Formally Verified Remote Attestation Protocols with Strong Authentication

Formellt verifierade fjärrattesteringsprotokoll med stark autentisering

Johannes Wilson

Supervisor: Felipe Boeira
Examiner: Mikael Asplund

External supervisor: Niklas Johansson
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtidiga ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnt som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida
http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page:
http://www.ep.liu.se/

© Johannes Wilson
Abstract

Most commodity processors available today provide hardware-supported security extensions. Remote attestation has been declared an important step towards providing security to users through such solutions, yet remote attestation has seen limited deployment in practice. For existing protocols, analysis of the protocol security is not always publicly available. This thesis utilises formal methods in order to investigate an existing remote attestation protocol in the form of Samsung Knox Enhanced Attestation V3 developed by Samsung for Samsung Knox devices. Requirements are formalised into verifiable security properties. Formal verification reveals a minor weakness when considering strong adversarial models that can control parts of the device through run-time attacks. Such adversarial models are generally stronger than what is typically considered for Knox devices. This thesis also develops general remote attestation protocols which remedy the found weakness with a simple mechanism. Our developed protocols are formally verified, showing that they are suitable for platforms with Trusted Execution Environments even when considering strong adversarial models.
Acknowledgments

I would like to thank Sectra Communications, and especially Niklas Johansson for trusting me with this project. I want to thank Niklas for the excellent report feedback he provided, as well as the very helpful discussions. I am very grateful for the positive environment that the people at Sectra have provided me with. I would also like to thank my supervisor Felipe Boeira and my examiner Mikael Asplund for lending their time and their expertise on the subject area through many insightful meetings and e-mails. Finally, I would like to thank my friends and family who have encouraged me all throughout my studies.
Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures vii
List of Tables viii
List of Listings ix
1 Abbreviations 1
2 Introduction 2
  2.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  2.2 Aim . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  2.3 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  2.4 Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  2.5 Delimitations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  2.6 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
3 Background and Related Work 6
  3.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    3.1.1 ARM TrustZone . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    3.1.2 Remote Attestation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
    3.1.3 Samsung Knox Enhanced Attestation V3 . . . . . . . . . . . . . . . . . . 9
    3.1.4 Formal Verification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
    3.1.5 Security Properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  3.2 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
    3.2.1 Verification of Direct Anonymous Attestation . . . . . . . . . . . . . . . . 14
    3.2.2 EPID-based Remote Attestation in Intel SGX . . . . . . . . . . . . . . . . . 14
    3.2.3 Formal Verification of Remote Attestation in Intel TDX . . . . . . . . . . 15
    3.2.4 Verified Hybrid Attestation . . . . . . . . . . . . . . . . . . . . . . . . . . 15
    3.2.5 Formally Verified TrustZone TEE . . . . . . . . . . . . . . . . . . . . . . . 16
    3.2.6 Verification of TLS 1.3 Specification . . . . . . . . . . . . . . . . . . . . . . 16
4 Adversary Models and Security Requirements 17
  4.1 Terminology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
  4.2 Adversary Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
    4.2.1 Adversary Model A . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
    4.2.2 Adversary Model B . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  4.3 Attestation Goals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Remote attestation example</td>
<td>3</td>
</tr>
<tr>
<td>3.1</td>
<td>TrustZone Software Stack</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Enhanced Attestation v3 for Samsung KNOX devices</td>
<td>10</td>
</tr>
<tr>
<td>4.1</td>
<td>Adversary model A</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>Adversary model B</td>
<td>18</td>
</tr>
<tr>
<td>5.1</td>
<td>Two-agent RA protocol</td>
<td>28</td>
</tr>
<tr>
<td>5.2</td>
<td>Two-agent RA protocol with secret sharing</td>
<td>30</td>
</tr>
<tr>
<td>5.3</td>
<td>Three-agent RA protocol</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>Three-agent RA protocol with secret sharing</td>
<td>34</td>
</tr>
<tr>
<td>7.1</td>
<td>Relay attack on Enhanced Attestation V3 by strong adversary</td>
<td>43</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Attacks that could compromise the security of remote attestation</td>
<td>9</td>
</tr>
<tr>
<td>3.2</td>
<td>Attestation Principles presented by Coker et al.</td>
<td>9</td>
</tr>
<tr>
<td>3.3</td>
<td>Summary of related work</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>Attestation goals</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>Mapping of Attestation Principles presented by Coker et al. and attestation goals</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>Protocol attacks and what attestation goal they are covered by</td>
<td>21</td>
</tr>
<tr>
<td>4.4</td>
<td>Verification goals and their corresponding security properties</td>
<td>24</td>
</tr>
<tr>
<td>5.1</td>
<td>Description of steps in two-agent RA protocol</td>
<td>28</td>
</tr>
<tr>
<td>5.2</td>
<td>Description of steps in two-agent RA protocol with secret sharing</td>
<td>31</td>
</tr>
<tr>
<td>5.3</td>
<td>Description of steps in three-agent RA protocol</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>Description of steps in three-agent RA protocol with secret sharing</td>
<td>33</td>
</tr>
<tr>
<td>5.5</td>
<td>Summary of developed RA protocols</td>
<td>33</td>
</tr>
<tr>
<td>6.1</td>
<td>Tamarin models</td>
<td>36</td>
</tr>
<tr>
<td>6.2</td>
<td>Tamarin model overview</td>
<td>36</td>
</tr>
<tr>
<td>6.3</td>
<td>List of all lemmas used during verification</td>
<td>36</td>
</tr>
<tr>
<td>6.4</td>
<td>Summary of tamarin modelling decisions</td>
<td>40</td>
</tr>
<tr>
<td>6.5</td>
<td>Verification results for protocols with adversary model A</td>
<td>41</td>
</tr>
<tr>
<td>6.6</td>
<td>Verification results for protocols with adversary model B</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Listings

