CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Vehicle Authentication with Threshold Signatures

Master’s thesis in Computer science and engineering

Tommi-Roy Karlsson
Lam Nguyen

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Vehicle Authentication with Threshold Signatures

Tommi-Roy Karlsson
Lam Nguyen

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Vehicle Authentication with Threshold Signatures
Tommi-Roy Karlsson Lam Nguyen

Supervisor: Elena Pagnin, Department of Computer Science and Engineering, Chalmers University of Technology
Advisors: Ivan Oleinikov, Chalmers University of Technology,
Tomas Carlfalk, Jeffrey Spång, Anton Lundén, and Jens Andersson
Examiner: Magnus Almgren, Department of Computer Science and Engineering, Chalmers University of Technology

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Vehicle Authentication with Threshold Signatures

Lam Nguyen & Tommi-Roy Karlsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis work aims to evaluate the practical application of threshold signature schemes in vehicular settings, with a specific focus on strengthening the security and redundancy of private keys used in mutual TLS (mTLS) in vehicle-to-everything (V2X) communication. The proposed VATS (Vehicular Application of Threshold Signature) scheme introduces an innovative approach to secure key sharing among electronic control units (ECUs) within vehicles, significantly enhancing key management security. Using a carefully designed secure secret sharing scheme, the VATS scheme enables the reconstruction of the private key by using a predetermined threshold of ECUs. This approach ensures that no single ECU possesses the complete private key, mitigating the risks associated with key compromise or unauthorized access. Instead, multiple ECUs collaboratively contribute their shares to reconstruct the private key, thus establishing a higher level of security and resilience in vehicular networks. To validate the effectiveness and practicality of the VATS scheme, extensive evaluations and benchmarking have been performed. The performance of the scheme, including execution time, resource utilization, and scalability, has been measured and analyzed. These evaluations provide valuable insights into the scheme's efficiency and its ability to handle various amounts of participants in the scheme.

Keywords: Thershold Signature, RSA, ECDSA, EdDSA, Schnorr, Cryptography, VSS, PSS, Accountability

Acknowledgements

We would like to express our sincere gratitude to our supervisors Elene Pagnin and Ivan Oleinikov, for their invaluable help and guidance throughout the project. We would also like to thank Tomas Carlfalk, Jeffrey Spång, Anton Lundén, and Jens Andersson for their support and guidance as company supervisors.

Tommi-Roy Karlsson & Lam Nguyen, Gothenburg, 2023-09-18

Contents

List of Figures

List of Tables

1
Introduction

This chapter briefly introduces the challenges, motivations, and goals of this project.

1.1 Context

In the Internet of Things era, billions of devices are connected, presenting new possibilities and exposing us to greater security and privacy risks. Therefore, manufacturers must address user privacy in detail and pay more attention to security during technology development [1].

Modern vehicles can communicate and share information with each other or with servers through vehicle-to-everything communication (V2X). This capability enables them to carry out over-the-air operations, such as updating the vehicle's firmware or software, without being physically brought to a dealership. This convenient feature is becoming increasingly favoured by original equipment manufacturers (OEMs), but it also comes with the risk of being infiltrated by malicious actors. Severe consequences can be caused by such security breaches in a vehicle. These consequences include the potential loss of users' confidential data, vehicle failure due to sabotaged critical components that leads to malfunctions, and so on. Therefore, OEMs have increasingly focused on automotive cybersecurity to combat these emerging threats. As the result of this effort, proposed over-the-air schemes, such as the one offered by Shavit et al. [2], help mitigate this issue by providing firmware updates over the Internet, as seen in figure 1.1. This is just one example from a long list of challenges and risks that intelligent vehicles face in the new age of the Internet of Things.

One of the most challenging aspects of wireless communications is ensuring confidentiality, integrity, and authentication. There are existing secure communication protocols that can address this issue. Many of these protocols require some private key to be kept secret. Losing this key would compromise the privacy and security of the conversation and its participants. The protection mechanism for private keys in existing protocols has not been thoroughly discussed.

1.2 Problems and Challenges

Cryptography is a technique used to secure communication in the presence of adversarial behaviour, playing a significant role in ensuring privacy and improving security.

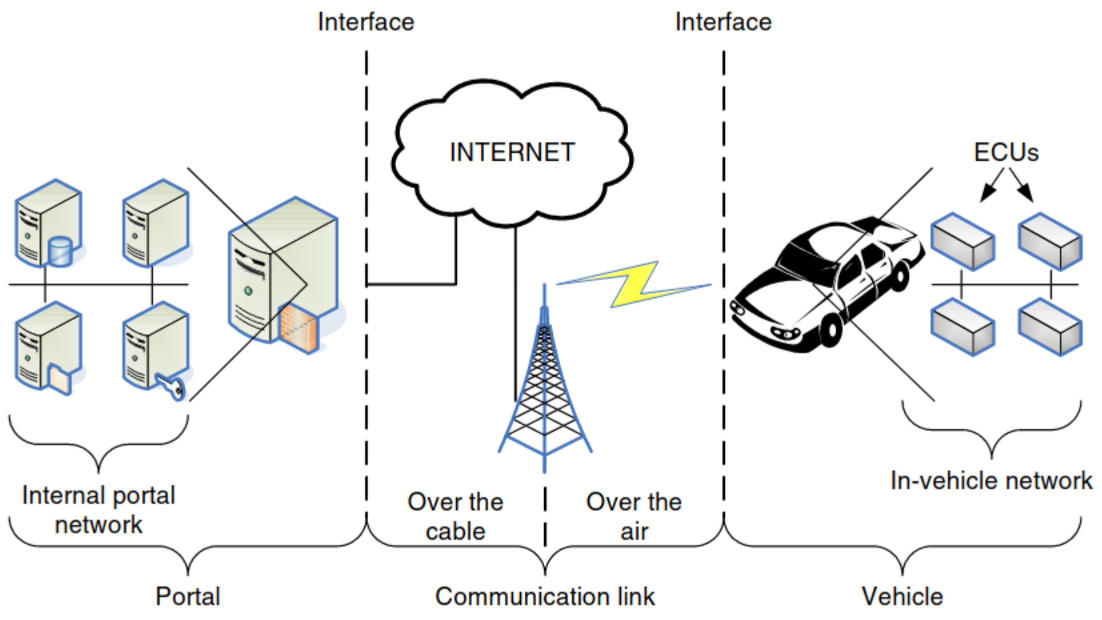


Figure 1.1: Over-the-air updates sent to vehicles. (© 2008 IEEE) [3]

Transport Layer Security (TLS) is a widely adopted security protocol that facilitates privacy and secures communication over the Internet [4]. The implemented TLS in a vehicle can establish a secure communication channel between the said vehicle and external entities, such as OEM servers, traffic lights, or other vehicles. In TLS, the digital identity of the communicating parties can be verified using asymmetric cryptography, which consists of a private key and a public key for signing, and verifying messages, respectively. It is critical to ensure the confidentiality of the private key. A compromised private key allows the adversary to forge the victim's digital identity, violating future communications' integrity.

TLS plays an essential role in securing communication with V2X. It ensures that only trusted parties can communicate and interact with the vehicle, preventing malicious attacks and unauthorised control over the vehicle. Using TLS to secure V2X communication implies the existence of a private key, which has to be stored somewhere in the vehicle to ensure that the said vehicle assumes the right digital identity during communication. Such storage can come from one of several embedded computers distributed around the vehicle, called electrical control units (ECUs). These ECUs control various vehicle functions, including external communication. In mutual TLS (mTLS) both parties are required to present valid signatures before establishing a secure communication channel with each other [5].

Keeping the private key secure is not trivial. There are several methods for creating, storing, and reconstructing a private key in the event of loss or corruption. Therefore, meticulous evaluation and strategic alignment with the security needs of the vehicular communication system are vital to ensure confidentiality and integrity in secure V2X communications. Mismanagement when creating or storing keys can lead to disastrous outcomes. Even having an unintentional malfunctioning ECU with the private key could make it impossible for the affected vehicle to communicate externally, creating a single point of failure in certificate-based authentication used when establishing TLS communication in V2X communication. As such, there is an urgent need to fortify security protocols surrounding private key management while simplifying creation processes.

1.3 Purpose

Several methods can increase private key security, strengthen redundancy, and avoid single points of failure when working with private keys. However, replicating the private key and storing it in several electronic control units (ECUs) around the vehicle can expose the key to significant security risks. This is because an attacker must only compromise a single ECU to extract the private key and falsify the victim's digital identity. Instead, a multisignature scheme can be useful to prioritise strong security. Multisignature refers to the ability of a group of signers to collectively sign a message and produce a signature in an interactive signing protocol [6]. This technique requires a quorum of ECUs to sign a message that verifies the vehicle's identity and establishes TLS-protected communication. Compared to the replication technique, multi-signature offers better security of the vehicle's identity but lacks the redundancy needed in case of ECU failures since all ECUs in the designated quorum must actively sign the message. To simultaneously address these properties, threshold signature protocols can be considered.

A cryptographic primitive called threshold signature [7] allows a group of participants to jointly own a numerical secret. In a (t, n) threshold signature scheme, the secret is distributed among n participants, and it requires at least t of them to work together to produce a valid digital signature. These schemes provide good security since no single ECU stores the private key, thus eliminating the risk of a single point of failure. As long as enough ECUs remain operational in the vehicle to cooperate and generate a signature for authentication, redundancy is ensured.

This thesis work aims to develop and evaluate the application of threshold signature schemes in vehicular settings to strengthen the security and redundancy of private keys used in mTLS communication.

1.4 Goals

The goal is to design a (t, n) threshold signature scheme suitable for vehicular use. This means that with the help of n ECUs, the key can be shared among the ECUs to improve the security of key management in vehicles. Here, the threshold t is the minimum number of ECUs needed to be able to reconstruct the secret and authenticate the vehicle. This is implemented with the help of a secret sharing scheme that allows reconstruction of the private key using t ECUs.

There are a few standard techniques to revise the secret key, such as using the surviving ECUs to reconstruct the private key and resharing the signing key to trusted devices. However, this could cause problems with the security of the private key. Therefore, it could be more beneficial to use alternatives to these resharing methods that do not involve reconstructing the secret key s.

The objective of this project is to discover solutions to the following questions:

- How can a threshold signature scheme be utilized to improve security while avoiding the single-point-of-failure problem of private key employed in TLS communication?
- What are the various techniques to sign an electronic message and distribute keys in a threshold signature scheme and how do they compare with each other?
- Can these methods be feasibly, practically and efficiently implemented in resource-constrained devices like ECUs found in automobiles?
- What properties of a threshold signature scheme are desirable in a vehicular setting, and can a new protocol be created using existing schemes to achieve said properties?

1.5 Scope and Limitations

Evaluating the limitations and setting a clear scope for the thesis work helps to keep the project's focus more aligned with the goals. These are the limitations and scope of this work.

1.5.1 Limitations

One notable limitation in implementing the proposed protocol is the challenge of testing it in a real vehicular environment. Such testing would require access to the ECUs of an actual vehicle, which is costly and overly complex for the scope of this thesis. Instead, a proof-of-concept implementation in a simulated environment that closely resembles modern vehicle ECU networks would have to suffice in this project. This alternative approach can substantially reduce the testing effort required without compromising the validity of the implementation of the protocol.

Moreover, the design of ECUs is often protected by intellectual property rights held by manufacturers or third parties, which presents a significant obstacle to adapting them to test the proposed protocol. As such, conducting simulations is a more feasible and practical approach to testing the protocol's performance in a controlled environment.

Apart from the aforementioned constraints, it is crucial to consider the limitations of computational resources. Assuming that the ECUs in a vehicle possess hardware specifications similar to those of a contemporary personal computer is not a realistic assumption for this project. This may result in the exclusion of effective protocols that are computationally intensive. Communication restrictions between ECUs, such as bandwidth and capacity, must also be considered. Non-interactive protocols are preferable since they do not strain the communication link between ECUs. Reduced communication also reduces the likelihood of information leakage during transmission, in addition to TLS measures. Although scalability is typically a critical aspect when designing protocols, it is less significant in this project since the number of ECUs in a vehicular setting is limited. Therefore, efficiency and security are of greater significance than scalability.

1.5.2 Scope

Many techniques could be considered to address the challenges mentioned in Section 1.2. But the scope of this thesis work is limited to using cryptographical approaches and threshold signatures, in particular, to address these issues. The internal communication network of a vehicle could vary for different vehicles depending on the manufacturers' design of choice. Therefore, finding a generic solution for every possible network design is not practical. Therefore, this thesis focuses on vehicles with similar communication mechanisms and network designs as described in the next Section 4.2.

The development of a new cryptographic protocol from scratch is generally discouraged due to the considerable knowledge and expertise required and the need for rigorous study, review, and peer approval to ensure its soundness. Given the timeline for this thesis project, undertaking these steps may not be feasible without sacrificing other essential aspects of the work. Therefore, the focus will be on adapting and enhancing existing, well-established, and proven protocols to simplify the creation process and enhance the new protocol's security analysis. Consequently, the proposed protocol draws inspiration from an established and proven cryptographic protocol.

1.6 Contributions

The project develops a new scheme explicitly designed for V2X communication. To ensure its practicality and effectiveness, the protocol has been designed to have certain desired properties, which include:

- Threshold tolerance with the ability to accommodate up to n possible signers and at least t signers.
- Committee selection
- Multi-signing capabilities
- Verifiable secret sharing (VSS):
  The validity of a share as part of the secret is verifiable.
- Accountability/traceability for the signing committee: The non-repudiation property of the signing committee.

- Publicly verifiable secret sharing (PVSS):
  The ability to verify the validity of other participants' shares.
- Proactive verifiable secret sharing to renew keys (PSS):
  The ability to renew shares at will without changing the original secret.
- Effective communication between parties.
- A proven and secure design.

Each of the desired properties listed for the protocol plays a vital role in ensuring the security and reliability of vehicular communication.

Threshold tolerance with the ability to accommodate up to a certain number of signers, is essential in a distributed system such as vehicular communication, where nodes may be lost or compromised. The ability to tolerate a certain number of faults or attackers is critical to maintaining the integrity of the communication.

The selection of committees is essential to ensure that the appropriate nodes are chosen to participate in the communication process. This can help prevent malicious or unreliable nodes from participating in the protocol.

Multisigning capabilities, including verifiable secret sharing, accountability, and traceability for each signer, proactive verifiable secret sharing for key renewal, and publicly verifiable secret sharing, are critical to ensuring the protocol is secure and reliable in long-lived systems. In addition, these capabilities help ensure that communication is protected against attacks while also providing a way to hold individual signers accountable if necessary.

Effective communication between parties is essential to ensure that the protocol functions appropriately and that all nodes can understand and participate in the communication.

Finally, a proven and secure design is vital to ensure that the protocol is reliable and can be trusted to protect communication. A secure design ensures that the protocol is resistant to attacks and can be relied upon to function correctly in a real-world vehicular communication environment.

To develop a suitable scheme for real-world application, all the aforementioned properties should be considered. Including all these properties strengthens the rationale for implementing the scheme proposed for use in the real world. The proposed scheme will be nicknamed **V**ehicle **A**uthentication using **T**hreshold **S**ignature **(VATS)**.

In their papers, FROST [7], ICE-FROST [8], and Musig2 [6], the authors mention possible applications of their respective schemes in cryptocurrency. Theoretically, these schemes can also be applied to other applications, including in a vehicular setting. However, the suitability and practicality of this idea have not been tested or mentioned. Therefore, another contribution of this work is the evaluation of the said schemes in our particular setting. We have also combined these well-regarded schemes into a threshold multisignature scheme with the desired properties as shown in table 1.1, which makes it more practical and suitable for vehicular applications.

	Schnorr signature [9]	FROST [7]	ICE-FROST [8]	MuSig2 [6]	VATS (Scheme 5)
Threshold tolerant	✗	✓	✓	✗	✓
Committee selection	✗	✗	✗	✗	✓
Multi-signing	✗	✓	✓	✓	✓
VSS	✗	✓	✓	✗	✓
Accountability	✓	✗	✗	✓	✓
PVSS	✗	✓	✓	✗	✓
PSS	✗	✗	✓	✗	✓

Table 1.1: Scheme properties comparison

1.7 Thesis outline

This chapter of the thesis provides an overview of the project, including its definition, objectives, and limitations. The following is a list of the remaining chapters of the thesis.

- Chapter 1 gives context and introduces the problem that vehicular authentication is facing, as well as presenting our motivations and goals with the project.
- Chapter 2 introduces various cryptographic schemes, signatures, and properties that give context to the work done in this thesis. It will also discuss the security implications of these schemes and how they can be used to protect data and communications. This background should assist in comprehending how the scheme proposed in this thesis is constructed.
- Chapter 3 is the section of related work where important papers for developing the proposed scheme are presented. This section provides an overview of the existing literature and explains how different schemes work. Additionally, it should provide a comparison between the proposed scheme and the existing ones and should explain why the proposed scheme is improved compared to the existing ones for vehicular implementation.
- Chapter 4 outlines the methods used in creating the proposed scheme and describes how the implementation was carried out so that the scheme could be tested. The types of tests that are important to evaluate the scheme are described, such as execution time, resource usage, and scalability, and why they are specifically tested.
- Chapter 5 presents the VATS scheme and its components.
- Chapter 6 describes the proofs for the scheme to ensure that the scheme is correct and secure.
- Chapter 7 is the section where the scheme evaluation and performance are presented and gives the results of the testing conducted on the scheme's im-

plementation.

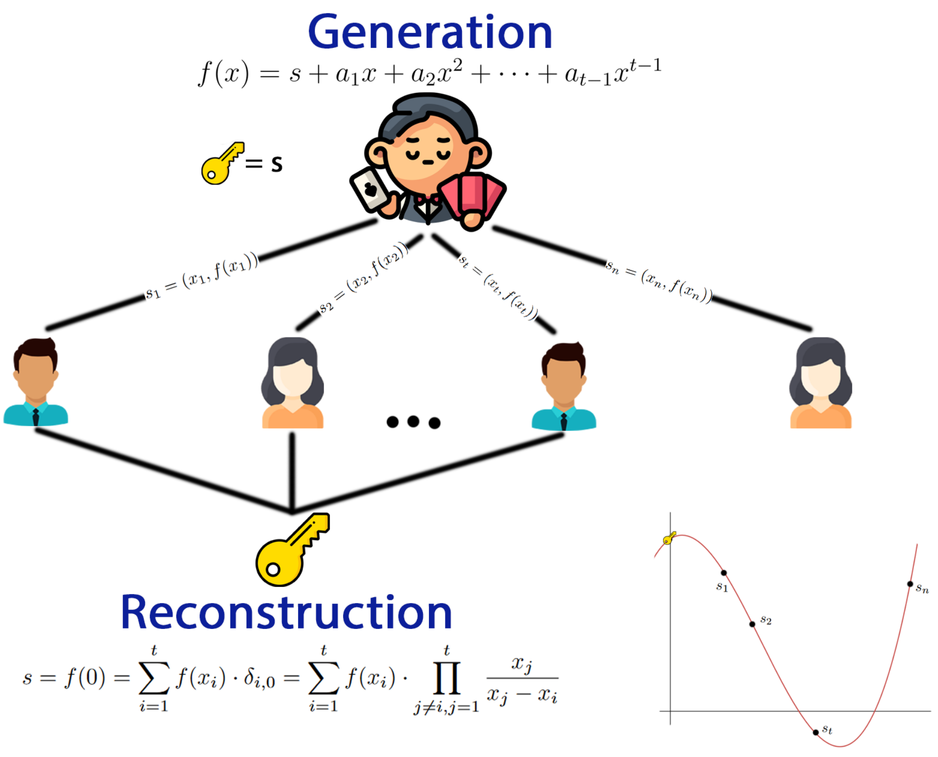
- Chapter 8 discusses the design choices made for the scheme, which gives a context as to why the scheme is done in specific ways, and it also discusses the result of the testing done in evaluation to give context to the graphs and calculations presented. In addition, the ethical aspects of the project are discussed, as well as future work.
- Chapter 9 provides the conclusion of the thesis on the scheme and its usability.

2

Background

This chapter presents fundamental background knowledge in the research area, covering cryptographic schemes and the attributes that may be desired for a scheme.

2.1 Threshold Cryptography



Generation
$f(x) = s + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$
= s
$s_1 = (x_1, f(x_1))$
$s_2 = (x_2, f(x_2))$
$s_t = (x_t, f(x_t))$
$s_n = (x_n, f(x_n))$
Reconstruction
$s = f(0) = \sum_{i=1}^{t} f(x_i) \cdot \delta_{i,0} = \sum_{i=1}^{t} f(x_i) \cdot \prod_{j \neq i, j=1}^{t} \frac{x_j}{x_j - x_i}$
s_1
s_2
s_n
s_t

Figure 2.1: Shamir's Secret Sharing scheme

The secret-sharing schemes of Shamir and Blakley [10] [11] allow a collection of n nodes to share a secret value, such that any t, $(t \leq n)$ nodes can collaborate to perform operations using the secret. The secret is only compromised when an adversary controls t or more nodes.

Shamir's Secret Sharing scheme is shown in figure 2.1 and is described as follows.

