security*, 1996.

[176] Frederic Stumpf. Hardware security modules for embedded systems. https://studylib.net/doc/8154745/hardware-security-modules-forembedded-systems, 2011.

[177] Darell J J Tan, Tong wei Chua, and Vrizlynn L L Thing. Securing android: A survey, taxonomy, and challenges. *ACM Comput. Surv*, 47, 2015.

[178] Hailun Tan, Wen Hu, and Sanjay Jha. A remote attestation protocol with trusted platform modules (tpms) in wireless sensor networks. *Security and Communication Networks*, 8:2171–2188, 9 2015.

[179] Hailun Tan, Gene Tsudik, and Sanjay Jha. Mtra: Multi-tier randomized remote attestation in iot networks. *Computers and Security*, 81:78–93, 3 2019.

[180] Fei Tao, Qinglin Qi, Lihui Wang, and A. Y.C. Nee. Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, 5:653–661, 8 2019.

[181] TCG. Tcg mobile trusted module specification specification version 1.0. https:\www.trustedcomputinggroup.org, 4 2010.

[182] Adam Thelen, Xiaoge Zhang, Olga Fink, Yan Lu, Sayan Ghosh, Byeng D. Youn, Michael D. Todd, Sankaran Mahadevan, Chao Hu, and Zhen Hu. A comprehensive review of digital twin – part 1: Modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization 2022 65:12*, 65:1–55, 8 2022.

[183] Vrizlynn L.L. Thing and Jiaxi Wu. Autonomous vehicle security: A taxonomy of attacks and defences. *Proceedings - 2016 IEEE International Conference on Internet of Things; IEEE Green Computing and Communications; IEEE Cyber, Physical, and Social Computing; IEEE Smart Data, iThings-GreenCom-CPSCom-Smart Data 2016*, 2017.

[184] Andrew Tomlinson, Jeremy Bryans, Siraj Ahmed Shaikh, and Harsha Kumara Kalutarage. Detection of automotive can cyber-attacks by identifying packet timing anomalies in time windows. *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*, pages 231–238, 7 2018.

[185] Mathieu Trudel-Lapierre. How to sign things for secure boot — ubuntu. https://ubuntu.com/blog/how-to-sign-things-for-secure-boot, 8 2017.

[186] Gene Tsudik. Challenges in remote attestation of low-end embedded devices. pages 1–1, 2014.

[187] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. Intra-vehicle networks: A review, 2015.

[188] Ubuntu. How to sign things for secure boot. https://ubuntu.com/blog/how-to-sign-things-for-secure-boot.

[189] Uptane. Uptane securing sw updates for automobiles. https://uptane.github.io/, 2022.

[190] Chris Valasek and Charlie Miller. Remote exploitation of an unaltered passenger vehicle. https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Car_Hacking-1.pdf, 2015.

[191] Aurélien Vasselle, Hugues Thiebeauld, Quentin Maouhoub, Adèle Morisset, and Sébastien Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot-extended version. *IEEE Transactions on Computers*, 69:1449–1459, 2020.

[192] AIAG & VDA. *AIAG & VDA FMEA Handbook*. Automotive Industry Action Group, 1 edition, 2019.

[193] Vector. Automotive safety and security. https://www.vector.com/int/en/products/solutions/safety-security/.

[194] Qian Wang, Zhaojun Lu, and Gang Qu. An entropy analysis based intrusion detection system for controller area network in vehicles. *International System on Chip Conference*, 2018-September:174–179, 1 2019.

[195] Rory Ward and Betsy Beyer. Beyondcorp: A new approach to enterprise security. https://research.google/pubs/pub43231/, 2014.

[196] André Weimerskirch. Secure software flashing. *SAE Technical Papers*, pages 83–86, 2009.

[197] Marko Wolf, Andre Weimerskirch, and Christof Paar. *Secure In-Vehicle Communication*, pages 95–110. Springer, 2006.

[198] Samuel Woo, Hyo Jin Jo, In Seok Kim, and Dong Hoon Lee. A practical security architecture for in-vehicle can-fd. *IEEE Transactions on Intelligent Transportation Systems*, 17, 2016.

[199] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. A practical wireless attack on the connected car and security protocol for in-vehicle can. *IEEE Transactions on Intelligent Transportation Systems*, 16, 2015.

[200] Liu Xian'gang, Xing Zhang, Yingfang Fu, and Changxiang Shen. National high technology research and development program of china and the national basic research program of china. 15, 2010.

[201] Guoqi Xie, Wei Wu, Gang Zeng, Renfa Li, and Shiyan Hu. Risk assessment and development cost optimization in software defined vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 22:3675–3686, 6 2021.

[202] Linxi Zhang, Xuke Yan, and Di Ma. Accelerating in-vehicle network intrusion detection system using binarized neural network. *SAE International Journal of Advances and Current Practices in Mobility*, 4:2037–2050, 3 2022.

[203] Linxi Zhang, Xuke Yan, and Di Ma. A binarized neural network approach to accelerate in-vehicle network intrusion detection. *IEEE Access*, 10:123505–123520, 2022.

[204] Youqian Zhang and Kasper Rasmussen. Detection of electromagnetic interference attacks on sensor systems. http://www.cs.ox.ac.uk/files/11036/SP.pdf.

[205] Dong Zhong, Zhelei Xia, Yian Zhu, and Junhua Duan. Overview of predictive maintenance based on digital twin technology. https://pubmed.ncbi.nlm.nih.gov/37025897/, 2023.