<table>
<thead>
<tr>
<th>3.1 Example Tamarin rule</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 TLS channel modelled using confidential and secure channels in Tamarin</td>
<td>38</td>
</tr>
<tr>
<td>6.2 Secure TrustZone channel modelled in Tamarin</td>
<td>38</td>
</tr>
<tr>
<td>6.3 Insecure TrustZone channel modelled in Tamarin</td>
<td>39</td>
</tr>
<tr>
<td>9.1 Lemma 1: Correctness of Protocol</td>
<td>56</td>
</tr>
<tr>
<td>9.2 Lemmas 2–4: Authenticity of TEE</td>
<td>57</td>
</tr>
<tr>
<td>9.3 Lemma 5: Freshness of Attestation Data</td>
<td>58</td>
</tr>
<tr>
<td>9.4 Lemma 6 for two-agent: Integrity of Attestation Data</td>
<td>58</td>
</tr>
<tr>
<td>9.5 Lemma 6 for three-agent: Integrity of Attestation Data</td>
<td>58</td>
</tr>
<tr>
<td>9.6 Lemma 7–8: Integrity of Attestation Verdict</td>
<td>59</td>
</tr>
<tr>
<td>9.7 Lemmas 9–12: Authenticity of Verifier</td>
<td>60</td>
</tr>
<tr>
<td>9.8 Lemma 13: Confidentiality of Shared Secret</td>
<td>61</td>
</tr>
<tr>
<td>9.9 Lemmas 14–15: Integrity of Shared Secret</td>
<td>62</td>
</tr>
</tbody>
</table>
Abbreviations

Following is a list of abbreviations that are used throughout the thesis.

- RA Remote Attestation
- TEE Trusted Execution Environment
- REE Rich Execution Environment
- TOCTOU Time-Of-Check to Time-Of-Use
- IoT Internet of Things
- OS Operating System
- EPIP Enhanced Privacy ID
The introduction chapter first presents the motivation for the thesis (Section 2.1). Next, the aim and overall goals are described in Section 2.2 followed by the research questions in Section 2.3. The method used during the thesis is described in Section 2.4. Finally, the delimitations are presented in Section 2.5 and the main contributions of the thesis are described in Section 2.6.

2.1 Motivation

Malware currently poses a significant threat to security on mobile devices. A large userbase, along with the storage of highly sensitive and personal data, causes mobile devices to be frequently targeted by malware. Energy constraints limit the possibility for anti-malware software on the device itself [9]. While separate servers can be used to detect vulnerabilities [45], it is still very difficult to provide real-time protection on mobile devices. Privilege escalation attacks [46] and rootkits [10] also challenge the assumption that the operating system can be trusted, requiring a deeper fundamentally secure unit of computation [48]. Modern processors come equipped with specific hardware solutions for providing security-critical services such as the storage of cryptographic keys. Such solutions have gained a lot of interest as a means to not only provide safe execution of security and integrity critical software, but to also verify the correct execution of software on the device through the use of remote attestation. Remote attestation allows a provider to verify that a device is running as intended, and has not been compromised through malware [23]. The device, known as the prover, is sent a request and collects information about the device state that can be sent back to the remote, known as the verifier, for verification. An example remote attestation procedure is shown in Figure 2.1.