- **Key generation:**
  1. Choose a prime number p such that $p > n$, where n is the total number of shares, and t is the threshold for reconstruction.
  2. Choose a random secret key $s \in 0, 1, \ldots, p-1$.
  3. Define the polynomial $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$, where the coefficients a_i are chosen randomly and uniformly from $0, 1, \ldots, p-1$, and $a_{t-1} \neq 0$.
  4. Compute $f(i)$ for $i = 1, 2, \ldots, n$, and distribute the shares $(i, f(i))$ to n different participants.
- **Secret reconstruction:**
  1. Choose any t shares, for example $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$, where $x_1, x_2, \ldots, x_t$ are distinct.
  2. Define the Lagrange polynomial $L_i(x)$ as follows:
  $$L_i(x) = \prod_{j \neq i, j=1}^{t} \frac{x - x_j}{x_i - x_j}, \quad i = 1, 2, \ldots, t.$$
  In this work, the Lagrange polynomial $L_i(x)$ is denoted by $\delta_{i,j}$.
  3. Compute the secret key s as:
  $$s = f(0) = \sum_{i=1}^{t} y_i \cdot \delta_{i,0} = \sum_{i=1}^{t} y_i \cdot \prod_{j \neq i, j=1}^{t} \frac{x_j}{x_j - x_i}$$

There are many desirable properties that a protocol can have. Depending on different use cases, certain properties are more suitable than others. Some of these properties are described in Section 2.3. It can be noted that by setting n equal to t, one would instead get multi-signature where all participant's shares have to be used in the calculations to reconstruct the secret key s.

2.2 Digital signature

This section explains the different digital signature schemes known for being rigorously proven to be sound and commonly used. By understanding the differences and workings of these schemes, the readers can better grasp the different digital signatures that are discussed in the paper.

2.2.1 Schnorr Signature

The Schnorr signature scheme, proposed by Claus-Peter Schnorr [12], utilises a prime order cyclic group $\mathbb{G}$ with prime order p, a generator g from the group, and a hash function H.

Here is an overview of how Schnorr works:

- **Key Generation**:
  To generate a private/public key pair, a private key $x \in (0, \ldots, p-1) \in \mathbb{G}$ is chosen, and the corresponding public key is $Y = g^x$.
- **Signing**:
  Signing a message involves selecting a random integer $r \in \mathbb{Z}_p$ and computing a "number used once" (nonce) $R = g^r$, which is used to provide a unique signature. The challenge $c = H(X, R, m)$ is then obtained by applying the hash function H to the concatenation of the public key Y, the nonce R, and the message to be signed. Finally, the signature is calculated as $s = r + cx$, and the pair (R, s) is used as the signature.
- **Verification**:
  The validity of a signature can be verified by checking if $g^s = RY^c$.

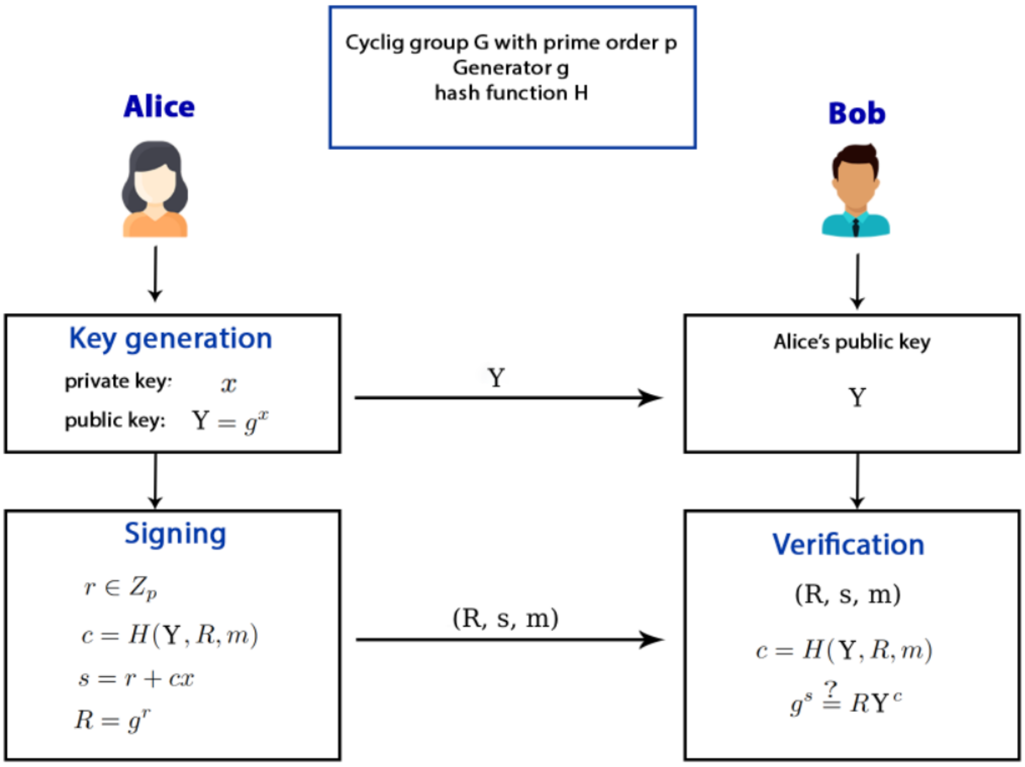


Figure 2.2: Schnorr Signature with Alice and Bob

2.2.2 EdDSA

EdDSA (Edwards-curve Digital Signature Algorithm) is a digital signature scheme based on elliptic curve cryptography. It was developed by Bernstein et al. [13], and has since become widely used in various cryptographic applications.

Here is an overview of how EdDSA works:

- **Key Generation**:
  A private key, x, is randomly generated from a secure random number generator. A public key, Y, is then derived from the private key by scalar multiplication with a base point on the chosen elliptic curve: $Y = [x] \cdot G$, where $([x] \cdot G)$ denotes the scalar multiplication of G with x, where G is the base point on the elliptic curve.

- **Signing**:
  To sign a message, the signer computes a random value, r, using a secure random number generator. A nonce, R, is then computed as the scalar multiplication of $R = [r] \cdot G$. The signer then computes a hash of the message and the nonce, $H(m, R)$. A scalar value, S, is then calculated as $S = r + H(m, R) \cdot x$. The signature is then the pair (R, S).

- **Verification**:
  To verify a signature, the verifier first checks that the public key, Y, is valid by ensuring that it lies on the elliptic curve and is not the point at infinity. The verifier then computes the hash of the message and the nonce, $H(m, R)$, and computes a scalar value, u, as $u = H(m, R) \cdot Y + S \cdot G$. The signature is valid if the point u equals R.

An example of a curve that can be used with EdDSA is curve25519, defined as a prime number $2^{255} - 19$ and using the base point $x = 9$, Curve25519 offers 128-bit security [14].

2.2.3 RSA Signature

RSA (Rivest-Shamir-Adleman) signing by Rivest et al. [15] is used to generate a signature for a message using a private key, which can be verified using the corresponding public key.

Here is an overview of the RSA signing protocol:

- **Key Generation**:
  A private key, d, is generated by selecting two large prime numbers, p, q, the modulus is calculated $n = p \cdot q$ and then computing the totient $d = (p-1)(q-1)$. A public key, e, is then generated by selecting a value that is co-prime to d, typically a small odd integer such as 65537. The pair (n, e) is the public key, and (n, d) is the private key.

- **Message Padding**:
  The message to be signed is first padded to ensure that it is the same length as the RSA modulus n. Various padding schemes, such as PKCS#1 v1.5 or RSA-PSS, can be used.

- **Signature Generation**:
  To generate a signature for the message, the signer first computes a hash value of the padded message using a secure hash function, such as SHA-512. The hash value, h, is then encrypted using the private key, d, to produce the signature, $s : s = h^d \ mod \ n$.

- **Verification**:
  To verify the signature, the verifier first decrypts the signature using the public key, $e : h' = s^e \ mod \ n$. The verifier then computes a hash value of the original message using the same hash function as the signer and compares it to h'. If they are equal, then the signature is valid.

In February 2023, the National Institute of Standards and Technology, NIST, published the Digital Signature Standard (DSS) [16]. Of the algorithms specified in that document, two of them, EdDSA and ECDSA, are variants of the Schnorr digital signature, and the last one is based on RSA. The recommended bit length for modulus in RSA is currently 2048 bits. Meanwhile, the EdDSA and ECDSA's recommended bit lengths are significantly lower (greater than 256 bits).

2.3 Extended Properties

Several modifications can be implemented to better manage the private key in vehicular authentication to increase the security of the private key, strengthen redundancy, and avoid a single point of failure. As such, these properties can be used to strengthen the scheme and are used together with the signature schemes described previously.

2.3.1 Interactive and Non-Interactive

In an interactive protocol, the interaction between participants is allowed [17]. Said interaction is considered to be messages sent over a communication channel. Non-interactive protocols, on the other hand, only allow a certain participant called a dealer to communicate with the rest of the participants.

2.3.2 Verifiable secret sharing

Verifiable secret sharing (VSS) is one of the fundamental ways of ensuring that the secrets shared by the dealer in a threshold cryptographic scheme are indeed valid shares of the bigger secret. Something that was introduced primarily by Chor et al. in [18].

Feldman [19] introduces a noninteractive VSS which widens the applicability of VSS to scenarios in which interaction is infeasible, such as sharing a secret among an entire nation. Also, several executions of noninteractive protocols may be run in parallel. On the contrary, interactive schemes may have to run serially. Thus, noninteraction allows us to use VSS as a subroutine without increasing the round complexity.

To facilitate the verification of the distributed shares, Feldman [19] [20] proposes a scheme where the dealer also broadcasts a public commitment vector. Feldman's (t, n) threshold VSS scheme, where t is the threshold and n is the total number of participants, for secret s, $s \in \mathbb{Z}_g$ is as follows:

- Let g be of a group of order p (p is a large prime).

- Dealer chooses a random polynomial: $a(X) = s + u_1X + ... + u_tX^t$ where $u_j \in_{\mathbb{R}} \mathbb{Z}_p, 1 \leq j \leq t$.
- The dealer sends the secret shares $s_i = a(i)$ to participants P_i in private for $i = 1, ..., n$.
- The dealer also **Broadcasts** commitments $B_j = g^{u_j}, 0 \leq j \leq t$
- When Participant P_i receives their share s_i, they are able to verify their share with the commitment: $g^{s_i} = \prod_{j=0}^{t} B_j^{i_j}$

Reconstruction:

- Each participant's share (P_i) is verified with $g^{s_i} = \prod_{j=0}^{t} B_j^{i_j}$
- The secret is then recoverable with Shamir's Scheme such that $s = a(0)$

Pederson's VSS scheme [17] works similarly to Feldman's. However, it is different from Feldman's scheme in that it does not assume that the a priori distribution of s used by the dealer is the uniform distribution on $\mathbf{Z}_n$, and instead ensures that the security of the secret is not dependent on the a priori distribution of s. That said, the major difference is the commitments.

2.3.3 Publicly Verifiable Secret Sharing

VSS enables participants to verify the validity of their own shares but they do not know if other participants in the protocols also received valid shares. If an ill-intended dealer distributes invalid shares to a subset of participants and these participants accept those shares, the reconstruction of the original secret will fail. Publicly verifiable secret sharing, PVSS, is defined informally as the ability to verify the validity of any other share in addition to the participant's own share [21].

Assume a public encryption function E_i with respective decryption function D_i where only the corresponding participant i knows D_i. The set of all n participants is denoted by L. In a PVSS scheme, each share is encrypted before being distributed by the dealer:

$$S_i = E_i(s_i)\ ,\ i \in L.$$

Verification of all the encrypted shares is carried out using an algorithm *PubVerify* with the following property:

$$\exists\ u\ \forall\ i \in L : (PubVerify(\{S_i \mid i \in L\}) = 1) \implies Recover(\{D_i(S_i) \mid i \in L\}) = u$$

Simply put, if the function *PubVerify* validates the validity of a set of encrypted shares to be good, then honest participants can decrypt them and recover the secret. One requirement in *PubVerify* is that it must be executable even if the participants have not received their shares.

2.3.4 Accountability

With accountability, one can ensure with threshold signature that the shares are used to sign the message, something that helps to know which devices were active in the signature process. This way, it enables the message receiver to identify whether some devices are compromised or not. A protocol is suggested by Boneh et al. [22] that uses an accountable signature and a private signature, which is called a threshold, accountable, and private signature (TAPS). During the key generation phase, they also create a tracing key, which helps trace the signature back to the quorum that generated it. The TAPS protocol achieves the private threshold signature by giving the tracing key to only trusted parties. If the tracing key goes public, the protocol becomes a "normal" accountable threshold signature scheme, whilst destroying the key would simply make it a private threshold signature scheme. Keeping the signature private and accountable simultaneously would require that the trace key remain in the possession of the trusted parties [22].

2.3.5 Proactive Secret Sharing

Ensuring that key shares remain secret forever in a long-lived system is not realistic, as nodes can become compromised or fail over time, potentially exposing secret key shares. This gives the possible adversary more time to collect shares and recover the secret. Proactive secret sharing schemes (PSS) [23] address this issue by using a secret sharing regeneration protocol, which allows one to keep the main secret the same while refreshing the shares of trusted devices. This renders old key shares useless, and mobile adversaries trying to compromise the system for a longer period than the refreshing of shares will not be able to meet the threshold t to reconstruct the secret.

Refined PSS schemes allow for the renewal of existing shares and the distribution of new shares to different signers, meaning that new members are able to hold a share of the secret. This can be seen as a redistribution of shares instead of renewals. By periodically updating the shared secret, parties can ensure that the data remains secure even if one or more parties become compromised [8].

Here is how a proactive secret sharing scheme works [23]:

- Proactive secret sharing involves periodically updating the shares of a secret among a set of n participants. Assume that there is a (t, n) secret sharing scheme and that each participant has a valid share.
- At regular intervals of T time units, a subset of α participants is selected from the set of n participants.
- Each selected participant constructs a random polynomial $f_i(x)$ of degree $t-1$ such that $f_i(0)$ is the participant's current share of the secret.
  $f_i(x) = s_i + a_{i,1} + \ldots + a_{i,t-1}$
- The coefficients of $f_i(x)$, $a_{i,0}, a_{i,1}, \ldots, a_{i,t-1}$, are chosen randomly from $\mathbf{Z}_p$, where p is a large prime number.

- Each selected participant sends its share of the updated secret, which is the value of $f_i(j)$ for $j = 1, 2, \ldots, n$, to all other correct participants in the system.
- Each nonselected node updates its share of the secret by computing $u_{j,i} = f_j(i)$ for all j in the subset of selected nodes and then updating its share as $x_i^{t+1} = x_i^t + u_{1,i}^t + u_{2,i}^t + \cdots + u_{t,i}^t$, where t is the time interval in which the shares are valid.
- All nodes store the updated shares and use them to reconstruct the secret key whenever necessary.

Secret reconstruction: The secret key can be reconstructed using the same procedure as in the non-proactive secret sharing scheme like Shamir's [10].

All of the non-updated shares an attacker accumulated become useless. An attacker can only recover the secret if they can find enough other non-updated shares to reach the threshold. This situation should not happen because the participants delete their old shares. The dealer can change the threshold number while distributing updates, but must always remain vigilant of participants keeping expired shares. However, this is a somewhat limited view since the original method gives the community of servers the ability to be the re-sharing dealer and the regenerator of lost shares.

2.3.6 Distributed Key Generation

Distributed key generation (DKG) allows participants $P = \{P_1, ..., P_n\}$ to generate a public and secret key together. This excludes the need for any dealer to be trusted. Something that was originally proposed by Pederson [24]. In this scheme, each participant P_i creates their own secret s_i and then creates the corresponding shares for all other participants. The scheme requires two rounds:

- **Public communication round**:
  Each party broadcasts a commitment to s_i and the coefficients of the corresponding polynomial.
- **Secure communication round**:
  Secret shares of s_i are securely sent to all other participants by each participant P_i. After receiving a share from P_i, each participant checks if the received share is consistent with the previously published commitment. The received share is designated as qualified if this is the case. By adding up all the qualified shares that each participant has received, they are able to determine their total share. Any party does not compute the secret shared value s itself: however, it is equal to the sum of the shared s_i, which passed commitment checks by all recipients of the shares.

2.4 TLS in vehicles

TLS is a cryptographic protocol that enables secure communication over a network. It is commonly used to secure web browsing sessions and email communications, among other things. TLS ensures the confidentiality, integrity and authentication of data exchanged between two parties using encryption algorithms and digital certificates.

An example use case is when a user connects to a website using HTTPS (HTTP Secure), the web server and the user's browser initiate a TLS handshake to establish a secure connection. During the handshake, the server presents a digital certificate that verifies its identity to the user's browser. The browser then uses public key cryptography to authenticate the certificate and establish a secure session key for the connection. This session key is used to encrypt and decrypt data exchanged between the server and the browser, ensuring that the data remains confidential and secure.

There are mainly two TLS versions used today, version 1.2 and version 1.3. The TLS 1.3 protocol provides several improvements over TLS 1.2, such as reduced latency and improved security. One significant change is the replacement of several older cryptographic algorithms with newer, more secure ones. For instance, TLS 1.3 no longer supports the RSA key exchange algorithm and instead relies on Diffie-Hellman key exchange, which is more secure against attacks.

Various ECUs are responsible for controlling various functions, such as the engine, transmission, and infotainment system. These ECUs need to communicate with each other to ensure that the vehicle operates correctly. Therefore, it is vital to establish a secure internal communication protocol that can transmit messages between ECUs safely. To achieve this, Zelle et al. [25] propose using TLS for internal vehicle communication. TLS can provide authentication, confidentiality, and integrity for data exchanged between ECUs, ensuring that messages are secure and trustworthy. Implementing TLS in internal vehicle communication can protect against potential attacks and ensure the safety and reliability of the vehicle. However, implementing TLS in-vehicle communication systems can be challenging due to the limited resources available in ECUs.

3 Related Work

The threshold signature concept has been around for a long time [26] and has gained much traction along with the cryptocurrency uprising. This chapter looks closely at some publications that are closely related to and inspired this thesis work.

In multiparty computation, having as few communication rounds between participants as possible is desirable because these interactive rounds are inefficient and costly in terms of time and network bandwidth. To our knowledge, two-round communication is still the lower limit for Schnorr-based multiparty computed digital signature schemes. In distributed signing, the aggregated signature includes randomness from each signing party for security reasons. Hence, it takes at least one round of communication for nonce exchange and one other round for the actual signing of the message.

Constructing multi-party Schnorr-based signing schemes with optimal communication efficiency is not trivial. Although, there have been several multisignature schemes that could archive it, for example, the preliminary version of the Musig [27] by Maxwell et al. or Bagherzadi et al. scheme [28]. The idea of doing multisignature in these schemes is to have the signing committee exchanges nonces, which are aggregated in later steps and then hashed to create a challenge c. This challenge c is embedded together with each signer's self-generated nonce and secret key to create its partial signature. An aggregated Schnorr-like signature is the sum of all the partial signatures. It can be verified just like a non-distributedly generated signature. The respective papers refer to more details and the actual algorithm of these schemes.

Unfortunately, these schemes have been proven flawed under normal discrete-logarithmic assumptions by Drijvers et al. [29]. Their work on the security of two-round multi-signature proves that all the known two-round Schnorr-based signature schemes are insecure in a parallel setting. Drijvers et al. build the attack around the adversary's ability to open concurrent signing sessions to collect other signers' commitments and manipulate the challenge c to produce a valid signature. The challenge c is a number that is mapped onto group $\mathbb{G}$ from a hash of the public key Y, the commitment R, and the message m. With the commitment and message embedded in the hash, falsifying only the message would create a different output than the valid hash, thus failing the challenge calculation and, eventually, verifying the signature. But if the adversary manipulates the commitment and hashes it with the forged message, they can potentially generate a hash with the same mapping as if the hash was genuine.

Attacking the hash function itself is computationally impractical and borderline infeasible. But attacking the hash function mapping over to the cyclic group can be reduced to subexponential complexity with the help of Wagner's algorithm [30]. Using Wagner's technique, the adversary can find a commitment which, together with the falsified message, generates the same challenge as a legitimate combination of commitment and message. This commitment is then sent to the honest signers, who are tricked into signing a fraudulent message. The final step of the attack is for the adversary to collect all the necessary partial signatures, aggregate them, and produce a legitimate signature for the falsified message. The attack is explained by Drijvers et al. in their paper [29] or in the MuSig2 paper [6].

Even though Drijvers et al. propose an attack on multi-signature schemes, the same technique can be abused to attack threshold signature schemes with two rounds of communication. Komlo and Goldberg touched on this concern in their paper, proposing a solution for the vulnerability of existing two-round multiparty signing digital signatures. In their paper [7], they propose a simple two-round Schnorr (t, n) threshold signature scheme, FROST, that addresses the forgery attack under concurrent signing. Their scheme relies on the idea of using a linear combination of multiple nonces to achieve better parallelism for two-round signing schemes while avoiding breaking the security proof under the pure discrete-logarithmic assumption. The efficiency of FROST can be further increased by a pre-processing stage, which reduces the number of interactive rounds while signing to only one. FROST has a distributed key generation to generate a secret, which ensures that none of the participants singlehandedly manipulates the private key.

The way the proposed scheme VATS achieves the threshold property for a Schnorr-like digital signature scheme is based on Shamir's secret-sharing technique. But still, our scheme takes a lot of inspiration from FROST on how to thresholdize a Schnorr digital signature scheme. Regarding key generation algorithms, FROST proposes distributed key generation to avoid reliance on a trusted dealer. Meanwhile, VATS uses the traditional verifiable secret-sharing scheme proposed by Feldman [19] because of the assumption that the share generation or reinitialization is only carried out by authorised entities. The dealer in the secret-sharing scheme for the setting is considered to be trusted. In addition, the private key, which is the secret to be thresholdized, is predetermined and, therefore, cannot be easily randomly generated by a distributed key generation protocol.

Based on FROST, ICE-FROST [8] is another threshold signature scheme which is proposed to add robustness to FROST's key generation. The authors define robustness as the property that finishes the execution of the protocol once it is started. FROST's key generation lacks robustness, making it unable to create and distribute the shares if any participants are corrupted. ICE-FROST prevents this by a mechanism called *identifiable cheating entity*, which enables participants to identify and exclude rogue participants from the key generation phase. Another worth mentioning property, which ICE-FROST offers over FROST, is proactivization. Protactivization enables refreshing distributed shares while keeping the original private key unchanged. ICE-FROST implements this property by having participants accumulatively change the random polynomial without either revealing or changing the original secret. The key-update procedure proposed in this work takes inspiration from this idea.

FROST and ICE-FROST consider digital signatures in a threshold setting. The same logic of using a linear combination of nonces can also be applied to multisignature schemes to reduce a communication round, as shown in the MuSig2 signature scheme [6]. Multisignature can be seen as a particular case of threshold signature where the threshold t equals the number of participants n. For this reason, MuSig2 is better optimised to handle the case (n, n) compared to the threshold signature (n, n) that FROST can provide.

FROST, ICE-FROST, and MuSig2 all have in common the ability to partially do the signing operation offline. The preprocessing phase that FROST and ICE-FROST have is still an interactive protocol, but it can be done even before knowing the message to sign and actually signing said message. This speeds up the actual signing phase and improves other Schnorr digital signature schemes prior to their work. Though the MuSig2 scheme is not explicitly divided into a preprocessing phase and a signing phase, the authors touched on the ability to partially do the signing offline, just like in FROST and ICE-FROST. VATS inherits this ability from MuSig2.

One of the important properties that VATS achieves is having the accountability of the signing committee. In threshold signature schemes such as FROST and ICE-FROST, accountability is not included since those schemes enjoy the anonymity of the committee. In other words, different signing committees produce the same signature once the required signer's threshold is passed. Being a multisignature scheme, MuSig2 guarantees that all committee signers participate and sign the message. VATS achieves accountability along with the threshold property by thresholding MuSig2. The digital signature produced by VATS proves the knowledge of the private key of the signing committee, as in threshold signature schemes, as well as the identity of each signer, similar to multi-signature schemes.

4
Methods

To achieve the goals set out in Section 1.4, this project was carried out in two main phases: research and development. This chapter gives an overview of what and why.

4.1 Research Methodology

Before seeking solutions, it is crucial to understand the problem clearly. Therefore, the project began with a comprehensive literature review to better understand the problem. Smart vehicles are rising, significantly changing how internal and external vehicle communication is handled [31]. Therefore, considerable time was spent on this topic to understand the current and future landscape better. Initial research confirmed the need for better redundancy for the private key used in V2X communication, and the project was deemed potentially significant for industry and academia.

When searching for a cryptographic solution to the problem, the idea of distributing the private key among many parties was of interest. Consequently, many different secret-sharing schemes were studied, focussing on Shamir secret sharing, the fundamental technique that enables threshold signatures. A broad survey of threshold signature schemes was also conducted to gain an idea of the existing techniques and their pros and cons. This helps narrow the scope of this thesis, accelerating the research phase. From the initial research on threshold signature schemes, it was observed that, among EdDSA, RSA, and Schnorr digital signature schemes, Schnorr schemes have some desirable properties for a vehicular setting. Therefore, an in-depth study of threshold signature schemes that produce Schnorr or Schnorr-like signatures was performed.

A threshold signature scheme can have many different properties, such as identifiable cheater, accountability, and verifiability of the signers. The selection of the features that a protocol should have to be the most suitable and beneficial for V2X communication is based on the supervisors' recommendations in the company where the scheme is developed. Among these properties, some were essential and must-haves, while others were less important. The decision on whether to include these non-essential protocols depended on the possible extra complexity they would have added to the signature schemes and hardware constraints.

With a clear view of all the properties and features (shown in Table 1.1) that threshold schemes should have, the existing protocols studied were re-evaluated again. The most suitable schemes were then selected to be further developed. This concludes the review of the literature for this project.

4.2 Reference Architecture

Since the protocol for this thesis is intended for use in a vehicular environment, it is necessary that the vehicle specifications are suitable for such an implementation. In addition, depending on the car's internal network communication, some of the communication might be too limiting for the protocol to be used effectively. Therefore, it would be beneficial to work with a predefined internal network, allowing the implementation to be adapted to some reference vehicles.

The electrical system architectures of vehicles vary greatly depending on the manufacturer, type, and even configuration. As there is no generic architecture, we consider an abstract architecture with different domains. ECUs are separated into distinct domains based on their functionality. For instance, a powertrain domain can contain ECUs responsible for engine control and gear shift, while the infotainment domain provides functions such as a radio or navigation system [32].

For the sake of simplicity, an assumption like that is illustrated in figure 4.1. The vehicle consists of multiple local ECUs capable of executing minor tasks to reduce the load on the central ECU. The domain ECU manages all local ECUs in a particular section of the vehicle. Each domain ECU is linked to the central communication ECU, which provides external connectivity through the Telematics Control Unit (TCU). The TCU establishes connections with other vehicles and infrastructure using V2X communication and the Internet through cellular connections. These domain ECUs could serve as shareholders (and signers) in a potential threshold signature scheme, considering their direct connection to the central ECU, minimising the chance for faulty ECUs in between the connection to the central ECU.

For intra-vehicle communications, CAN with a bandwidth capacity of around 1 MB/s [33] is traditionally used. But in modern vehicles, many other networks are integrated and used to enable communication between ECUs. One of them is Ethernet, which offers much higher capacity (up to 1 GB/s). For this project, it is assumed that all connections are implemented using automotive Ethernet. Ethernet is beneficial since it can handle network communication protocols like TCP, which can be wrapped with a security protocol [25].

4.3 Implementation

Some protocols work in theory but are impossible to implement or impractical to use because of certain constraints. Considering the goal of this project, which was to have a practical scheme suitable for automotive applications, the scheme needed to be implemented. A functioning implementation is crucial to evaluating the performance and usability of a developed protocol. Considering the lack of a simulated

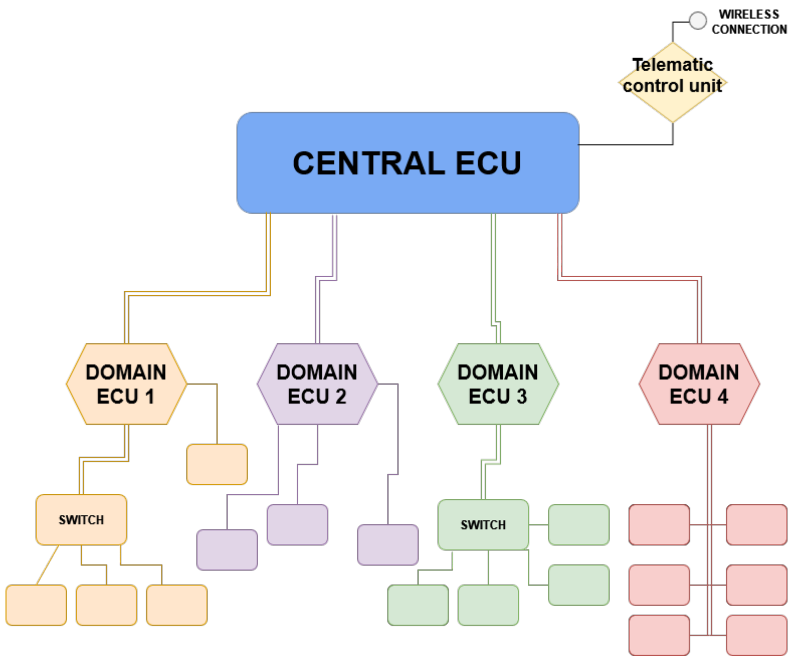


Figure 4.1: Internal car architecture

environment compared to an actual vehicle, it was decided to implement the scheme, including network communication capabilities. Therefore, it was possible to simulate an actual implementation as accurately as possible. By releasing the code publicly anyone interested is able to contribute, add, or take inspiration from the code.

4.3.1 Programming Language

Rust is becoming increasingly popular for cryptography, as it offers secure memory management, reducing the risk of buffer overflows and other memory-related vulnerabilities and making it safer than other high-performance languages. Rust is designed to support concurrent and parallel computations and can run on most modern hardware. Rust's design allows it to map directly to the hardware, making it an efficient programming language because it can control memory representations and support stack allocation and contiguous record storage directly [34]. Several well-built and well-maintained cryptographic libraries are available in Rust. Therefore, it was chosen when implementing a proof-of-concept VATS scheme.

4.3.2 Libraries

Implementing a cryptographical scheme requires algebraic operations performed in a finite field. To accomplish this task, **Ed25519-dalek** [35] was chosen. It is a fast and efficient Rust implementation of ed25519 key generation, signing, and verification. A proof-of-concept implementation of the project scheme is implemented in Rust using Risttreto over curve25519 as the group operations. Therefore, it makes significant use of the Ed25519-dalek library. The Dalek Library is frequently used for cryptographic schemes and is used in implementations of FROST and ICE-FROST, Ed25519-daleks implementation of correctness, safety, and clarity as a priority and a secondary goal of performance makes it a very suitable library for a scheme like VATS. Since VATS implementation is created to be a secure signature scheme but also feasible and measured to be practical, it requires security and performance.

To effectively handle asynchronicity in communications between network participants, **Tokio** was used. Tokio [36] is an asynchronous runtime for the Rust programming language that provides the functions necessary to write networking applications. It is flexible and can target various systems, from large servers with dozens of cores to small embedded devices such as vehicle ECUs. Furthermore, Tokio helps to ensure that a functional network interface is present in the vehicle and adaptable to the internal network infrastructure of the vehicle (Fig. 4.1). At a high level, Tokio provides a few major components that make networking efficient and possible.

- A multithreaded runtime to execute asynchronous code.
- An asynchronous version of the standard library.
- A large ecosystem of libraries.

Also, as part of the underlining network to test the implementation of VATS, **Warp** was used to enable communication between participants. Warp [37] is a composable web server framework for creating a fast and reliable server and client communication. Warp is also adaptable to be used with TLS, where the user can provide locally stored keys and signatures. Since secure communication will be essential to the ECU's internal communication, Warp with TLS enabled helps ensure that such communication is secure against man-in-the-middle attacks. In addition, Warp can use custom input certificates in its TLS communication, making it easy to provide it with a certificate authority made locally for testing and official certificate authorities used in real-world implementations.

4.4 Testing

This project evaluates the practicality of implementing the scheme (VATS) in a vehicular setting with an automotive application in mind. The protocol's functionality is tested in a simulated environment where the vehicle's internal communication between different ECUs and external communication with cloud instances can be simulated.

To simulate this environment, **Docker** creates Docker containers to run the application as an independent entity, just like an ECU in a vehicle. The advantage of using Docker is the ability to port applications to different machines and environments [38]. Each Docker container can be considered a possible ECU, and the communication between each Docker is similar to that of Ethernet connections in a vehicle.

The scheme's feasibility is evaluated through various measurements, such as execution time, data complexity, and memory complexity. Multiple graphs are presented to show efficiency in vehicle architecture that allows greater or lesser numbers of

ECUs, providing a perspective and allowing for discussion of the scheme's practicality. These metrics are desirable to analyse:

- Execution Time: The time the scheme took to perform various operations, such as key generation, signing, and verification, was measured. This metric provides information on the speed and responsiveness of the scheme in real-world scenarios.
- Resource Utilisation: The utilisation of system resources, including data complexity and memory complexity was analysed in the implementation of the scheme. Understanding the scheme's resource requirements helps evaluate its efficiency and potential impact on system performance.
- Scalability: The scalability of the scheme was examined, considering an increasing number of participants or increasing data size. Scalability assessment is crucial to determine whether the scheme can handle larger workloads or accommodate a growing number of users without significant performance degradation. However, it is not the primary consideration during the scheme's development because the vehicle has quite a few signers, according to the architecture in Section 4.2.

The idea is to benchmark the performance of the scheme so that the evaluation gives an estimate of the scheme's performance in a real-world application. These metrics and the choices for the number of participants (n) and the threshold (t) are inspired by other papers that perform similar measurements, such as ICE-FROST [8]. However, ICE-FROST only benchmarked its key generation and signing. Whereas, in the case of VATS, benchmarking is intended to be conducted for several crucial components individually, therefore, having a more transparent performance evaluation.

5

Scheme

This chapter shows in detail the preliminaries and description of VATS. A pseudocode is also provided for clarity and ease of implementation.

5.1 Preliminaries

Security parameter The security parameter is denoted κ, and is borrowed from the MuSig2 papers [6]. Nick et al. defined their security parameter as the result of one of their proofs and is described as:

> Sampling an element of a sampleable set S uniformly at random and assigning it to s can be denoted as $s \leftarrow^{\$} S$. For a randomised algorithm A, the operation of running A on inputs $x_1, \ldots$ and random coins ρ, and assigning its output to y, can be represented as $y \leftarrow A(x_1, \ldots; \rho)$. If random coins ρ are chosen uniformly at random, the notation $y \leftarrow A(x_1, \ldots)$ can be used instead. Given a $Game_A$ parameterized by the adversary A, we define the advantage of A in $Game_A$ as $Adv_A^{Game}(\kappa) := Pr[Game_A(\kappa) = true]$

How exactly these proof games work is given in the MuSig2 paper [6].

Group Description. A group description is a triple $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of order p and g is a generator of G. A group generation algorithm, denoted by GrGen, is a prime-order algorithm that takes a security parameter κ as input and returns a group description $(\mathbb{G}, p, g)$, where p is a prime number of κ bits. In this group, the group operation is multiplication, and group elements and their encodings are equivalent when passed as input to hash functions.

Given an element $X \in \mathbb{G}$, we use $\log_g(X)$ to denote the discrete logarithm of X in base g. In other words, $\log_g(X)$ is the unique integer $x \in \mathbb{Z}_p$ such that $X = g^x$.

Selection of v. Choosing the number of v, the number of nonces to generate, can affect the scheme's performance. So, it is beneficial to select a v as small as possible, but choosing v too small risks the protocol's security, as proven in Section 6.2.3. In their paper, Nick et al. [6] mention that having $[v = 2]$ saves multiexponentiation of size three plus single exponentiation, as well as three group elements of communication in the first round (all per signer). They proved in the reduction from the random oracle model that $[v = 4]$ is secure, and in the random oracle model with the algebraic group model that MuSig2[$v = 2$] is secure. Nick et al. state that they could not find evidence of choosing [$v = 2$] to be insecure and that it yields the best performance if one can accept the weaker proof (since the proof with an algebraic group model is strictly weaker than the generic group model, since it places fewer restrictions on the proof).

Committee Selection. Choosing the proper committee to sign a document can significantly improve the signing process's efficiency. However, limiting the pool of possible signers can be considered wasteful, and excluding specific signers could lead to confusion about their ability to produce a valid signature. In this protocol, the committee is randomly selected based on the credibility of the signers to generate a correct signature. If specific signers do not sign, they lose credibility. To minimise communication traffic, the signing aggregator subjectively assesses each signer's credibility, which is not shared with others.

5.2 Scheme Description

In this section, we introduce the detailed description and pseudocode for the components in the proposed scheme VATS. The scheme needs n shareholders and a threshold t, where the algorithm is made such that it is from the point of view of the participant P_i, where $1 \leq i \leq n$.

5.2.1 Setup

In the setup with input 1^κ, the setup algorithm runs $(\mathbb{G}, p, g) \leftarrow GroupGen(1^\kappa)$, selects four hash functions $(H_{agg}, H_{non}, H_{sig}, H_{upd})$ from $\{0,1\}^*$ to $\mathbb{Z}_p$, and returns $par := ((\mathbb{G}, p, g), H_{agg}, H_{non}, H_{sig}, H_{upd})$.

5.2.2 Key Generation (KeyGen)

Share Generation:
A trusted dealer takes in two parameters, the number of signers n and a threshold t, and samples t random values such that $(a_1, ..., a_{(t-1)}) \xleftarrow{\$} \mathbb{Z}_p^t$. The dealer will also define the global public key as $Y = g^{a_0} \in \mathbb{G}$; this key will be the one certified by the OEM. Each participant will receive their share of the key depending on their index x, so $f(x)$ will be sent to the participant x, similar to Shamir's secret sharing scheme [10]. The dealer then creates commitments $B := (g^{a_0}, ..., g^{a_{t-1}})$ that are broadcasted. With the help of Shamir's secret sharing scheme in this step, Threshold Tolerance is acquired, and with the commitments, VSS is achievable.

Share Verification (ShareVer):
Share verification helps ensure that each participant x receives their rightful share. This is done with the help of Feldman's verifiable secret sharing [19], where the participant P_i can check with $g^{s_i} \overset{?}{=} \prod_{j=0}^{t-1} B_j^{ij}$; then they can be sure that the share is indeed the right one. Setting $g^{s_i} = Y_x$, participant P_i gets their personal

public key and calculates the global public key $B_0 = Y$. The verification of shares in this step can also be used for PVSS, since it can be calculated for anyone with a public key.

An illustration of the key generation protocol is shown in figure 5.1.

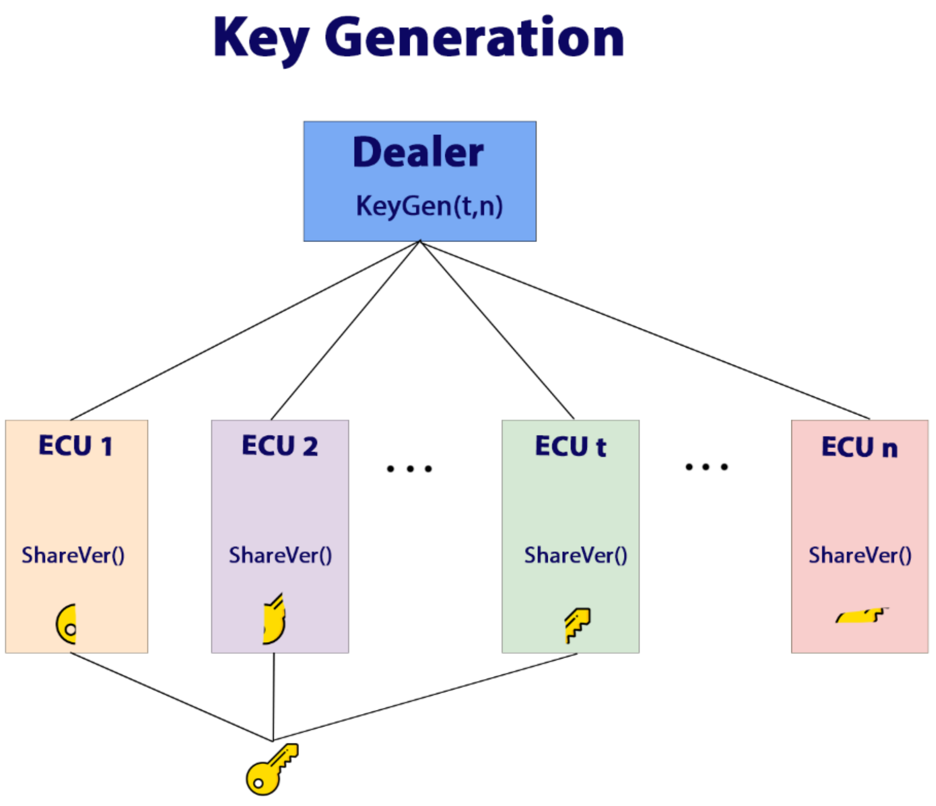


Figure 5.1: An illustration of (t, n) key generation and distribution scheme.

5.2.3 Committee selection

The signing committee for each signing session contains at least t signers randomly selected from n participants. The signing aggregator keeps a record of how well each participant has performed in producing valid signatures. This performance metric is increased every time a participant successfully signs a message or decreased if the participant produces an incorrect signature. Participants with a higher performance metric have a better chance of being selected for the next signing session. In this way, we create a committee selection that is based on cheating scores. All participants have a chance to be selected and increase their score as long as they do not repeat producing incorrect partial signatures. It is possible for a participant to get excluded (due to their score being lowered) if they fail their signature frequently.

5.2.4 Key Aggregation (KeyAgg)

The public keys of the selected committee are denoted by $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$. To create the coefficient ρ_i of key aggregators using the committee L and the public key Y_x, the coefficient is obtained for the participant x, with ρ_i through $H_{agg}(L, Y_x)$. The corresponding aggregated key for L is then given by $\widetilde{Y} := \prod_{x=1}^{t} Y_x^{\rho_i} \in \mathbb{G}$. In this way, creating a public key for that selected committee is possible.

5.2.5 Offline Signing Phase (SignOff and SignAgg)

SignOff: Every committee member has the ability to perform the signing process offline and locally. Every participant creates $j \in \{1, .., v\}$ random values $r_{i,j} \xleftarrow{\$} \mathbb{Z}_p$ and calculate $R_{i,j} := g^{r_{i,j}} \in \mathbb{G}$. They then generate and provide v nonces, represented as $(R_{i,1}, ..., R_{i,v})$.

SignAgg: Assuming that there are t participants, the aggregator obtains the results $(R_{1,1}, \ldots, R_{1,v}), \ldots, (R_{t,1}, \ldots, R_{t,v})$ from all participants. The aggregator then aggregates the results by calculating $R_j = \prod_{x=1}^{t} R_{x,j}$ for each $j \in 1, \ldots, v$, and finally produces the output $(R_1, \ldots, R_v)$.

5.2.6 Online Signing Phase (SignOn, SignAgg2 and Sign)

Let Y_i and s_i be the public and private keys of a specific signer, respectively. Let m denote the message that must be signed, $\{Y_1, ..., Y_t\}$ be the public keys of the participants in the signing committee, and $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$ represent the multiset of all the public keys involved in the signing process.

SignOn: SignOn works as follows. First, the signer applies the key aggregation algorithm to compute $\widetilde{Y}$, and then stores its own key aggregation coefficient $\rho_x := H_{agg}(L, Y_i)$. Next, after receiving the aggregate first-round output $(\widetilde{R}_1, ..., \widetilde{R}_v)$, the signer computes $b := H_{non}(\widetilde{Y}, (\widetilde{R}_1, ..., \widetilde{R}_v), m)$.