Remote attestation can be used to assure the level of trust in a platform. In addition to malware detection, remote attestation can make sure that mobile devices adhere to a company policy before connecting to a company network [49]. For cloud services, remote attestation can be used before provisioning data workloads to the service in order to ensure the platform can support the confidentiality and integrity of the data through hardware support [29].

Remote attestation normally assumes some level of hardware support, and remote attestation protocols have been proposed and implemented for a number of hardware solutions, including Intel Software Guard Extensions (SGX), Intel Trust Domain Extensions (TDX), and
2.2. Aim

The purpose of this thesis is to provide a solid foundation for the design of remote attestation protocols for mobile devices. For the purpose of formal verification of remote attestation protocols, this thesis collects necessary security requirements for remote attestation protocols, and designs general-purpose remote attestation protocols that fulfills all requirements.

There protocols are also designed to allow provisioning secrets to the device in order to guarantee hardware supported storage. Existing remote attestation protocols for Intel SGX and TDX exemplify why such mechanisms are often necessary for practical remote attestation protocols. This thesis generalises such mechanisms in order to provide secret sharing for any platform utilising Trusted Execution Environments, including mobile devices.

Remote attestation has seen limited use on mobile devices so far, yet interest in trusted devices has only increased. Enterprise devices often contain security extensions to protect devices, yet attestation of such security extensions is often lacking or limited. In order to
evaluate if existing remote attestation protocols fulfil the specified requirements, this thesis additionally provides analysis of a state-of-the-art remote attestation protocol. Samsung’s Enhanced Attestation for their KNOX devices was selected as a suitable verification target for this thesis. Formally verifying such a protocol is meant to exemplify a method that can be utilised in the verification of other remote attestation protocols that target the TrustZone architecture.

2.3 Research Questions

This thesis focuses on the following research questions:

1. What security properties are necessary to uphold in a protocol for remote attestation?
2. How can a protocol for remote attestation be designed for platforms with Trusted Execution Environments in order to guarantee the security properties?
3. How can remote attestation protocols be formalised for the purpose of formal verification using Tamarin?
4. How can remote attestation be extended in order to provide secret sharing to mobile devices?

2.4 Approach

In order to answer the first research question regarding what security properties are necessary, a literature study is performed. The requirements are based on existing research on remote attestation protocols. Appropriate requirements are specified and formalised as security properties, and these security properties form the basis of the development of the attestation protocols. The attestation protocols are developed iteratively as follows:

1. Develop an attestation protocol for use on platforms with Trusted Execution Environments with existing protocols as a starting point.
2. Develop a model for the protocol in a language for formal verification, and attempt to verify that the security properties hold.
3. Based on what security properties could not be verified, refine and improve the attestation protocol. Repeat until the security properties can be successfully verified.

The security properties are also used to analyse the existing Samsung KNOX Enhanced Attestation v3 attestation protocol, in order to compare the developed protocols with the security of existing state-of-the-art protocols.

Formal verification is performed using two adversary models, one with a weak adversary and the other with a strong adversary. The strength of the protocols can then be compared by analysing which security properties can hold for each model.

2.5 Delimitations

This thesis only investigates high-level protocol implementations for Remote Attestation. Integrity measurement mechanisms that could be used as part of the attestation are out of scope. Implementation level verification is also out of scope. This thesis only investigates remote attestation protocols applicable for platforms with Trusted Execution Environments, focusing in particular on the TrustZone architecture.
2.6 Contributions

The main contributions of this thesis are the following:

• A specification of the necessary security properties of a protocol for remote attestation.

• Two robust protocols for remote attestation that fulfils strong requirements for authentication and secrecy.

• An analysis of an existing state-of-the-art protocol for remote attestation, and a discussion of its strengths and weaknesses.