The signer then combines the message with the nonces to prevent replayability by setting $\widetilde{R} := \prod_{j=1}^{v} R^{b^{j-1}} j \in \mathbb{G}$. The challenge is calculated as $c := H_{sig}(Y, \widetilde{R}, m)$, where Y represents the signer's public key. Additionally, the signer calculates $R_i := \prod_{j=1}^{v} R_{i,j}^{b^{j-1}} \in \mathbb{G}$, so that the signing aggregator can verify the partial signature.

To create their own unique signature, the signer computes their own Lagrange coefficient $\lambda_i = \prod_{j=1, i \neq j}^{t} \frac{j}{j-i}$ and then applies the formula: $z_i := c \cdot (s_i \cdot \rho_i + \lambda_i \cdot s_i) + \sum_{j=1}^{v} r_{i,j}^{b^{j-i}}$ *mod* p. Finally, the output of the signing algorithm is the following, $state_i' := \widetilde{R}$; $out_i' := (z_i, R_i)$.

SignAgg2: The aggregator receives the output $((z_1, R_1), ..., (z_t, R_t))$ of all signers and verifies them by checking $g^{z_x} \stackrel{?}{=} R_x Y_x^{c(\rho_x + \lambda_x)} \in \mathbb{G}$ for each signer, after reacreating the challenge $c := H_{sig}(Y, \widetilde{R}, m)$ and the individual $\rho_x := H_{agg}(L, Y_x)$ for each committee member. This check helps validate the partial signature of the signer x, thus identifying any cheating (providing individual accountability). Then, the signing aggregator aggregates the signatures by summing $z := \sum_{x=1}^{t} z_x$ *mod* p.

Sign: The signer receives z and returns the signature $\sigma := (R, z)$.

An illustration of the signing procedure is shown in figure 5.2.

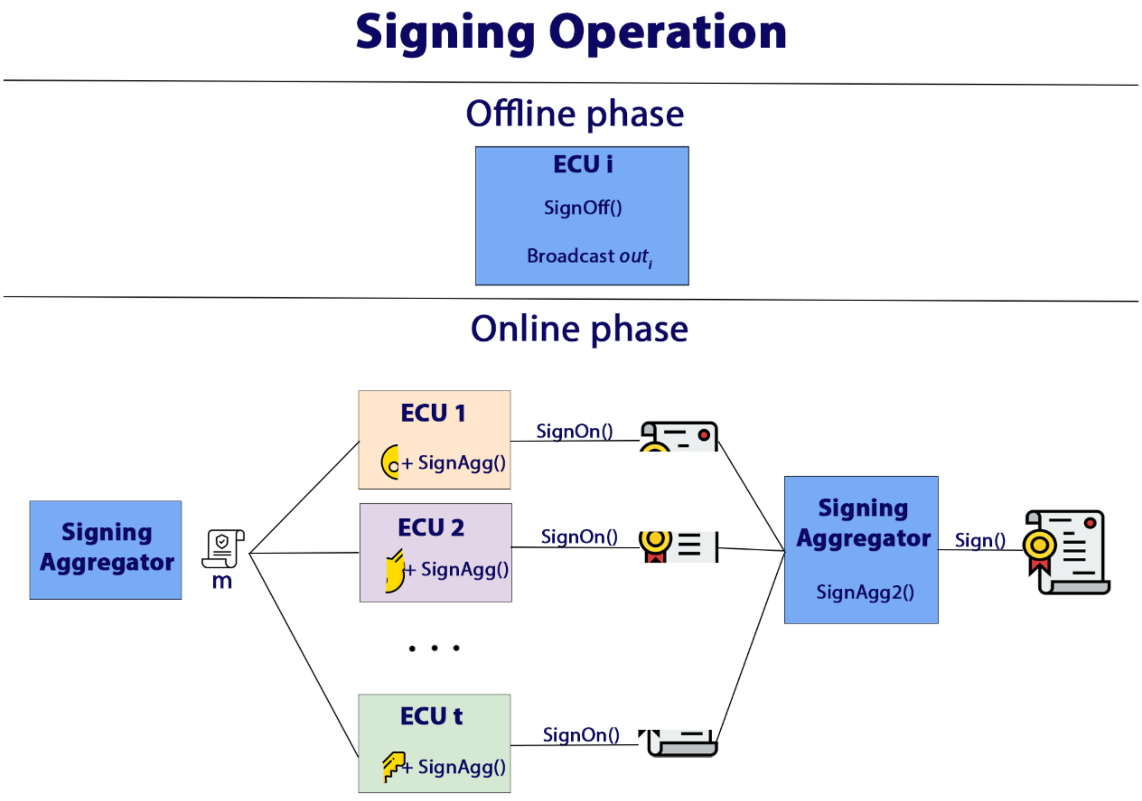


Figure 5.2: The signing procedure is separated into two phases, online and offline phase.

5.2.7 Verification (Ver)

Given the global public key, the multiset of all public keys involved in the committee $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$, the message m and $\sigma := (R, z)$ the verifier is able to calculate $\widetilde{Y}$ by aggregating the multiset of public keys. The verifier can also recreate the challenge by running $c := H_{sig}(Y, R, m)$. By adding Y and $\widetilde{Y}$ the verifier creates $Y' = \widetilde{Y} \cdot Y$ such that, by checking the signature, $g^z =^? R \cdot Y'^c$, the verifiers can be confident that the signature is correct for that specific committee. By having two keys, such that there is one public key for the committee $(\widetilde{Y})$ but also a public key for the original secret (Y) this scheme achieves accountability for that committee in verification.

5.2.8 Key Update (KeyUpd)

First Round: For each committee member chosen to participate in the key updating protocol, each participant samples t random values such that $(a_{i,0}, ..., a_{i,(t-1)}) \leftarrow^{\$} \mathbb{Z}$. Each participant in the committee calculates a polynomial with their secret $f_i(x) := s_i + \sum_{j=i}^{t-1} a_{i,j} x^j \ mod\ p$ for each n share. Now, each member must compute a proof of knowledge of the corresponding secret $a_{i,0}$. First, by creating a nonce:

$$k \leftarrow^{\$} \mathbb{Z}_p, R_i := g^k \in \mathbb{G}$$

followed by a hash with a context string Φ selected at random from the signing aggregator,

$$\hat{c}_i := H_{upd}(i, \Phi, g^{a_{i,0}}, R_i)$$

Then each participant makes $\hat{z}_i$,

$$\hat{z}_i := k + a_{i,0} \cdot \hat{c}_i \ \ mod\ p$$

and generates,

$$\sigma'_i := (R_i, \hat{z}_i)$$

now each participant generates $t-1$ commitments, creates $\overrightarrow{C}_i := (A_{i,0}, ..., A_{i,(t-1)})$ and broadcasts $\overrightarrow{C}_i, \sigma'_i$. Upon receiving $\overrightarrow{C}_x, \sigma_x$ from participants $1 \leq x \leq n,\ x \neq i$, the participant does the following validation for each received message:

$$(R_x, \hat{z}_x) := \sigma'_x$$

$$\hat{c}_x := H(x, \Phi,\ A_{x,0}, R_x)$$

aborting on failure by checking:

$$R_x =^? g^{\hat{z}_x} \cdot A_{x,0}^{-\hat{c}_x}$$

Upon success, each participant deletes $\{\sigma'_x : 1 \leq x \leq n\}$

Second Round: Each participant in the committee sends P_x: $(x, f_i(x))$ to the other $n-1$ participants. Upon receiving $(i, f_x(i))$, P_i verifies their shares by calculating $g^{f_x(i)} =^? \prod_{k=0}^{t-1} A_{xk}^{x^k}\ mod\ p$. The participant P_1 then calculates $s_i := \sum_{x=1}^{n} f_x(i) \cdot \lambda_x\ mod\ p$. where λ_i is the Lagrange coefficient corresponding to the participant P_i. Then the participant P_i stores s_i securely and deletes each $f_x(i)$. $Y_i := g^{s_i} \in \mathbb{G}$. Each participant can get their public key by calculating $Y := \prod_{j=1}^{n} Y_j^{\lambda_j} \in \mathbb{G}$. Moreover, any participant can calculate the public verification share of any other participant by calculating $Y_x := \prod_{j=1}^{n}(\prod_{k=0}^{t-1} A_{jk}^{x^k})^{\lambda_j}\ mod\ p \in \mathbb{G}$. The key updating protocol ends with the outputs $sk_i := s_i,\ pk_i := Y_i$.

With the help of KeyUpd the scheme achieves PSS, such that new shares are renewed at decided intervals.

5.3 Pseudocode

This section presents pseudocode for the proposed scheme.

Algorithm 1: Setup (1^κ)

```
1 $(\mathbb{G}, p, g) \leftarrow GroupGen(1^\kappa)$
  // Select three hash functions:
2 $(H_{agg}, H_{non}, H_{sig}, H_{upd}) : \{0,1\}^* \rightarrow \mathbb{Z}_p$
3 $par := ((\mathbb{G}, p, g), H_{agg}, H_{non}, H_{sig}, H_{upd})$
4 return $(par)$
```

Algorithm 2: KeyGen(t, n)

```
   // Sample $t$ random values
1  $(a_1, ...., a_{(t-1)}) \xleftarrow{\$} \mathbb{Z}_p^t$
   // the secret is denoted by $s$
2  $a_0 = s$
   // Create a global public key
3  $Y = g^s \in \mathbb{G}$
   // Use the $t$ random values as coefficients to define a degree $t-1$
      polynomial, and index x of participants $(1 \leq x \leq n)$
4  for $x := 1..n$ do
5  |   $f(x) := \sum_{j=0}^{t-1} a_j x^j \quad mod\ p$
6  end
   // Dealer creates the commitments to be broadcasted
7  $B := (g^{a_0}, ..., g^{a_{t-1}}) \in \mathbb{G}$
   // Send $f(x)$ to each participant x, and broadcast B
8  return $(f(x), B)$
```

Algorithm 3: ShareVer($B := (g^{a_0}, ..., g^{a_{t-1}}), f(i)$)

```
   // Upon receiving the share, each participant $P_i$ verifies its
      validity by evaluating the following
1  $s_i := f(i)$
   // on failure (bad share), request a new share from the trusted
      dealer by checking
2  $g^{s_i} \ =^? \ \prod_{j=0}^{t-1} \ B_j^{i^j} \ \in \mathbb{G}$
3  $sk_i := s_i$
4  $pk_i := g^{s_i}$
5  $pk = B_0$
6  return $(sk_i, pk_i, pk)$
```

Algorithm 4: ComSelect()

```
// $t$ signers are selected from $n$ participants
1  $P := (P_0, ..., P_{n-1})$
2  $Scores = (score_0, ..., score_{n-1})$ // Participants with higher performance
      scores are more likely to be selected.
3  $L := (P_0, ..., P_{t-1}) \in P$
   // Start signing procedure with committee $L$
4  $signing\_successed = true$
5  for $j := 1..t,\ j \in L$ do
       // Evaluate performance based on partial signature verification
          in $SignAgg2$
6      if $P_j$'s partial signature verified then
7          $\mathrm{Score}_j = Score_j + 1$
8      end
9      else
10         $\mathrm{Score}_j = Score_j - 1$
11         $signing\_successed = false$
12     end
13 end
14 if $signing\_successed$ is false then
       // Restart ComSelect()
15 end
16 return $signature$
```

Algorithm 5: KeyAgg(L)

```
1  $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$
2  for $x := 1..t$ do
3      $\rho_x := H_{agg}(L, Y_x)$
4  end
5  return $\widetilde{Y} := \prod_{x=1}^{t} Y_x^{\rho_x} \in \mathbb{G}$
```

Algorithm 6: SignOff(v)

```
   // Each participant $P_i$ generates $v$ random nonces
1  for $j := 1..v$ do
2      $r_{i,j} \xleftarrow{\$} \mathbb{Z}_p$
3      $R_{i,j} := g^{r_{i,j}} \in \mathbb{G}$
4  end
5  $out_i := (R_{i,1}, ..., R_{i,v})$
6  $state_i := (r_{i,1}, ..., r_{i,v})$
7  return $(out_i, state_i)$
```

Algorithm 7: SignAgg($out_1, .., out_t$)

```
1  for $x := 1..t$ do
2      $(R_{x,1}, ..., R_{x,v}) := out_i$
3  end
4  for $j$ := 1..$v$ do
5      $R_j := \prod_{x=1}^{t} R_{x,j} \in \mathbb{G}$
6  end
7  $out = (\widetilde{R}_1, ..., \widetilde{R}_v) \in \mathbb{G}^v$
8  return $out$
```

Algorithm 8: SignOn$(state_i, out, sk_i, m, (pk_1, pk_2, ..., pk_t), out_i)$

```
// Signing must be called at most once per state.
// Assemble parameters
1  $(r_{i,1}, ..., r_{i,v}) := state_i$
2  $s_i := sk_i,\ Y_i := g^{s_i}$
3  $\{Y_1, ..., Y_t\} := (pk_1, ..., pk_t)$
4  $(\widetilde{R}_1, ..., \widetilde{R}_v) := out \in \mathbb{G}^v$
5  $(R_{i,1}, ..., R_{i,v}) := out_i \in \mathbb{G}^v$
6  $L := \{Y_1, ..., Y_t\}$
// Recreating keys public keys to match signing
7  $\rho_i := H_{agg}(L, Y_i)$
8  $\widetilde{Y} := KeyAgg(L)$
9  $b := H_{non}(\widetilde{Y}, (\widetilde{R}_1, ..., \widetilde{R}_v), m)$
10 $\widetilde{R} := \prod_{j=1}^{v} \widetilde{R}_j^{b^{j-1}} \in \mathbb{G}$
11 $c := H_{sig}(Y, \widetilde{R}, m)$
// Creating the Lagrange coefficient
12 $\lambda_i = \prod_{j=1, i \neq j}^{t} \frac{j}{j-1}$
13 $R_i := \prod_{j=1}^{v} R_{i,j}^{b^{j-1}} \in \mathbb{G}$
// Calculating partial signature locally
14 $z_i := c \cdot (s_i \cdot \rho_i + \lambda_i \cdot s_i) + \sum_{j=1}^{v} r_{i,j} \cdot b^{j-1}\ mod\ p$
15 $out'_i := (z_i, R_i)$
16 **return** $(\widetilde{R}, out'_i)$
```

Algorithm 9: SignAgg2$((out'_1, .., out'_t), L, m, \widetilde{R}, Y)$

```
1 $((z_1, R_1), ..., (z_t, R_t)) := (out'_1, ..., out'_t)$
2 $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$
  // Verification of partial signatures:
3 $c := H_{sig}(Y, \widetilde{R}, m)$
4 for $x := 1..t$ do
5     $\rho_x := H_{agg}(L, Y_x)$
      // on failure, reduce participants score and abort by checking
6     $g^{z_x} \ =^? \ R_x Y_x^{c(\rho_x + \lambda_x)} \ \in \mathbb{G}$
7 end
  // Summing the shares
8 $z := \sum_{x=1}^{t} z_x \ mod \ p$
9 return $(out' := z)$
```

Algorithm 10: Sign$(out', state'_1)$

```
1 $\widetilde{R} := state'_i; z := out'$
2 return $(\sigma := (\widetilde{R}, z))$
```

Algorithm 11: Ver(pk, L, m, σ)

```
1 $\{Y_1, ..., Y_t\} := L \in \mathbb{G}^t$
2 $Y := pk \in \mathbb{G}$
3 $(R, z) := \sigma \in \mathbb{G} \times \mathbb{Z}_p$
  // Accountability ensured by recalculating $\widetilde{Y}$
4 $\widetilde{Y} \ := KeyAgg(L)$
  // Recreate challenge for verification
5 $c := H_{sig}(Y, R, m)$
6 $Y' := \widetilde{Y} \cdot Y$
7 return $g^z =^? R \cdot Y'^c$
```

Algorithm 12: KeyUpd(L, sk_i)

```
// ----------Round 1--------------
// Run by each member of L
// Sample $t$ random values
1  $s_i := sk_i$
2  $(a_{i,0}, ...., a_{i,(t-1)}) \xleftarrow{\$} \mathbb{Z}$
3  for $x$ := $1..n$ do
4  |   $f_i(x) := s_i + \sum_{j=i}^{t-1} a_{i,j} x^j \ mod\ p$
5  end
// Computes a proof of knowledge to the corresponding secret $a_{i,0}$
6  $k \xleftarrow{\$} \mathbb{Z}_p$
7  $R_i := g^k \in \mathbb{G}$
// $\Phi$ is a context string to prevent replay attacks.
8  $\hat{c}_i := H_{upd}(i, \Phi, g^{a_{i,0}}, R_i)$
9  $\hat{z}_i := k + a_{i,0} \cdot \hat{c}_i \ \ mod\ p$
10 $\sigma'_i := (R_i, \hat{z}_i)$
// Compute public commitments
11 for $j := 0..(t-1)$ do
12 |   $A_{i,j} := g^{a_{i,j}} \in \mathbb{G}$
13 end
// commitment list
14 $\overrightarrow{C}_i := (A_{i,0}, ..., A_{i,(t-1)})$
// BroadCast $\overrightarrow{C}_i, \sigma'_i$
// ----------Verification of shares--------------
// Upon receiving $\overrightarrow{C}_x, \sigma_x$ from participants $1 \le x \le n, \ \ x \ne i$
15 for $x := 1..n, x \ne i$ do
16 |   $(R_x, \hat{z}_x) := \sigma'_x$
17 |   $\hat{c}_x := H(x, \Phi, \ A_{x,0}, R_x)$
   |   // aborting on failure, by checking
18 |   $R_x =^? g^{\hat{z}_x} \cdot A_{x,0}^{-\hat{c}_x}$
19 end
// Upon success, delete $\{\sigma'_x : 1 \le x \le n\}$
```

```
    // ----------Round 2--------------
    // $P_i$ sends
 1  for $x := 1..n, x \neq 1$ do
 2      Send $P_x$: $(x, f_i(x))$
 3  end
    // Upon receiving $(i, f_x(i))$, $P_i$ verifies their shares by
       calculating:
    // abort on failure, by checking
 4  $g^{f_x(i)} \ =^? \ \prod_{k=0}^{t-1} \ A_{xk}^{x^k} \ mod \ p$
    // $P_i$ calculates:
 5  $s_i \ := \ \sum_{x=1}^{n} \ f_x(i) \cdot \lambda_x \ mod \ p$
    // where $\lambda_i$ is the Lagrange coefficient corresponding to $P_i$.
 6  Then $P_i$ stores $s_i$ securely, and deletes each $f_x(i)$.
    // Calculate and store public key:
 7  $Y_i := g^{s_i} \ \in \mathbb{G}$
 8  $Y := \prod_{j=1}^{n} \ Y_j^{\lambda_j} \ \in \mathbb{G}$
    // Any participant can compute the public verification share of
       any other participant by calculating
 9  $Y_x := \prod_{j=1}^{n}(\prod_{k=0}^{t-1} \ A_{jk}^{x^k})^{\lambda_j} \ mod \ p \ \in \mathbb{G}$
10  $sk_i := s_i, \ pk_i := Y_i$
11  return $(sk_i, pk_i, Y)$
```

6

Proofs

In order to provide a secure scheme, there needs to be proofs. Considering that other schemes inspire VATS, many of the underlying schemes are already secure; hence, this security carries over to the security of VATS. To clarify how exactly this is done, some proofs are given in this chapter.

6.1 Correctness

The first criterion for a digital signature scheme to be usable is its correctness, which guarantees that under correct assumptions, the scheme produces a verifiable signature. This section goes over the correctness proof of the signature and the key updating scheme.

6.1.1 Correctness of Signature

Assume the existence of all the preliminaries mentioned in Chapter 5. Then, the correctness of our VATS scheme can be proven by verifying that any subset L with at least t honest members can produce a valid signature on message m. A valid signature is a Schnorr signature that returns a *true* value when passed in as input to the verification method *Ver* defined in Chapter 5.

Assume that there is an honest committee L, which has t members whose index i ranges from $(1, ..., t)$. Since the signing committee L is honest, correct behaviour and communication between signing participants are assumed. This means the following assumptions hold:

- Every participant follows and completes the algorithms in a reasonable timeline.
- If the participant A sends a message to participant B, then B gets that message.

Assume also that the vehicle in question has a pair of secret and public keys (sk, pk) where $pk = Y$.