• Two remote attestation protocols that provide secret sharing to Trusted Executions Environments on mobile devices.
3 Background and Related Work

This chapter summarises some of the important concepts related to remote attestation and formal verification, and also provides an overview of previous work in the area. In Section 3.1 background theory is presented. Section 3.2 showcases previous work within formal verification of remote attestation or related technologies.

3.1 Background

In this section, the main concepts are explained and defined, starting with an introduction to the Arm TrustZone architecture and trusted execution environments in Section 3.1.1. Next, remote attestation is explained in more detail in Section 3.1.2 along with an example of an existing implementation in the form of Samsung’s Enhanced Attestation protocol described in Section 3.1.3. Finally, concepts related to the formal verification of protocols are introduced in Section 3.1.4 including an introduction to the Tamarin prover, along with relevant security properties in Section 3.1.5.

3.1.1 ARM TrustZone

Trusted Computing [40] has been discussed in the literature since the early 2000’s. The need for trusted computing has evolved as computers have become increasingly connected and accessible, with many users carrying computers in the form of smartphones and laptops with them at all times. Security is difficult to guarantee from software alone since the software itself can be vulnerable to attacks. Securing operating systems has also proved difficult due to their sheer size and ever-increasing complexity. In addition, highly sophisticated malware (e.g. the Stuxnet worm [22]) written by highly motivated attackers have appeared. These things taken together highlights the importance of secure systems, and has motivated the development of Trusted Computing.

Instead of trying to secure every part of a system, which is a daunting challenge, the idea of trusted computing is to provide a Root of Trust. A Root of Trust provides some basic guarantees on the system, such as the secrecy and integrity of some cryptographic keys. The Root of Trust can then be extended through careful measurements of the system state.

Current trusted computing implementations leverage hardware solutions in order to provide security extensions. Some systems leverage dedicated hardware chips separate from
3.1. Background

Figure 3.1: TrustZone Software Stack. The Normal World is shown on the left, and the Secure World on the right. The Normal World implements the REE, while the Secure World implements the TEE. Both worlds use multiple privilege levels, named from EL0 to EL3.

The main processor, known as a Trusted Platform Module. Other systems, such as ARM TrustZone or Intel SGX/TDX processors instead provide a secure environment for software to run, known as a Trusted Execution Environment (TEE). Since Arm processors have been dominating the marked for mobile and IoT devices [24], ARM TrustZone has become the most common solution to provide security extensions for mobile devices.

Figure 3.1 shows an overview of the TrustZone software stack based on the Arm documentation [35]. In TrustZone, execution is divided into two environments, known as the Secure World and the Normal World. In the figure, the Normal World is shown on the left, and the Secure World on the right. The Secure World is the implementation for the TEE on TrustZone, while the Normal World contains the usual software stack, referred to as the Rich Execution Environment (REE). Both worlds share the processor, and so both worlds cannot run at the same time. Both worlds use multiple privilege levels, named from EL0 to EL3. The privilege level EL0 has the least privilege, and is used by the user applications. Privilege levels EL1 and above all have increasing levels of privilege. The secure Monitor, running at the highest privilege, is responsible for changing context between the Normal World and the Secure World. In the figure, each row uses the same privilege level.

The intended operation is for the user applications and OS functionality to remain in the Normal World, while security critical functionality such as cryptographic functions can be handled in the Secure World. The idea is that as long as the TEE can remain secure, the cryptographic keys should never be able to leak to an adversary, even in the case that malware has compromised privileged software such as the OS on the REE. The TEE can also be used as a trusted environment from which integrity measurements of the software running can be performed. Such integrity measurements can for instance be used as a means to perform remote attestation.

3.1.2 Remote Attestation

Remote Attestation (RA) is the process in which a device, known in RA as the prover (sometimes called attester or target), proves its level of trust to a remote party, known as the verifier (sometimes called challenger or appraiser). A successful attestation should provide sufficient guarantees to the verifier that the prover is operating as intended and has not been compromised. RA is a necessary component for establishing the trustworthiness of a remote party.

One crucial aspect for establishing trust is that a compromised device cannot be trusted to display accurate information about its state [42]. If we consider sophisticated attacks that compromise the OS of the device, any attempt to display a warning to the user that the device is compromised could be subverted by the malware, which could even display a seemingly
valid but faked report instead. One benefit of Remote Attestation is that there is no reliance on the device itself to display the results. Although a verifier could in practice be compromised, we do not require a single specific verifier. Thus, we can cross-reference the results through another request from another verifier, or change verifier to one that we trust. This is especially beneficial if the device we are attesting is at higher risk of being compromised than the verifier.