In order for function *Ver* to verify the signature and return *true*, equation

$$g^z = \widetilde{R} \cdot Y'^c \tag{6.1}$$

must hold. The right-hand side of the equation, *RHS*, could be expanded as follows:

$$RHS = \widetilde{R} \cdot Y'^{c} = \widetilde{R} \cdot \widetilde{Y} \cdot Y \tag{6.2}$$

Where:

$$\begin{aligned}
\widetilde{R} &= \prod_{j=1}^{v} \widetilde{R}_j^{b^{j-1}} \\
&= \prod_{j=1}^{v} g^{\sum_{i=1}^{t} r_{i,j} \cdot b^{j-1}} \\
&= \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}} \\
\widetilde{Y} &= \prod_{i=1}^{t} Y_x^{\rho_i} \\
&= \prod_{i=1}^{t} g^{s_i \cdot \rho_i} \\
Y &= g^{s}
\end{aligned} \tag{6.3}$$

The right-hand side equation 6.2 can be rewritten as:

$$\begin{aligned}
RHS &= \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}} \cdot \left(\prod_{i=1}^{t} g^{s_i \cdot \rho_i} \cdot g^{s}\right)^{c} \\
&= \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}} \cdot \prod_{i=1}^{t} (g^{s_i \cdot \rho_i})^{c} \cdot (g^{s})^{c} \\
&= \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}} \cdot \prod_{i=1}^{t} g^{c \cdot s_i \cdot \rho_i} \cdot g^{c \cdot s}
\end{aligned} \tag{6.4}$$

The left-hand side of the equation in 6.1, *LHS* is the aggregated signature and can be expanded, as shown below.

$$\begin{aligned}
LHS &= g^{z} \\
&= g^{\sum_{x=1}^{t} z_x} \\
&= \prod_{i=1}^{t} g^{z_x} \\
&= \prod_{i=1}^{t} g^{c \cdot (s_i \cdot \rho_i + \lambda_i \cdot s_i) + \sum_{j=1}^{v} r_{i,j} \cdot b^{j-1}} \\
&= \prod_{i=1}^{t} g^{c \cdot s_i \cdot \rho_i} \cdot g^{c \cdot \lambda_i \cdot s_i} \cdot g^{\sum_{j=1}^{v} r_{i,j} \cdot b^{j-1}} \\
&= \prod_{i=1}^{t} g^{c \cdot s_i \cdot \rho_i} \cdot \prod_{i=1}^{t} g^{c \cdot \lambda_i \cdot s_i} \cdot \prod_{i=1}^{t} g^{\sum_{j=1}^{v} r_{i,j} \cdot b^{j-1}} \\
&= \prod_{i=1}^{t} g^{c \cdot s_i \cdot \rho_i} \prod_{i=1}^{t} g^{c \cdot \lambda_i \cdot s_i} \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}}
\end{aligned} \tag{6.5}$$

Since signing participants hold shares generated according to Shamir's algorithm [10], the original secret can be reconstructed with enough shares (equal or more than t shares) with the help of Lagrange interpolation. In other words,

$$\begin{aligned} s &= \prod_{i=1}^{t} \lambda_i \cdot s_i \\ \Leftrightarrow g^{c \cdot s} &= g^{c \cdot (\sum_{i=1}^{t} \lambda_i \cdot s_i)} \end{aligned} \tag{6.6}$$

Equation 6.5 can be rewritten as follows:

$$LHS = \prod_{i=1}^{t} g^{c \cdot s_i \cdot \rho_i} \cdot \prod_{i=1}^{t} \prod_{j=1}^{v} g^{r_{i,j} \cdot b^{j-1}} \cdot g^{c \cdot s} \tag{6.7}$$

Based on equations 6.4 and 6.7, it can be concluded that equation 6.1 holds if all the signing participants in the committee L are honest. In other words, the generated signature is verified, and the function *Ver* returns *true*.

6.1.2 Correctness of Key Updating

Assume that all the following preliminaries and assumptions described in section 5.1 are met. At least t honest participants in the network come together to form a committee L to refresh the share of all the signing participants. Assume that the vehicle has the secret key s, which is mathematically embedded in the aggregated signature to be verified against the respective public key Y during verification.

Since there are t honest participants in the key updating committee, that committee can together reconstruct s and correctly perform the digital signing of any given message. Denote the secret share of each member of L as s_x where x is the index of the respective participant. The following equation holds:

$$s = \sum_{x=1}^{t} s_x \cdot \delta_{x,0} = s_1 \cdot \delta_{1,0} + s_2 \cdot \delta_{2,0} + \dots + s_t \cdot \delta_{t,0} \tag{6.8}$$

where $\delta_{x,0}$ is the Lagrange polynomial as defined in Section 2.1.

According to VATS, Shamir's secret-sharing technique is used to further divide each participant's secret share s_x into n share parts, which will be distributed among n signing participants later. Sharing is done so that using any t share parts, s_x can be reconstructed with Lagrange interpolation, just like Shamir's secret sharing. Notice here that each share s_x is split into n parts. But reconstructing s_x requires only the sum of at least t share parts with respective Lagrange interpolation.

$$\begin{aligned} & s_1 = \delta_{1,0} \cdot s_{1,1} + \dots + \delta_{n,0} \cdot s_{1,n} \\ & s_2 = \delta_{1,0} \cdot s_{2,1} + \dots + \delta_{n,0} \cdot s_{2,n} \\ & \dots \\ & s_t = \delta_{1,0} \cdot s_{t,1} + \dots + \delta_{n,0} \cdot s_{t,n} \end{aligned} \tag{6.9}$$

Equation 6.8 can now be rewritten as:

$$\begin{aligned} s &= \delta_{1,0} \cdot s_1 + \delta_{2,0} \cdot s_2 + ... + \delta_{t,0} \cdot s_t \\ &= \delta_{1,0} \cdot (\delta_{1,0} \cdot s_{1,1} \ + \ ... \ + \ \delta_{n,0} \cdot s_{1,n}) + \\ &\quad \delta_{2,0} \cdot (\delta_{1,0} \cdot s_{2,1} \ + \ ... \ + \ \delta_{n,0} \cdot s_{2,n}) + \\ &\quad ... \\ &\quad \delta_{t,0} \cdot (\delta_{1,0} \cdot s_{t,1} \ + \ ... \ + \ \delta_{n,0} \cdot s_{t,n}) \\ &= \sum_{i=1,j=1}^{t,n} \delta_{i,0} \cdot \delta_{j,0} \cdot s_{i,j} \end{aligned} \tag{6.10}$$

After running the key updating scheme, signing participant x has the following share:

$$s_x \ = \ \delta_{1,0} \cdot s_{1,x} \ + \ ... \ + \ \delta_{t,0} \cdot s_{t,x} \tag{6.11}$$

Reconstructing the original secret can be done with the collective contribution of t out of n participants, as shown in the equation below.

$$\begin{aligned} s_{gen} &= \sum_{j=1}^{t} \delta_{j,0} \cdot (\delta_{1,0} \cdot s_{1,j} \ + \ ... \ + \ \delta_{t,0} \cdot s_{t,j}) \\ &= \sum_{j=1}^{t} \delta_{j,0} \cdot \sum_{i=1}^{t} \delta_{i,0} \cdot s_{i,j} \\ &= \sum_{i=1,j=1}^{t} \delta_{i,0} \cdot \delta_{j,0} \cdot s_{i,j} \end{aligned} \tag{6.12}$$

From equations 6.10 and 6.12, it can be concluded that $s = s_{gen}$. In other words, the reconstructed secret is the same as the original secret; hence, the signing scheme holds its correctness with the same s.

In conclusion, the key updating scheme presented in VATS refreshes the shares of each honest participant without affecting the correctness of the generated digital signature.

6.2 Security Proof

6.2.1 Random Oracle Model (ROM)

The main property of the Random Oracle Model (ROM) is to provide a publicly available random function. The idea is that ROM provides all protocol parties with good and bad intentions with access to a public function h and then proves the protocol correct, assuming that h maps each input to a truly random output. Thus, it behaves like a genuinely random oracle [39]. In practice, h would be a hash function. In this sense, it's viewed as a generator for perfectly secure hash functions. In proofs, the ROM provides heuristic support for actual security when a real hash function such as MD5, SHA-1 and SHA-256 would be used instead. The ROM enables security proofs for many schemes like MuSig2, as reductions can leverage the different properties of the ROM that can only be realized to a limited extent (if at all) in a standard model.

6.2.2 Discrete Logarithm (DL) / One-More DL

The one-more discrete logarithm (OMDL) is an extension of the discrete logarithm (DL) problem, as introduced by Bellare et al. in their paper [40]. In the DL problem, the adversary is given a group element X and needs to compute its discrete logarithm with respect to a given basis group.

In the OMDL problem, the adversary has more flexibility. It can request multiple challenges, denoted X_i, from the challenger. Additionally, the adversary has access to an oracle that can provide the discrete logarithm of any group element submitted by the adversary.

The adversary's objective in the OMDL problem is to compute the discrete logarithm of all the challenges X_i. In particular, the adversary must succeed in computing the discrete logarithm of at least one more challenge than the total number of calls made to the discrete logarithm oracle.

OMDL allows the adversary to obtain multiple challenges and exploit an oracle that returns discrete logarithms. The goal is to compute the discrete logarithms of the challenges, ensuring success in obtaining one more discrete logarithm than the number of oracle queries made.

6.2.3 Simulating Signatures

Nick et al. [6] demonstrate in their security proof about "The difficulty of simulating signature" how the first paper MuSig by Maxwell et al. [27], also referred to as "InsecureMuSig" is insecure since they claimed concurrent security under the one-more discrete logarithm assumption, but their proof turned out to be flawed. Details are mentioned in the forking lemma [41].

In MuSig2, the reduction process for handling the reduction is modified. It assumes a parameter $v = 2$ (number of nonces generated), indicating that the reduction will receive two group elements, namely $R_{i,1}$ and $R_{i,2}$, as discrete DL from the OMDL during the initial round of each signing session. This change enables the reduction to perform two discrete logarithmic queries per signing session, allowing it to simulate signatures even if the adversary forces different signature hashes ($c \neq c'$) in the two executions. By incorporating this modification, MuSig2 enhances the reduction's ability to handle different scenarios and ensures the secure execution of the scheme.

Considering the VATS scheme, the same type of nonce generation used in the key aggregation stage of the MuSig2 paper by Nick et al. is applied. This means that VATS solves the issue by having the signers effectively use the linear combination of $R_i = R_{i,1}R_{i,2}^b$ nonce, and thus get $b := H_{non}(\widetilde{Y}, (\prod_{j=1}^n R_{j,1}, \prod_{j=1}^n R_{j,2}), m)$. By programming the functions H_{non} and H_{sig} appropriately, the reduction ensures that if the adversary responds differently in the second execution with $c \neq c'$, the values b and b' will also differ. This leads to two DL queries made by the reduction producing two linearly independent equations:

$$z_i = r_{i,1} + br_{i,2} + a_i cs_i \quad \text{and} \quad z_i' = r_{i,1} + b'r_{i,2} + a_i c' s_i'.$$

These equations involve the signatures z_i and z_i', the nonce components $r_{i,1}$ and $r_{i,2}$. After the reduction has extracted s_i from the forgeries output by the adversary in the two executions, it can solve those equations for the unknowns $r_{i,1}$ and $r_{i,2}$, which are the discrete logarithms of the DL challenges $R_{i,1}$ and $R_{i,2}$. Similarly, in the case that $c = c'$, the reduction ensures that $b = b'$ and, thus, requires only one DL query to simulate the honest signer in both executions. This allows it to use the free DL query to obtain a second linear independent equation.

The reduction would work equally well in the case where the adversary controls the signature hash computed as $H_{sig}(\widetilde{Y}, R, m)$ and instead is able to choose the message m or the set of signers in the committee L (not just c); thus being able to aggregate the key $\widetilde{Y}$after seeing the honest signers' nonces. This enables the scheme to possess a preprocessing stage and broadcast the nonces without first choosing a committee and the message to sign.

According to Nick et al., there is no evidence that $v = 2$ is unsafe. However, since the reduction needs to fork the adversary twice to support key aggregation, it needs to handle four possible executions of the adversary. Therefore, Nick et al. prove that using four DL queries as well as $v = 4$ is secure. Further clarification can be found in the MuSig2 paper [6].

6.3 Reduction Proof

A reduction proof is a powerful mathematical technique used to establish the relative difficulty or security of two problems. By demonstrating that a solution to one problem can be used to construct a solution to the other problem, or vice versa, a reduction proof establishes the equivalence between the two problems. This technique is particularly valuable for analysing the security of cryptographic protocols [42].

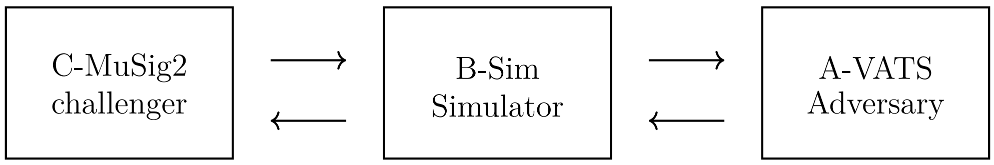


Figure 6.1: Reduction overview between MuSig2 and VATS

In the context of VATS and MuSig2, a reduction proof is employed to establish the security relationship between the two protocols. This proof allows us to conclude that if an adversary can successfully win in VATS, they would also be able to win in MuSig2, consequently compromising the security of the OMDL problem, which MuSig2 reduces to. The reduction process involves mapping the actions of the VATS adversary, who believes they are playing and attempting to break the VATS game, to the corresponding actions in the MuSig2 game, guided by a simulator. This reduction process is visually represented in Figure 6.1.

Through the reduction proof, we can confidently assert that VATS is at least as secure as MuSig2. It provides a formal justification for the security equivalence between the two protocols, reinforcing VATS's reliability and trustworthiness within the OMDL problem's broader context.

6.3.1 Programmable Random Oracle Model (PROM)

One property of the ROM is programmability, where a random oracle can be implemented and output dynamically selected values that are still correct, where they are distributed uniformly on the specific range; any method for selecting these values is considered correct [43]. A Programmable Random Oracle Model (PROM) is used in this reduction. The PROM extends the random oracle abstraction by allowing the behaviour of the oracle to be controlled programmatically during the execution of cryptographic protocols. It enables dynamic adaptation of the oracle's behaviour.

The PROM is utilized in this reduction, where a simulator (B-Sim) can interact and manipulate all hash functions called by the adversary (A-VATS). This interaction enables the reduction of the VATS scheme to the MuSig2 scheme without the hashes used in the two schemes being different. By utilizing the programmability of the random oracle, the possibility of changing the hashes of A-VATS becomes simplified, such that A-VATS does not notice that the inputs that it receives from the simulator have been manipulated.

6.3.2 Theorem

If MuSig2 is ECU-CMA secured in ROM, then VATS is also Game-VATS secure in the programmable random oracle model.

We claim that for any probabilistic polynomial time (PPT) adversary A, there is a negligible function that

$$\mid Pr[A^{Game-VATS}(\kappa) = 1] - Pr[B^{EUF-CMA}(\kappa) = 1] \mid \ \leq negl(\kappa)$$

6.3.3 Description

Before diving into the actual description of the proof, this is a quick reminder of some mathematical symbols' used in our paper.

$$
\begin{aligned}
z_x &= \text{Partial Signature} \\
\lambda_x &= \text{Langrage coefficient} \\
\rho_x &= H_{agg}(L, Y_x) \\
b &:= H_{non}(\widetilde{Y}, (\widetilde{R}_1, ..., \widetilde{R}_v),\ message) \\
c &:= H_{sig}(Y, \widetilde{R},\ message)
\end{aligned}
$$

Here is the step-by-step description and algorithms in Section 6.4 of the interaction between the challenger, simulator and adversary.

- Exchange of nonce commitments: The adversary calls Oracle SignOff() to obtain the nonce commitment of the honest party. The Simulator then queries the Oracle Sign() to the challenger and returns the honest party's nonce commitment (1). The simulator holds on to the nonce commitment out_1, which will be modified before returning it to the adversary.
- The adversary calls Oracle $Hash_{non}$ to obtain b and $Hash_{sig}$ to obtain c with the unmodified input from the adversary and the honest party. After receiving c from the oracle, the simulator changes the nonce of commitment as follows.

  $$out_1' = [R_{1,1} \cdot (Y_1^{-c\lambda_1})^{\frac{1}{b^0}}, ..., R_{1,v}]$$

  out_1' is sent back to the adversary as a response to Oracle $Sign_1$
- The adversary can now request the partial signature from the honest party by calling Oracle $SignOn()$. The simulator then calls Oracle $Sign'()$.
- The Oracle $Sign'()$ return the partial signature $\sigma_1 = (\widetilde{R}_1, z_1)$ to the simulator.
- The simulator modifies $\widetilde{R}_1$ in σ_1 as follows: $\widetilde{R}_1' = \widetilde{R}_1 \cdot Y_1^{-c\lambda_1}$
- The simulator then replies to the adversary's Oracle call $SignOn()$ with $\sigma_1 = (\widetilde{R}_1', z_1)$
- The adversary can run $SignAgg2()$ and checks the validity of each partial signature. Since the adversary generates all other partial signatures, the adversary only needs to check the validity of σ_1.

  $$
  \begin{aligned}
  LHS = g^{z_1} &= g^{cs_1\rho_1 + \sum_{j=1}^{v} b^{j-1} \cdot r_{1,j}} \\
  &= g^{cs_1\rho_1} \cdot g^{\sum_{j=1}^{v} b^{j-1} \cdot r_{1,j}} \\
  &= Y_1^{c\rho_1} \cdot Y_1^{c\lambda_1} \cdot Y_1^{-c\lambda_1} \cdot \prod_{j=1}^{v} \widetilde{R}_{1,j} \\
  &= Y_1^{c\rho_1} \cdot Y_1^{c\lambda_1} \cdot Y_1^{-c\lambda_1} \cdot \widetilde{R}_1 \\
  &= \widetilde{R}_1' \cdot Y_1^{c(\rho_1+\lambda_1)}) = RHS
  \end{aligned}
  \tag{6.13}
  $$

- The adversary now has enough to verify all partial signatures and can combine them into a valid VATS signature by calling Oracle Sign, which the simulator calls, in turn, Oracle Sign() to aggregate the partial signatures. $z = \sum_{j=1}^{t} z_j$

- The correctness of the signature z previously obtained is proven as follows (index 1 is the honest signer):

$$
\begin{aligned}
LHS &= g^{z} \\
&= g^{\sum_{j=1}^{t} z_j} \\
&= g^{(c\cdot s_1\cdot\rho_1+\sum_{j=1}^{v} r_{1,j}\cdot b^{j-1})+(c\cdot s_2\cdot(\rho_2+\lambda_2)+\sum_{j=1}^{v} r_{2,j}\cdot b^{j-1})+\ \dots\ +(c\cdot s_t\cdot(\rho_t+\lambda_t)+\sum_{j=1}^{v} r_{t,j}\cdot b^{j-1})} \\
&= g^{(c\cdot s_1\cdot\rho_1+\sum_{j=1}^{v} r_{1,j}\cdot b^{j-1})+\sum_{x=2}^{t}(c\cdot s_x\cdot(\rho_x+\lambda_x))+\sum_{j=1}^{v} r_{t,j}\cdot b^{j-1})} \\
RHS &= \widetilde{R}\cdot\widetilde{Y}^{c}\cdot\ Y^{c} \\
&= \prod_{j=1}^{v}\widetilde{R}_j^{b^{j-1}}\widetilde{Y}^{c}\cdot\prod_{j=1}^{t} Y_j^{c\cdot\lambda_j} \\
&= \widetilde{R}_1^{b'^0}\cdot\prod_{j=2}^{v}\widetilde{R}_j^{b^{j-1}}\cdot\widetilde{Y}^{c}\cdot\widetilde{Y}_1^{c\lambda_1}\cdot\prod_{j=2}^{t} Y_j^{c\cdot\lambda_j} \\
&= (R_{1,1}\cdot(Y_1^{-c\lambda_1})^{1/b^0})^{b^0}\cdot R_{2,1}^{b^0},..,R_{t,1}^{b^0}\cdot\prod_{j=2}^{v}\widetilde{R}_j^{b^{j-1}}\cdot\widetilde{Y}^{c}\cdot\widetilde{Y}_1^{c\lambda_1}\cdot\prod_{j=2}^{t} Y_j^{c\cdot\lambda_j} \\
&= R_{1,1}^{b^0},..,R_{t,1}^{b^0}\cdot\cancel{Y_1^{-c\lambda_1}}\cdot\prod_{j=2}^{v}\widetilde{R}_j^{b^{j-1}}\cdot\widetilde{Y}^{c}\cdot\cancel{\widetilde{Y}_1^{c\lambda_1}}\cdot\prod_{j=2}^{t} Y_j^{c\cdot\lambda_j} \\
&= g^{b^0\cdot\sum_{x=1}^{t} r_{x,1}}\cdot\prod_{j=2}^{v} g^{b^{j-1}\cdot\sum_{x=1}^{t} r_{x,1}}\cdot\prod_{j=1}^{t} g^{s_j\cdot\rho_j\cdot c}\cdot\prod_{j=2}^{t} g^{c\cdot s_j\cdot\lambda_j} \\
&= \prod_{j=1}^{v} g^{b^{j-1}\cdot\sum_{x=1}^{t} r_{x,1}}\cdot g^{s_j\cdot\rho_j\cdot c}\cdot\prod_{j=2}^{t} g^{s_j\cdot\rho_j\cdot c}\cdot\prod_{j=2}^{t} g^{c\cdot s_j\cdot\lambda_j} \\
&= g^{\sum_{x=1}^{t}\sum_{j=1}^{t} r_{x,1}\cdot b^{j-1}}\cdot g^{s_j\cdot\rho_j\cdot c}\cdot g^{\sum_{j=2}^{t} s_j\cdot\rho_j\cdot c}\cdot g^{\sum_{j=2}^{t} c\cdot s_j\cdot\lambda_j} \\
&= g^{(c\cdot s_1\cdot\rho_1+\sum_{j=1}^{v} r_{1,j}\cdot b^{j-1})+\sum_{x=2}^{t}(c\cdot s_x\cdot(\rho_x+\lambda_x)+\sum_{j=1}^{v} r_{x,j}\cdot b^{j-1})} \\
LHS &= RHS
\end{aligned}
\tag{6.14}
$$

- Since the adversary is assumed to be able to break the VATS scheme given a valid signature and control over $t-1$ parties, the adversary can now generate $\sigma_{A-VATS} = (R_{A-VATS}, z_{A-VATS})$, which is verifiable with VATS-scheme.

- The simulator can obtain a valid signature for MuSig2 by modifying the VATS signature generated by A-VATS. $\sigma_{C_{MuSig2}} = (R_{A-VATS}\cdot Y^{c}, z_{A-VATS})$

6.4 Game

The game for VATS shows the winning conditions for the VATS adversary and the oracles that would be included in such game.

Algorithm 13: Reduction Game

```
1 $par \leftarrow KeyGen(n, t)$
2 $counter := 0;\ k := counter$
3 $S := \emptyset,\ S' := \emptyset$
4 $Q' := \emptyset$
5 $(m, \sigma) \leftarrow A^{SignOff, Sign_1, SignOn, Sign}(B, (s_2, ..., s_t))$
6 return $Y_1 \in L \wedge (L, m) \notin Q' \wedge Ver(pk, L, m, \sigma) = true$
```

Algorithm 14: Oracle SignOff()

```
  // Incrementing session counter
1 $counter := counter + 1$
  // Open Session $k$
2 $k := counter\ ;\ S := S \cup \{k\}$
  // Generate nonces and corresponding commiment for session $k$
3 $(out_1, state_1) \leftarrow SignOff()$
4 $(R_{1,1}, ..., R_{1,v}) = out_1$
5 return $out_1$
```

Algorithm 15: Oracle SignOn($k, (out_2, ..., out_t), m, (pk_2, ..., pk_t)$)

```
  // Check the validity of session $k$
1 $if\ (k \notin S), return\ \perp$
2 $(out_1, .., out_v) := SignAgg(out_1, .., out_t)$
3 $out = (out_1, out_2, ..., out_v)$
  // Sign the message with the generated $out$, individual secret key
     $sk_1$ and the nonces $state_1$ for sesion $k$
4 $(\widetilde{R}, (z_1, R_1)) \leftarrow SignOn(state_1, out, sk_1, m, (pk_2, .., pk_t))$
5 $\sigma_1 = (z_1, R_1)$
6 $L = (pk_1, ..., pk_t)$
7 $Q := Q \cup \{(L, m)\}$
8 $S := S \setminus \{k\}; S' := S' \cup \{k\}$
9 return $(\widetilde{R}, \sigma_1)$
```

Algorithm 16: Oracle Sign($k, (\sigma_1, ..., \sigma_t)$)

```
1 $if\ (k \notin S), return\ \perp$
2 $out' \leftarrow SignAgg2((\sigma_1, ..., \sigma_t), L, m, \widetilde{R}, Y)$
3 $\sigma \leftarrow Sign(out', state'_1)$
4 $(R, z) := \sigma$
5 $S' := S' \setminus \{k\}$
6 $\sigma_{VATS} = (\widetilde{R}, z)$
7 return $\sigma_{VATS}$
```

6.5 Security Reduction

Here are the algorithms the simulator uses to reduce the VATS to MuSig2 in PROM, as illustrated in figure 6.2.

Algorithm 17: Sim-SignOff-Call()

```
// Waiting for the Oracle SignOff call from the adversary and then
   calling Oracle Sign to produce nonces.
1 $out_1 \leftarrow\ Oracle\ Sign()$
2 $(R_{1,1}, ..., R_{1,v}) = out_1$
3 $(R_1, .., R_v) := SignAgg(out_1, .., out_t)$
```

Algorithm 18: Sim-SignOff-Return(m)

```
1 $b \leftarrow ROM_{musig2}\ H_{non}(\widetilde{Y}, R_1, .., R_v, m)$
2 $R = \prod_{j=1}^{v}\ R_j^{b^{j-1}}$
3 $c \leftarrow ROM_{musig2}\ H_{sig}(\widetilde{Y}, R, m)$
4 $out'_1 = (R_{1,1} \cdot Y^{-c\lambda_1}, ..., R_{1,v})$
// $c$ and $b$ can be saved or recomputed later
5 return $out'_1$
```

Algorithm 21: Sim-SignOn($k, \hat{out}, m, (\sigma_2, .., \sigma_t), (pk_2, ..., pk_t)$)

```
1 $(\widetilde{R}_1, ..., \widetilde{R}_v) := \hat{out} \in \mathbb{G}^v$
2 $(\hat{R}_1, \hat{z_1})\ \leftarrow\ Oracle\ Sign'(k, out, m, (pk_2, .., pk_t))$
3 $out'_1 := (\hat{R}'_1, \hat{z_1})$
4 $\widetilde{R} := \prod_{j=1}^{v} \widetilde{R}_j^{b^{j-1}} \in \mathbb{G}$
5 return $(\widetilde{R}, out'_1)$
```

Algorithm 19: Hash$_{sig}$(k ,$\widetilde{Y}$, $\widetilde{R}$, m, out)

```
// All queries are made to the Musig2 ROM
1 $if\ (k \notin S), return\ \perp$
2 $(R_1, .., R_v) := out \in \mathbb{G}^v$
3 $R := \prod_{j=1}^{v} R_j^{b^{j-1}} \in \mathbb{G}$
4 $c \leftarrow ROM_{musig2}\ H_{sig}(\widetilde{Y}, R, m)$
5 return $c$
```

Algorithm 20: Hash$_{non}$(k ,$\widetilde{Y}$, R$'_1$, .., R'_v, m, out)

```
// All queries are made to the MuSig2 ROM
1 $if\ (k \notin S), return\ \perp$
2 $(R_1, .., R_v) := out \in \mathbb{G}^v$
3 $b \leftarrow ROM_{musig2}\ H_{non}(\widetilde{Y}, R_1, .., R_v, m)$
4 return $b$
```

Algorithm 22: Sim-Sign(k, out')

```
1 $\sigma \leftarrow Oracle\ Sign''(k, out')$
2 $(R, z) := \sigma$
3 $S' := S' \setminus \{k\}$
4 $\sigma_{VATS} = (\widetilde{R}, z)$
5 return $\sigma_{VATS}$
```

Algorithm 23: Sim-ForgeSig($\sigma_{VATS-forged}$)

```
1 $(R_{VATS}, z_{VATS}) = \sigma_{VATS-forged}$
2 $R_{MuSig2} = R_{VATS} \cdot Y^c$
3 $\sigma_{MuSig2-forged} = (R_{Musig2}, z_{VATS})$
4 return $\sigma_{MuSig2-forged}$
```

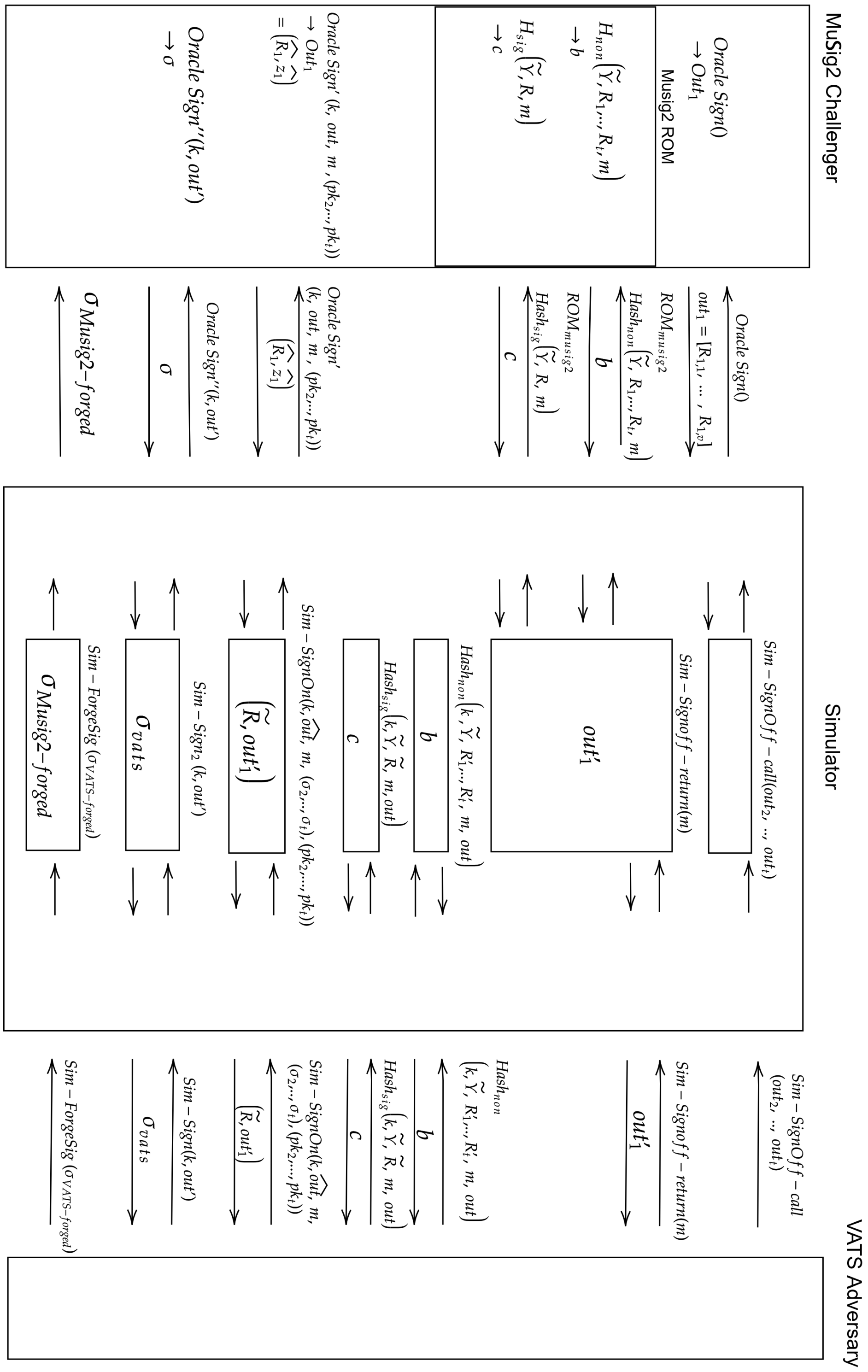

MuSig2 Challenger
Oracle $Sign()$ $\rightarrow Out_1$
Musig2 ROM
$H_{non}\left(\widetilde{Y}, R_1, \ldots, R_t, m\right) \rightarrow b$
$H_{sig}\left(\widetilde{Y}, R, m\right) \rightarrow c$
Oracle $Sign'(k,\ out,\ m\ , (pk_2, \ldots pk_t)) \rightarrow Out_1 = \left(\widehat{R_1}, \widehat{z_1}\right)$
Oracle $Sign''(k, out') \rightarrow \sigma$
Oracle $Sign()$
$out_1 = [R_{1,1}, \ldots, R_{1,v}]$
ROM_{musig2} $Hash_{non}\left(\widetilde{Y}, R_1, \ldots R_t, m\right)$
b
ROM_{musig2} $Hash_{sig}\left(\widetilde{Y}, R, m\right)$
c
Oracle $Sign'$ $(k,\ out,\ m,\ (pk_2, \ldots pk_t))$
$\left(\widehat{R_1}, \widehat{z_1}\right)$
Oracle $Sign''(k, out')$
σ
$\sigma_{Musig2-forged}$
Simulator
$Sim-SignOff-call(out_2, \ldots, out_t)$
$Sim-Signoff-return(m)$
out_1'
$Hash_{non}\left(k, \widetilde{Y}, R_1', \ldots R_t', m, out\right)$
b
$Hash_{sig}\left(k, \widetilde{Y}, \widetilde{R}, m, out\right)$
c
$Sim-SignOn(k, \widehat{out}, m, (\sigma_2, \ldots \sigma_t), (pk_2, \ldots pk_t))$
$\left(\widetilde{R}, out_1'\right)$
$Sim-Sign_2(k, out')$
σ_{vats}
$Sim-ForgeSig(\sigma_{VATS-forged})$
$\sigma_{Musig2-forged}$
$Sim-SignOff-call$ $(out_2, \ldots out_t)$
$Sim-Signoff-return(m)$
out_1'
$Hash_{non}$ $\left(k, \widetilde{Y}, R_1', \ldots R_t', m, out\right)$
b
$Hash_{sig}\left(k, \widetilde{Y}, \widetilde{R}, m, out\right)$
c
$Sim-SignOn(k, \widehat{out}, m, (\sigma_2, \ldots \sigma_t), (pk_2, \ldots pk_t))$
$\left(\widetilde{R}, out_1'\right)$
$Sim-Sign(k, out')$
σ_{vats}
$Sim-ForgeSig(\sigma_{VATS-forged})$
VATS Adversary

Figure 6.2: Reduction overview

7

Evaluation

In this chapter, the performance of the VATS scheme, which has been implemented, is assessed. The scheme offers a variety of features and aims to provide enhanced security and efficiency for specific use cases. To benchmark its performance, all tests were conducted on a 7800X3D (8 cores, 16 threads at 4.200 GHz base clock) processor running Manjaro v6.1.29. ECUs are generally less powerful than the computer used to evaluate the performance of VATS scheme. But considering the time stress, available hardware and the computational power needed to evaluate the scalability of the scheme, the computer configuration was chosen because of availability. This evaluation aims to measure the execution time, resource utilisation, and scalability of the scheme, providing insights into its practicality and effectiveness. For clarification, in this implementation, each n is the number of potential signers, such that increasing the number n gives you more potential signers, while increasing t requires more active signers for each signature. Having a hundred or more units in vehicle settings performing signing operations is informative to evaluate scalability but not in accordance with the vehicle architecture presented in section 4.2.

7.1 Performance Evaluation

To evaluate the performance of specific components' execution time in VATS, different numbers of threshold t and total n were used to demonstrate the scalability of the scheme. By varying these parameters, the impact on the scheme's execution time can be analysed. The resulting data were plotted on a graph, with the x-axis representing threshold t values and the y-axis representing the total n values. The colour-coded legend indicates the corresponding execution time for each data point, providing a visual representation of the performance characteristics. This visualisation enables a clear understanding of how the scheme's execution time varies based on different t and n values, helping to assess its scalability,l identify potential weak scalable components and highlight areas for optimisation. Reasonable times for these measurements are discussed later in Chapter 8.

7.1.1 General Purpose Evaluation

Notice that despite the colour legend for execution time is the same for all the graphs, the time scale is different for each graph. The x- and y-axis denote t and n are the same as all the presented graphs.

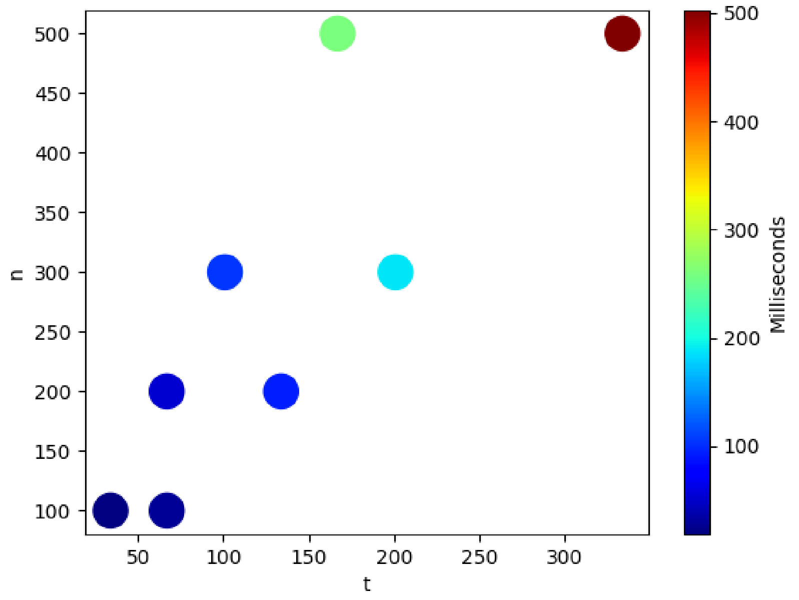


Figure 7.1: Execution time for Key generation (KeyGen) with different values of (t,n)

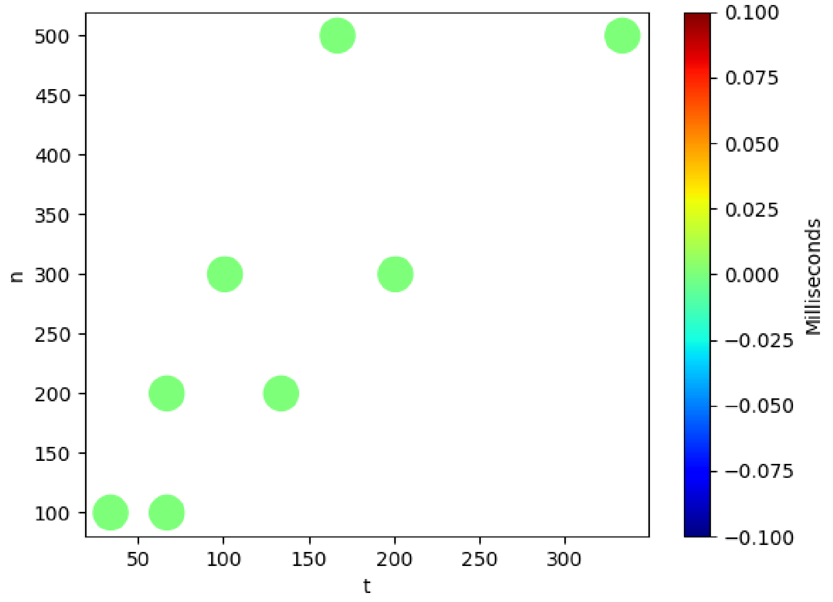


Figure 7.2: Execution time for Sign Offline (SignOff) with different values of (t,n)

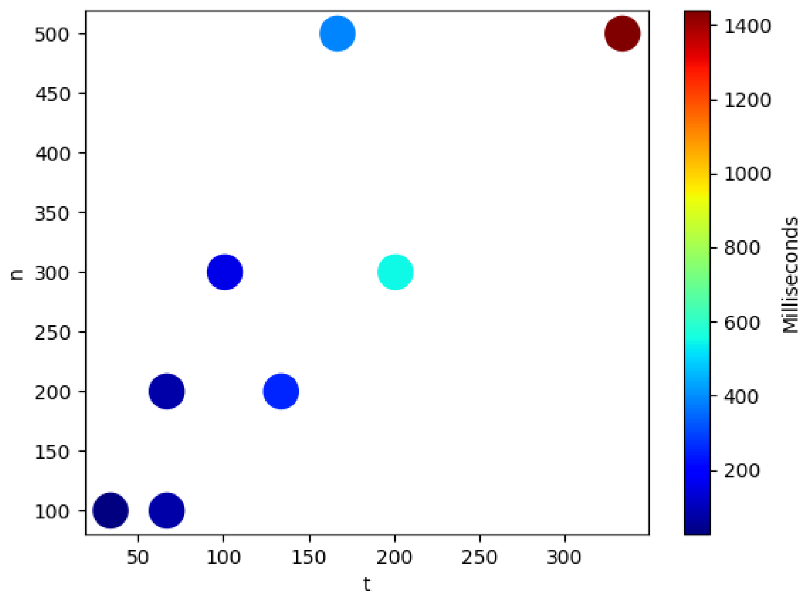


Figure 7.3: Execution time for Sign Online(SignOn) with different values of (t,n)

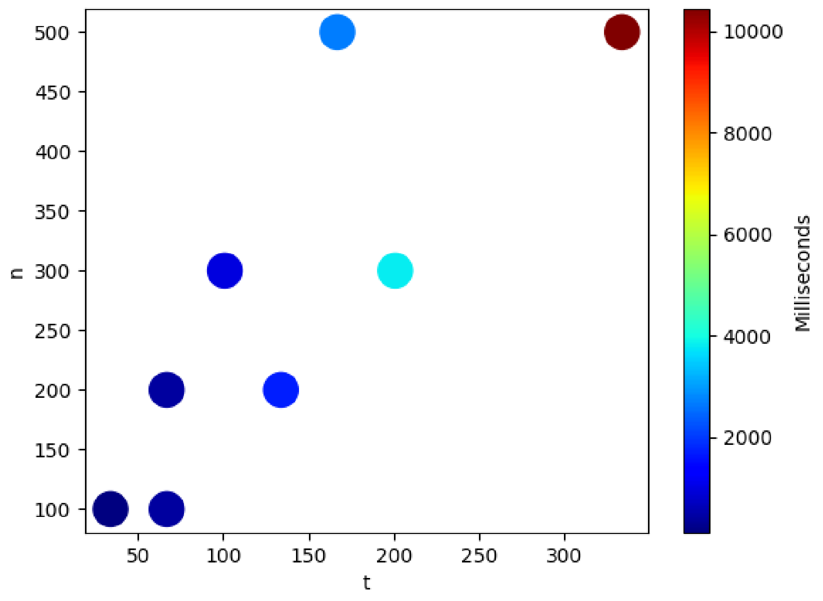


Figure 7.4: Execution time for Second Signing Aggregation (SignAgg2) with different values of (t,n)

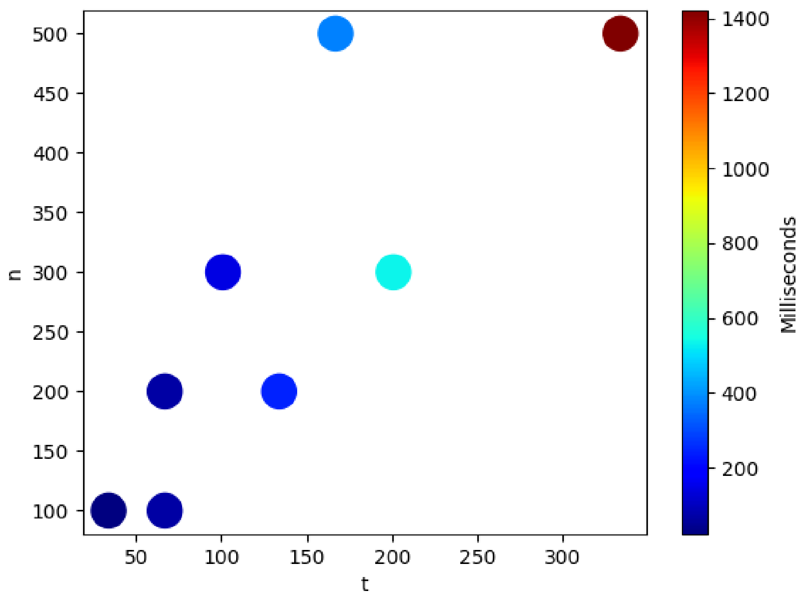


Figure 7.5: Execution time for Verification (Ver) with different values of (t,n)

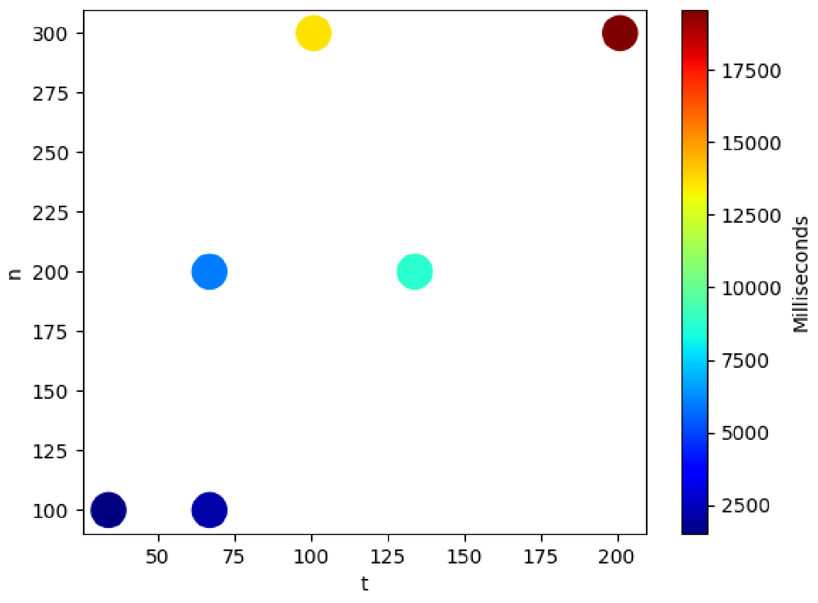


Figure 7.6: Execution time for Key updating (KeyUpd) with different values of (t,n)

7.1.2 Architecture Specific Evaluation

To better match the architecture described in Section 4.2, we conducted another evaluation using a much smaller number of signers (which also means fewer ECUs). This provides a more accurate reflection of the expected setup, giving us performance metrics relevant to real vehicle implementation. In this instance, the benchmarking remains in milliseconds and illustrates the plotting of different values using a reduced threshold (t) and fewer ECUs/signers (n).

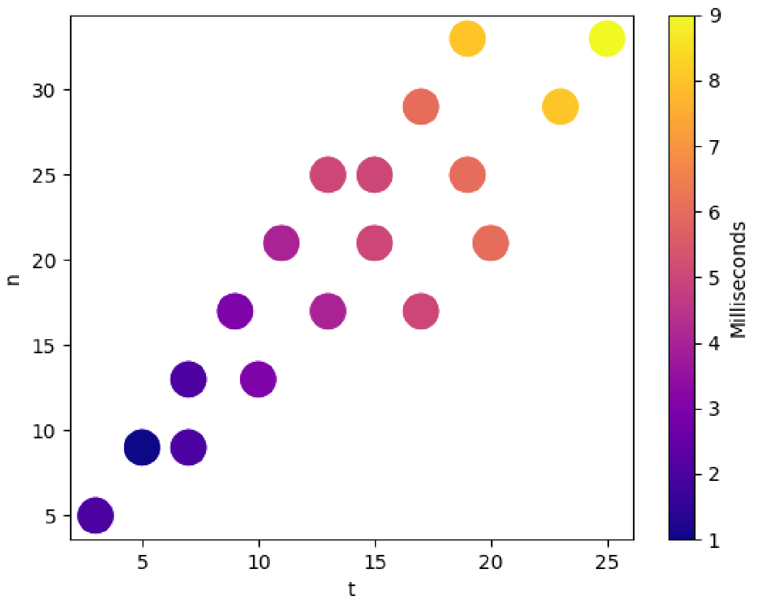


Figure 7.7: Execution time for Key generation (KeyGen) in accordance with the architecture

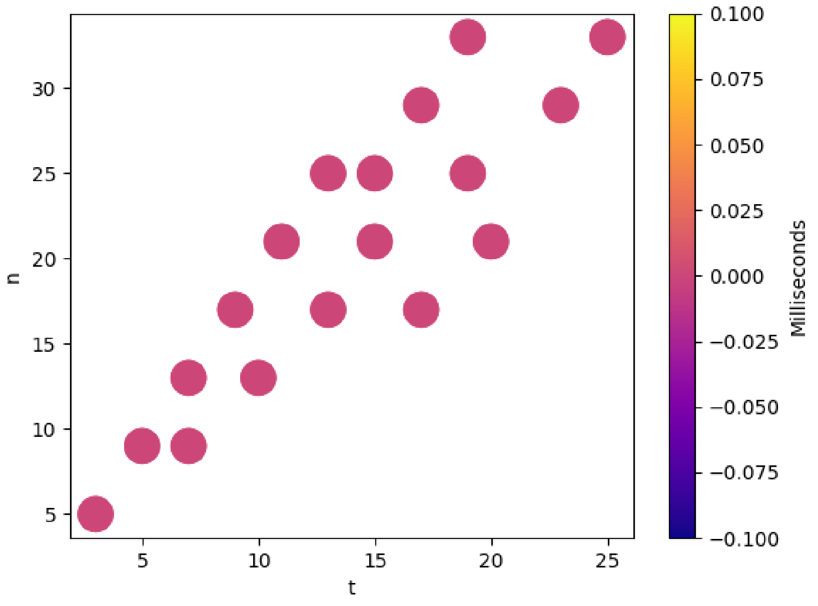


Figure 7.8: Execution time for Sign Offline (SignOff) in accordance with the architecture

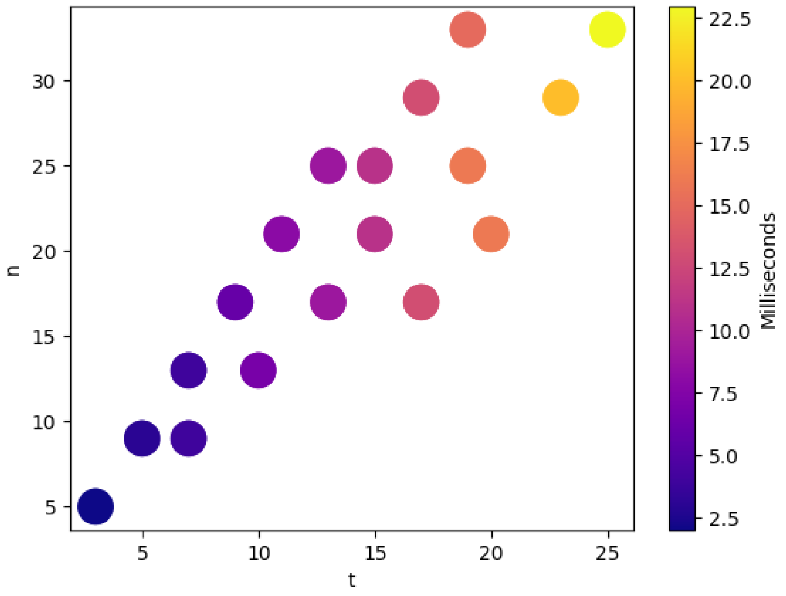


Figure 7.9: Execution time for Sign Online(SignOn) in accordance with the architecture

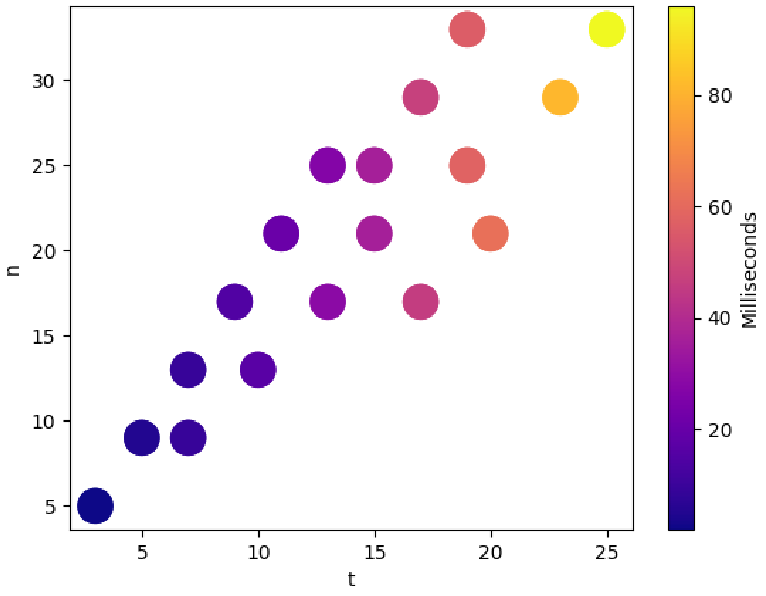


Figure 7.10: Execution time for Second Signing Aggregation (SignAgg2) in accordance with the architecture

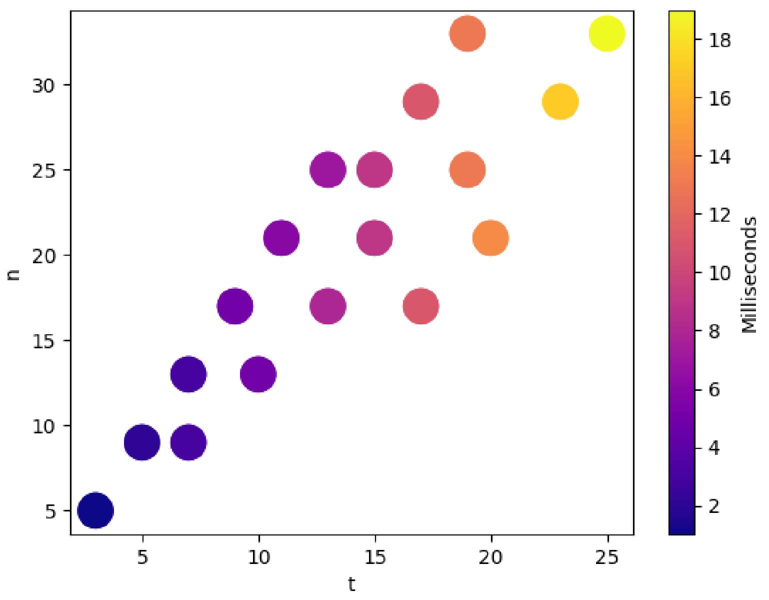


Figure 7.11: Execution time for Verification (Ver) in accordance with the architecture

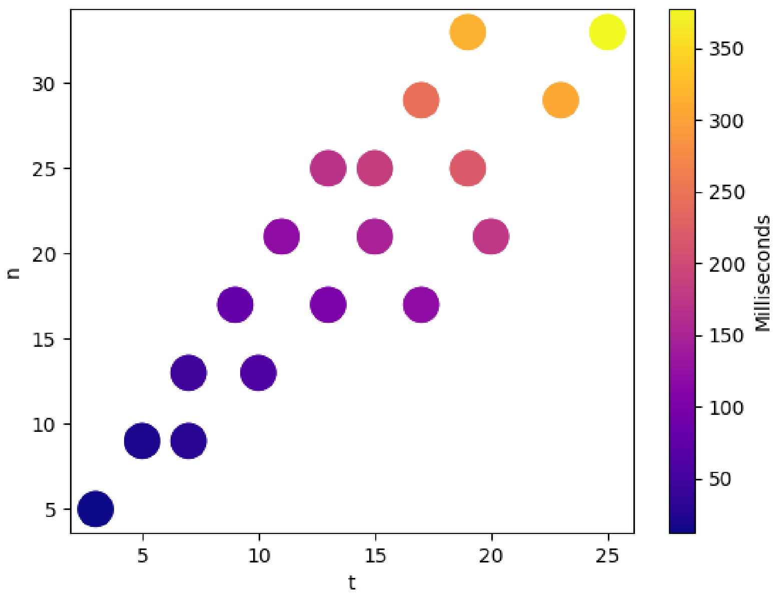


Figure 7.12: Execution time for Key updating (KeyUpd) in accordance with the architecture

7.2 Data Complexity

The data complexity of a scheme is difficult to measure due to the various configurations and design choices made for each particular implementation. In this report, the data presented complexity is solely evaluated on this implementation of VATS. Since VATS is an interactive scheme, it is of interest to measure the amount of data transferred between signers and the signing aggregator. Thus, the data complexity in this context refers to the amount of essential data, for example, public key, commitment, and signatures, being transferred in the network. Message headers are disregarded in this evaluation, as they are the same for all messages and greatly depend on the communication protocol used.

In this implementation of VATS, the public elements in the scheme, such as public key and commitment, are represented by *RistrettoPoint* which is 160 bytes in memory. Private keys and shares are *Scalar* variables that represent integers on the finite field defined by a prime number $2^{255} - 19$ and are 32 bytes in size each. The data sent and received by the scheme's participants consist of these two variable types with the exception of the message to sign in type *String* and the identifier of each participant in type u32, which is an unsigned integer of 32 bits (4 bytes). As previously defined, t and n are respectively the threshold and the total number of participants in this scheme. The following section presents the data complexity of the algorithms that construct the scheme.

- **KeyGen**: In this algorithm, the signing aggregator sends the share to each of the signing participants, as well as broadcasting the commitment to verify the shares. The share is a single scalar value. Meanwhile, the commitment's size dependence grows proportionally to the threshold t times the size of a RistrettoPoint. The following equation presents the data complexity of the KeyGen protocol:

$$d_{keygen}(t, n) \;=\; n\,(t \cdot 160 + 32)\; Bytes$$

- **SignOff**: After generating the nonces, each participant has to send the commitments on the randomly generated nonces to each other participant. Assume that the number of nonces is v, then the total load on the network will be:

$$d_{SignOff}(v, n) \;=\; n \cdot v \cdot 160(n-1)\; Bytes$$

- **SignOn**: After finishing generating its partial signature, each signer in the signing committee sends it back to the signing aggregator. The partial signature consists of three elements: the aggregated nonce commitment $\widetilde{R}$ - RistrettoPoint, the individual signature z_i - Scalar, and the individual commitment R_i - RistrettoPoint. The lower limit for the number of signers required in the signing committee is the threshold t by design. But there is no upper limit for it besides the obvious fact that there cannot be more than n signers in the committee because there are at most n honest signers. The inequality below shows the workload that the scheme put on the network during Signon.

$$t \cdot (160 + 32 + 160)\; Bytes \;\leq\; d_{SignOn} \;\leq n \cdot (160 + 32 + 160)\; Bytes$$

- **Sign**: The signature produced by VATS is the same as any other Schnorr-based digital signature. In this implementation, the aggregated signature consists of a RistrettoPoint and Scalar. The data size of the signature is unchanged regardless of the size of the committee, the threshold t, or the number of participants n.

$$d_{Sign} \ = \ 160 + 32 \ Bytes$$

- **KeyUpd**: KeyUpd is divided into two rounds with different complexity. The number of members in the committee L, which is responsible for creating and sharing the new share components, is not constrained by anything other than being less n. The choice of L has a great impact on the size of the messages to be sent in round one. In round one, each member of the KeyUpd committee broadcasts their commitment, which consists of t RistrettoPoint, and an individual KeyUpd signature of type Scalar to later prove the legitimacy of the new share. Assume that there are t' signers in the committee L. The following equation:

$$d_{KeyUpd_1} \ = \ t' \ \cdot \ n \cdot \ (t \cdot 160 \ + \ 32)$$

  describes the relationship between the complexity of the data in this stage with respect to (t', t, n).

  In the second round of KeyUpd, each member of the KeyUpd committee L sends to each of the participants their respective new share part, which is a Scalar to be added to their current secret share. Thus, the complexity here scales with t' and n.

$$d_{KeyUpd_2} \ = \ t' \ \cdot \ n \cdot 32 \ Bytes$$

  According to the pseudocode of the implementation, each committee member of L sends along the recipient's ID, which adds another 4 Bytes to the message. This is not completely necessary for the success of this algorithm and, therefore, omitted in this calculation.

7.3 Memory Complexity

Like data complexity, memory complexity depends greatly on the implementation itself. Factors such as t and n, as well as the choice of cyclic group and integer representations, affect the memory complexity of the implementation. This section presents the memory required for each roll in the scheme to be functional. This evaluation does not take into account the fixed memory amount that is required to store the application, but rather the memory that is required to store shares, public keys, commitments, etc.

- **Share generator**: The application runs the KeyGen algorithm to create and distribute the shares for signing participants. A share generator must be able to store a polynomial of degree $(t-1)$ and a total of n generated shares to be sent. Both are stored as Scalar values. Also, it has to store the commitment to the generated shares, which are t RistrettoPoint values. So, the memory complexity for the share generator is as shown in the following equation:

$$\begin{aligned} mem_{share\_generator} &= t \cdot coefficient \ + n \cdot share \ + \ t \cdot commitment \\ &= (t \ + \ n) \cdot 32 \ + \ t \cdot 160 \ Bytes \end{aligned}$$

- **Signing Aggregator**: The task of a signing aggregator is to verify, store, and aggregate the signature. In order to do that, it has to allocate enough memory to at least store t and at most n partial signatures as well as all the components to calculate the challenge to verify each partial signature. To do so, the signing aggregator needs to store the nonce commitments and public keys of the signing committee. At most, the signing committee can include n signers. Therefore, the signing aggregator has to have enough memory to accommodate that. In reality, the allocated memory space is not necessarily filled in every signing session.

$$\begin{aligned} mem_{signing\_aggregator} &= n \cdot v \cdot commitments \ + \ n \cdot pubkeys \ + \\ & \quad n \cdot partial\_signature \\ &= n \cdot 160 \cdot v \ + \ n \cdot 160 \ + \ n \cdot (160 + 32) \ Bytes \end{aligned}$$

- **Signing participant**: Each signer has to store a similiar amount of information compared to the signing aggregator, with the exception that it only has to store one partial signature. Moreover, each signing participant also has to store its own share and the nonces it generates in each signing session (SignOff).

$$\begin{aligned} mem_{signing\_aggregator} &= \ n \cdot v \cdot commitments \ + \ n \cdot pubkeys \ + \\ & partial\_signature \ + \ v \cdot nonces \ + \ share \\ &= \ n \cdot v \cdot 160 \ + \ n \cdot 160 \ + (160 + 32) \ + \\ & v \cdot 32 \ + \ 32 \ Bytes \end{aligned}$$

- **Signature verifier**: Verification of the aggregated signature requires the knowledge of the vehicle public key pk, and all the individual public keys of each signing participant. The maximum amount of memory that needs to be allocated for this function is:

$$\begin{aligned} mem_{verification} &= n \cdot individual\_pubkey \ + \ (aggregated\_nonce + signature) \\ &= n \cdot 160 \ + \ (160 + 32) \ Bytes \end{aligned}$$

8 Discussion

8.1 Design Choice

VATS is built based on Schnorr's digital signature, which was decided by considering the following factors.

- **Security**: Both RSA and Schnorr have been rigorously proven to be secure with the right key size. They are both accepted, standardised, and widely used in industry. Therefore, security is not a concern, and it is probable that a threshold signature scheme is built on either.
- **Signature and key size**: Signature and key size might not be a problem for regular computer systems, but in a vehicular setting, it is advantageous to aim to have a smaller signature and key size. This is because of the lesser storage it requires to store said keys, as well as faster computation on resource-limited devices. The key size recommendation to archive reasonable security for Schnorr-based signatures is significantly lower than that based on RSA, as described in Section 2.2.3. Since the signature is meant to be verified by an entity outside the vehicle, the Schnorr signature puts less load on the network than RSA-based because of the size difference.
- **Performance**: Doing calculations with the Schnorr signature is significantly easier than RSA due to the key length difference. In RSA, two large prime numbers must be calculated, and this task is nontrivial to do efficiently. Besides, the performance of Schnorr's algorithm benefits significantly from the small size of its components, such as challenge, key, and nonce.

In general, for vehicular applications, a Schnorr-based digital signature seemed to be the better choice when considering the specific requirements and compatibility of the system.

8.2 Performance

The assessment of the scheme's feasibility in vehicular implementation would be beneficial to determine its practicality. In Section 7, many graphs were presented to demonstrate the efficiency of the scheme. When these graphs are examined, a potential manufacturer can make an informed decision about integrating the protocol into a vehicle environment. Considering that each component can be run separately, evaluating the entirety of the protocol would need to be more accurate. Considering that the entirety of the scheme (from start to finish), when used in practice, does not run sequentially, even with some parts being done before signing as an offline stage.

Since each component plays an essential part in the scheme, it also gives excellent information about the weaknesses that the scheme might suffer from. As mentioned previously, security is the primary goal when developing a scheme like VATS. However, it is important to note that one can avoid suffering from performance loss if there is a better but still secure way of doing it.

To the best of our knowledge, VATS is the first scheme to apply all the requested properties. Therefore, it will also have to potentially suffer from the bad performance that can come from being the first protocol. Considering VATS as an early scheme, one can start discussing its performance. We discuss each component separately since they have been benchmarked separately. Doing separate benchmarking is necessary because the scheme is asynchronous, and the components are not required to be executed serially.

The performance of Keygen, as shown in Figure 7.7, is a crucial component of the scheme and is executed only once by a trusted dealer to certify the signing key with a certificate authority, usually the manufacturer. Although scalability in this component is less critical, since it occurs only once, its performance is still better than those of other components. Any potential performance issues in the Keygen phase would not significantly impact the overall signing process due to it being a component executed once at the manufacturer. However, for the sake of transparency, it is still an evaluated component in the thesis.

Signoff, evaluated in Figure 7.8 is a component of the scheme that each signer executes as a pre-processing stage for signing, which means that having executed SignOff before the signature request arrives is essential to obtain good performance, such that the signature process does not stall due to there being too heavy computations done. The performance of the component depends greatly on how many nonces each ECU is expected to generate; To maximise performance and security at this stage, two nonces ($v = 2$) are generated so that it is still safe according to the proof in Section 6.2.3. Increasing the threshold or the number of signers would not affect this component, which is best demonstrated in Figure 7.2.

SignOn performance, as shown in Figure 7.9, is the second part of the signature process, where each chosen signer on the committee attempts to sign the message. A well-performing SignOn is essential so that each signature runs efficiently. Too many delays would stall the signing process, much like SignOff. However, unlike SignOff, the SignOn execution time depends heavily on the number of signers in the scheme and is generally very computationally heavy. Therefore, it is critical to optimise SignOn, as shown in Figure 7.3. Each signer's performance is a significant variable, since SignOn is executed individually. If multiple ECUs in the vehicle vary too much in computational power, the execution time of SignOn will be significantly affected (especially by weak computation). Therefore, it could result in SignOn becoming a significant bottleneck in the signature process. This could be considered in the committee selection process, and ECUs with better performance also receive higher scores, such that they have a higher chance to be chosen during signing.

SignAgg2 is the component responsible for combining each signature in the signature aggregator. Its performance, as shown in Figure 7.4, depends on the number of signatures that must be aggregated. In VATS, the signing aggregator is also responsible for validating each signature to catch potential cheaters before their signature is aggregated. This requires significant computation, especially since it is performed sequentially. However, one potential optimisation is to aggregate the signatures as they arrive rather than waiting until we have enough partial signatures to aggregate. The estimated time in our evaluation is the time it takes to aggregate all the partial signatures received simultaneously. Some precomputation steps could be taken to optimise SignAgg2 further if the committee is known beforehand. For example, an optimisation could be precomputing the challenge c, ρ_x, etc.

Verification is the step executed by the receiver of the signature, which in this case might be a manufacturer server that expects an authentication message from the vehicle. The verification step requires some computation to ensure accountability. Figure 7.5 shows that the verification scalability is very similar to that of SignOn in circumstances with similar hardware. This is expected since a partial signature produced by SignOn is also similar to a Schnorr signature and is validated similarly to Schnorr's. However, because of the execution on different machines, this similarity cannot be expected in reality.

KeyUpd is the slowest component in the scheme and is extremely computationally heavy, as shown in both Figures 7.1 and 7.7. KeyUpd requires that each participant in the committee generates a polynomial of $t - 1$ and verifies the shares received from other participants. The KeyUpd method was only tested with a threshold t of 200 and a total n of up to 300 signing participants due to the computational complexity of its evaluation when $n = 500$. The evaluation was run single-threaded and sequentially for all signing participants. In reality, each signing participant performs the operations by itself and could significantly reduce the computation time. On the other hand, ECUs, as mentioned before, are expected to have less computational power than the computers used in this evaluation. Therefore, the execution time of KeyUpd and other VATS algorithms would be higher than what is presented in the evaluation. Hopefully, this would be less of a concern with ECU computing becoming stronger.

Although KeyUpd sacrifices efficiency for added security, it is an essential component that is necessary for the scheme to avoid being compromised over time. This is especially important, considering that VATS is meant to be a practical scheme.

Since no other schemes like VATS exist, it is hard to set standards to measure its effectiveness. Therefore, we present the performance results without deciding whether the performance can be deemed satisfactory or unsatisfactory. However, this limitation shows how unique and innovative the VATS scheme is for vehicular communication. It not only addresses security concerns but also brings up performance considerations that need to be explored and improved. Although performance measurements are helpful, they leave room for future research and comparison. By understanding the performance characteristics of the scheme, researchers can improve VATS for vehicular communication systems.

8.3 Data and Memory Complexity

Data and memory complexity are among the deciding factors in a scheme's usability in a vehicular setting. As mentioned in limitation 1.5.1, bandwidth and storage capacity for ECU in vehicles are limited. Looking at the evaluation, VATS has $O(n^2)$ complexity in the data during many operations, for example, KeyGen and SignOff. Compared to other components, the heaviest load is placed on the network during KeyUpd. Assume that all the data required for a vehicle with 100 ECUs to do key updating are unloaded onto the vehicle network at once. According to our estimation, the key update scheme will put a load of approximately 1.6 MB of data. Having 1.6 MB of data sent between ECUs would be difficult for a CAN bus to handle with a limited bandwidth of around 1 MB/s, as mentioned in Section 4.2. However, an Ethernet connection would have no problem handling similar amounts of data. In addition, this operation is not time-sensitive or meant to be performed regularly. Therefore, VATS is believed to be applicable and usable according to a vehicle's intended conceptual network configuration.

The memory complexity of VATS depends significantly on the number of signing participants n in the network. Two roles are meant to be run by ECUs in a vehicle: the signing aggregator and signing participant. Judging by the memory estimation in Evaluation 7, the largest memory required to run over 100 ECUs in a modern vehicle is still less than 1 MB. Since there is no standard specification for ECUs, it is hard to know if VATS is applicable for any specific ECU. However, it would not be unreasonable to assume that the memory requirement for running VATS does not prevent modern vehicles from running the scheme. Because in addition to strong security, Ethernet is meant to widen the bandwidth of the communication network to handle vast amounts of data for data-intensive applications.

Due to VATS's data and memory complexity, it does not feel suitable to use it in a setting with thousands of ECUs. But VATS should be applicable for the current vehicular settings depending on the actual configuration of ECUs in specific vehicles.

8.4 Risk Analysis and Ethical Considerations

Bleeding-edge technologies open up many opportunities and subject engineers to a sea of ethical challenges requiring careful navigation. Thorough analysis and consideration of the code of ethics for engineers is a must for engineers to not stubble over unwanted but avoidable ethical dilemmas with potentially devastating consequences. This section is a self-reflection of ethical and moral considerations while doing this work.

8.4.1 Software Engineering Code of Ethics

A joint task force between the Association for Computing Machinery and the IEEE Computer Society has developed a code of ethics for software engineering as a guideline on software engineers' ethical and professional responsibilities and obligations [44]. With the principles documented in the code of ethics, software engineers are inspired to broadly consider the affected parties by one's work, to analyse the effect that one's work has on the least empowered in society; and to consider whether one's actions would be judged to be worthy of the ideal software engineering profession. So, in light of the code of ethics, our work in the project will be analysed and carried out as most consistent with its spirit, given the circumstances.

8.4.1.1 Product

The principle states that engineers shall try to ensure that the quality of the software they work on is acceptable to the public, employers, etc. Therefore, the highest possible standards must thrive. Although this project aims only to provide a proof-of-concept of threshold signatures, the proof-of-concept will be rigorously tested. The results of these tests will be documented in detail to further improve the quality of the proof-of-concept to the best possible quality in the given time frame of this project.

8.4.1.2 Public

According to this principle, software engineers should act in ways consistent with the public's interests regarding safety, health, and welfare to the best of their professional capacities. The advancement of humanity's knowledge on the topic of threshold signatures has always been the fundamental goal of this research. As knowledge should be free and accessible to all who are eager to learn, it means this knowledge could be applied in such a way that dramatically benefits humanity or seriously violates the interests of the major public.

The main application of the threshold signature schemes that are considered, researched, and developed in this project is intended to increase the security and redundancy of vehicular identity by protecting the private key in TLS communication. This is a noble cause and consistent with the public interest. For this reason, the project is consistent with the public principle.

Recently, there have been many discussions and even lawsuits about what is called *Right to Repair* [45]. *Right to Repair* refers to a movement that demands the right of customers to repair electronic components in machines and devices by themselves or third parties of their choice. This stems from the fact that many machines nowadays are computerised, greatly increasing repair processes' complexity. Manufacturers abuse this fact and do not publicly release the same repair manual that their franchised dealerships have access to. For example, customers must go to authorised workshops to repair their vehicles. In addition, embedded components contain chips that monitor their state of health and communicate with the rest of the vehicle. If these components are coded to only work if authorised by manufacturers, this severely limits the end-users ability to do the reparation themselves. The customer does not truly own the component or machine because they do not have complete control over their machines. This is not ethical and goes against the public interest.

8.4.1.3 Judgment and profession

This principle concerns the integrity and independence of the developers, which should be maintained during the project. Although it is under direct supervision from the university and the host company, this project has been and will be judged as unbiased and independently as possible. The project and we, as engineers working on it, shall live up to the integrity and reputation of the profession by revisiting ethical considerations as often as necessary during the project development.

According to the Code of Ethics for Software Engineers, other principles should be considered. However, they are not applicable and relevant to this project.

8.4.2 Risk Analysis

The intended threshold signature scheme researched and developed in this project strengthens the security of the digital signature scheme while adding redundancy compared to multiparty digital signature schemes. The protocol and implemented proof-of-concept are non-invasive and not a direct threat to anyone. A proof of concept should not be used directly as part of an application in real life. The reason is that developers, for several thinkable reasons, might omit certain procedures to ensure security, usability, and efficiency when designing and implementing the proof-of-concept as opposed to developing a part of an actual application. With that said, there is still a risk of this proof-of-concept being used in a non-intended way and causing unnecessary risks for users down the line.

Threshold digital signature schemes are well known among the cryptocurrency community as a tool to increase the protection of private keys in crypto wallets [46], for example, or to mitigate the risk of single-point failure when one entity is entrusted with securing private keys [47]. Cryptocurrency as a currency itself is neither ethically good nor bad. It only becomes ethically questionable when it is used, for example, as payment for illegal or/and unethical services or goods. The question of whether the researchers who develop these signature schemes are now responsible emerges. The scope of controllability over the usage of a protocol that a researcher produces is limited. As long as the knowledge is accessible to the public, there is next to nothing that the researcher can do to completely prevent the said protocol from being used with bad intentions. Moreover, even when knowledge is locked up and only accessible to a selected few, assuming that such knowledge would only be used for good causes is unreasonable. A conclusion can be drawn that there is a small but real risk that the technique developed in this project will be used unethically.

However, given the benefits of the research field and with consideration of the potential risks, the advancement of knowledge should not be stopped since the benefits still outweigh the risks.

8.5 Sustainability

While VATS may entail a modest uptick in computational demands compared to conventional signature schemes, it extends a substantial boost to the security of resultant signatures. The importance of heightened vehicular security can not be undermined in the ongoing effort to protect the identity and integrity of vehicle users. Ensuring security in V2X communication could save a great deal of resources in resolving the damages in case of a large-scale cyberattack on the transportation infrastructure as we move closer to autonomous driving.

The knowledge and result of this work contribute to the growing effort of deepening our understanding and developing a sustainable way to use threshold cryptography to ensure accountability and security of digital communication. The VATS scheme contributes to our sustainability by enabling others to explore the capability of technology in the internet-of-things era without security concerns.

8.6 Future Work

This project successfully developed a threshold signature scheme with some desirable properties, such as verifiable secret sharing, accountability, and proactivity for applications in vehicular settings. Considering the given time frame for this project, some aspects could have been improved to optimise the scheme and its implementation further.

The KeyUpd component of VATS is computationally intensive and time-consuming compared to the other components, as demonstrated by the evaluation and discussed in the discussion. Although it can be argued that this inefficiency is justifiable due to the fact that it can be done while the car is not in use, it is still of interest to improve the efficiency in terms of both computational power consumption and time requirement. One potential solution to this problem is to have a trusted central computational unit perform the work and distribute the updated shares to all participants instead of having each participant do it individually. This would significantly reduce the computation and time required for vehicle key updates. However, this would require a trusted entity, which could potentially undermine the security of the implementation if the risk of relying on one ECU unit in the vehicle outweighs the benefits gained. This performance impact would also be somewhat mitigated if an implementation of the scheme picks ECUs that are stronger computationally; hence, the computational overhead would not be as severe. However, this would require optimisation concerning computational power, which is outside of the scope of what this thesis is trying to achieve in the given timeframe; as such, it could be seen as more of an additional project on top of this one.

The proof-of-concept implementation of the scheme is functional, but it can still be vulnerable to cyberattacks due to loopholes in the implementation. The assumptions and preliminaries made for the scheme to be functional are achievable in reality, as demonstrated by our implementation. The curve25519 used in this implementation can be improved for better security, and the length of the key can be adjusted to improve performance on resource-limited hardware, although this would lead to a lower security grade. Therefore, the design choices for implementation can be tailored to specific hardware configurations and security requirements. This is also why some things about the scheme can not be generated, such as how often one runs key updating and when we generate new nonces with signOff, because this can be very subjective depending on the implementation and is intended to be configured depending on the need or security standard.

9 Conclusion

This project has successfully developed a threshold signature scheme, VATS, with verifiable secret sharing, accountability, and share renewal. It is suitable for use in resource-constrained devices in a vehicular setting.

When evaluating the performance of VATS, it is natural to question whether its efficiency meets the requirements for practical implementation. However, it is essential to consider the broader context and weigh the trade-offs between performance and security. Although it is true that the VATS scheme may introduce some additional performance loss compared to traditional signature schemes, it also significantly enhances the security of the generated signatures. This trade-off between performance and security is a critical consideration in vehicular environments where ensuring the authenticity and integrity of communication is of utmost importance.

The argument can be made that prioritising the security of signatures outweighs the potential performance impact. If the primary concern is to provide end-users with secure and trustworthy vehicular communication systems, a scheme like VATS becomes important. By utilising threshold signature techniques and secret sharing schemes, VATS ensures that the private key remains protected even in the presence of compromised or malfunctioning electronic control units.

In summary, while the performance of VATS components may present some trade-offs, the scheme's focus on enhancing security and preserving the integrity of signatures makes it a valuable solution for practical implementation in vehicular systems.

Bibliography

[1] L. Strous, S. von Solms, and A. Zúquete, "Security and privacy of the internet of things," *Computers Security*, vol. 102, p. 102 148, 2021, ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2020.102148`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167404820304211`.

[2] M. Shavit, A. Gryc, and R. Miucic, "Firmware update over the air (FOTA) for automotive industry," SAE Technical Paper, Tech. Rep., 2007. [Online]. Available: `https://doi.org/10.4271/2007-01-3523`.

[3] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, 2008, pp. 380–384. DOI: `10.1109/ICCW.2008.78`.

[4] H. Tschofenig and T. Fossati, "Transport layer security (TLS)/datagram transport layer security (DTLS) profiles for the internet of things," Tech. Rep., 2016. [Online]. Available: `https://www.rfc-editor.org/rfc/rfc7925`.

[5] M. S. U. Alam, "Securing vehicle electronic control unit (ECU) communications and stored data," Ph.D. dissertation, Queen's University (Canada), 2018.

[6] J. Nick, T. Ruffing, and Y. Seurin, *MuSig2: Simple two-round Schnorr multi-signatures*, Cryptology ePrint Archive, Paper 2020/1261, 2020. [Online]. Available: `https://eprint.iacr.org/2020/1261`.

[7] C. Komlo and I. Goldberg, "FROST: flexible round-optimized Schnorr threshold signatures," in *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, Springer, 2021, pp. 34–65.

[8] A. González, H. Ratoanina, R. Salen, S. Sharifian, and V. Soukharev, *Identifiable cheating entity flexible round-optimized Schnorr threshold (ICE FROST) signature protocol*, Cryptology ePrint Archive, Paper 2021/1658, 2021. [Online]. Available: `https://eprint.iacr.org/2021/1658`.

[9] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in CryptologyCRYPTO89 Proceedings 9*, Springer, 1990, pp. 239–252.

[10] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[11] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on*, IEEE Computer Society, 1979, pp. 313–313.

[12] C.-P. Schnorr, “Efficient signature generation by smart cards,” *Journal of cryptology*, vol. 4, pp. 161–174, 1991.

[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, *High-speed high-security signatures*, Cryptology ePrint Archive, Paper 2011/368, 2011. [Online]. Available: `https://eprint.iacr.org/2011/368`.

[14] D. J. Bernstein, “Curve25519: New Diffie-Hellman Speed Records,” in *Public Key Cryptography - PKC 2006*, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 207–228, ISBN: 978-3-540-33852-9.

[15] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[16] National Institute of Standards and Technology, *Digital signature standard*, 2023. [Online]. Available: `https://csrc.nist.gov/publications/detail/fips/186/5/final`.

[17] T. P. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Advances in CryptologyCRYPTO91: Proceedings*, Springer, 2001, pp. 129–140.

[18] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, “Verifiable secret sharing and achieving simultaneity in the presence of faults,” in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, IEEE, 1985, pp. 383–395.

[19] P. Feldman, “A practical scheme for non-interactive verifiable secret sharing,” in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, 1987, pp. 427–438. DOI: `10.1109/SFCS.1987.4`.

[20] B. Schoenmakers, *Lecture notes cryptographic protocols*, 2022.

[21] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, “Verifiable secret sharing and achieving simultaneity in the presence of faults,” in *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985, pp. 383–395. DOI: `10.1109/SFCS.1985.64`.

[22] D. Boneh and C. Komlo, “Threshold signatures with private accountability,” in *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*, Springer, 2022, pp. 551–581.

[23] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: How to cope with perpetual leakage,” in *Advances in Cryptology — CRYPT0’ 95*, D. Coppersmith, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 339–352, ISBN: 978-3-540-44750-4.

[24] T. P. Pedersen, “A threshold cryptosystem without a trusted party,” in *Advances in CryptologyEUROCRYPT91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, Springer, 1991, pp. 522–526.

[25] D. Zelle, C. KrauSS, H. StrauSS, and K. Schmidt, “On using tls to secure in-vehicle networks,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES ’17, Reggio Calabria, Italy: Association for Computing Machinery, 2017, ISBN: 9781450352574. DOI: `10.`

1145/3098954.3105824. [Online]. Available: https://doi.org/10.1145/3098954.3105824.

[26] Y. Desmedt, "Society and group oriented cryptography: A new concept," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 120–127, ISBN: 978-3-540-48184-3.

[27] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, *Simple Schnorr multi-signatures with applications to bitcoin*, Cryptology ePrint Archive, Paper 2018/068, 2018. [Online]. Available: https://eprint.iacr.org/2018/068.

[28] A. Bagherzandi, J.-H. Cheon, and S. Jarecki, "Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma," in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 449–458.

[29] M. Drijvers, K. Edalatnejad, B. Ford, *et al.*, "On the security of two-round multi-signatures," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1084–1101. DOI: 10.1109/SP.2019.00050.

[30] D. Wagner, "A generalized birthday problem," in *Advances in Cryptology-CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*, Springer, 2002, pp. 288–304.

[31] R. Srinivasan, A. Sharmili, S. Saravanan, and D. Jayaprakash, "Smart vehicles with everything," in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, 2016, pp. 400–403. DOI: 10.1109/IC3I.2016.7917997.

[32] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, vol. 2014, p. 94, 2014. [Online]. Available: https://img.hardworkingtrucks.com/files/base/randallreilly/all/migrated-files/hwt/2014/09/Remote_Automotive_Attack_Surfaces.pdf.

[33] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *Trans. Intell. Transport. Syst.*, vol. 16, no. 2, pp. 534–545, Mar. 2015. DOI: 10.1109/TITS.2014.2320605. [Online]. Available: https://doi.org/10.1109/TITS.2014.2320605.

[34] N. D. Matsakis and F. S. Klock, "The Rust language," *Ada Lett.*, vol. 34, no. 3, pp. 103–104, Oct. 2014, ISSN: 1094-3641. DOI: 10.1145/2692956.2663188. [Online]. Available: https://doi.org/10.1145/2692956.2663188.

[35] D. cryptography, *Ed25519-dalek*, Github Repository, May 2023. [Online]. Available: https://github.com/dalek-cryptography/ed25519-dalek#readme.

[36] Tokio-Rs, *Tokio*, Github Repository, May 2023. [Online]. Available: https://github.com/tokio-rs/tokio.

[37] S. McArthur, *Warp*, Github Repository, May 2023. [Online]. Available: https://github.com/seanmonstar/warp.

[38] C. Boettiger, "An introduction to docker for reproducible research," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 71–79, Jan. 2015, ISSN: 0163-5980. DOI: 10.1145/2723872.2723882. [Online]. Available: https://doi.org/10.1145/2723872.2723882.

[39] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on*

Computer and Communications Security, ser. CCS '93, Fairfax, Virginia, USA: Association for Computing Machinery, 1993, pp. 62–73, ISBN: 0897916298. DOI: 10.1145/168588.168596. [Online]. Available: https://doi.org/10.1145/168588.168596.

[40] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The One-More-RSA-Inversion problems and the security of Chaum's blind signature scheme*, Cryptology ePrint Archive, Paper 2001/002, 2001. [Online]. Available: https://eprint.iacr.org/2001/002.

[41] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06, Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 390–399, ISBN: 1595935185. DOI: 10.1145/1180405.1180453. [Online]. Available: https://doi.org/10.1145/1180405.1180453.

[42] K. Brusse and F. Ellen, "Reductions and extension-based proofs," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21, Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 497–507, ISBN: 9781450385480. DOI: 10.1145/3465084.3467906. [Online]. Available: https://doi.org/10.1145/3465084.3467906.

[43] M. Fischlin, A. Lehmann, T. Ristenpart, T. Shrimpton, M. Stam, and S. Tessaro, "Random oracles with(out) programmability," in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 303–320, ISBN: 978-3-642-17373-8.

[44] D. Gotterbarn, K. Miller, and S. Rogerson, "Software engineering code of ethics," *Commun. ACM*, vol. 40, no. 11, pp. 110–118, Nov. 1997, ISSN: 0001-0782. DOI: 10.1145/265684.265699. [Online]. Available: https://doi.org/10.1145/265684.265699.

[45] Svensson, Sahra and Richter, Jessika Luth and Maitre-Ekern, Eléonore and Pihlajarinne, Taina and Maigret, Aline and Dalhammar, Carl, *The Emerging Right to Repair legislation in the EU and the U.S.* eng, 2018. [Online]. Available: https://lup.lub.lu.se/search/files/63585584/Svensson_et_al._Going_Green_CARE_INNOVATION_2018_PREPRINT.pdf.

[46] C. A. Team. "What is the threshold signature scheme?" (2021), [Online]. Available: https://cryptoapis.io/blog/78-what-is-the-threshold-signature-scheme.

[47] A. A. Cassandra Heart. "Threshold digital signatures." (2021), [Online]. Available: https://www.coinbase.com/blog/threshold-digital-signatures.