There may be several different goals for RA. Some remote attestation protocols only attest the software integrity at load-time [33], while others try to perform integrity measurements during run-time in order to detect exploits in vulnerable software [21] [1] [19] [64]. RA can also attest software at the user level, or only attest OS and firmware.

Remote attestation can also target a wide variety of platforms, from servers, personal computers and smartphones to low-end IoT devices. This diversity of attestation goals and target platforms has led to a wide variety of different attestation protocols.

Remote attestation usually relies on secure hardware to store cryptographic keys used to sign the attestation data. Assuming that these keys cannot be leaked from the secure hardware, such signature proves that the attestation was performed on the intended hardware. Some attestation protocols have been designed to be used without the use of dedicated hardware, known as software attestation [31] [34] [57] [56] [25]. Such protocols generally have to assume a weaker adversary, commonly by requiring that communication between prover and verifier is one-hop, meaning the message delay is statically known. This thesis will focus on utilising features present in secure hardware, since most commodity processors are shipped with hardware security extensions in some form.

Requirements

All remote attestation protocols involve the prover providing some evidence of its state. Sailer et al. [50] consider several attacks that are important for a RA protocol to defend against. In particular, a remote attestation protocol that measures the integrity of the system should be resilient to replay attacks, tampering of the integrity measurements and masquerading attacks. Kovah et al. [32] also point out the importance of protecting against time-of-check to time-of-use (TOCTOU) discrepancies. These attacks are listed and briefly explained in Table 3.1. A masquerading attack could be conducted using an attacker-in-the-middle attack, and so to protect from masquerading attacks we should also consider attacker-in-the-middle scenarios, including relay attacks where messages are simply forwarded to another device, and the response is relayed back. Protecting against these attacks requires strong authentication and integrity guarantees for the protocol, as well as some mechanism to prove recency of the attestation.

De Oliveira Nunes et al. [18] show how TOCTOU problems can be handled through sophisticated RA techniques. They utilise hardware support to track changes to memory. This allows them to detect the presence of malware, even if said malware was only present in-between RA requests.

In addition to the attacks listed previously, a paper by Coker et al. [14] specifies some general principles that remote attestation should adhere to. A summary of their findings is presented in Table 3.2. The descriptions from their paper uses slightly different terminology: the target corresponds to the prover, while the appraiser corresponds to the verifier. In addition, the attestation architecture corresponds to an attestation server. They consider both the goals of the prover and the verifier, which is reflected in their choice of principles. Their proposed solution consists of using an attestation proxy in order to fulfil the provers goals of constrained disclosure, while also giving comprehensive information to the verifier.
Table 3.1: Attacks that could compromise the security of remote attestation

<table>
<thead>
<tr>
<th>Attack</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerading Attack</td>
<td>The adversary is able to redirect a received attestation request to another</td>
</tr>
<tr>
<td></td>
<td>device, and then relay the response back to the verifier in order to attest</td>
</tr>
<tr>
<td>Measurement Tampering</td>
<td>The adversary is able to tamper with the attestation data, in order to</td>
</tr>
<tr>
<td></td>
<td>remove any discriminating evidence of device compromise.</td>
</tr>
<tr>
<td>Replay Attack</td>
<td>The adversary is able to reuse previous attestation data or messages in</td>
</tr>
<tr>
<td></td>
<td>order to attest itself.</td>
</tr>
<tr>
<td>Time-Of-Check to</td>
<td>The adversary is able to compromise a device in-between the time during</td>
</tr>
<tr>
<td>Time-Of-Use</td>
<td>which attestation is performed.</td>
</tr>
</tbody>
</table>

Table 3.2: Attestation Principles presented by Coker et al. ([14], p. 66–67)

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh Information</td>
<td>“Assertions about the target should reflect the running system, rather than</td>
</tr>
<tr>
<td></td>
<td>just disk images.”</td>
</tr>
<tr>
<td>Comprehensive Information</td>
<td>“Attestation mechanisms should be capable of delivering comprehensive</td>
</tr>
<tr>
<td></td>
<td>information about the target, and its full internal state should be accessi-</td>
</tr>
<tr>
<td></td>
<td>ble to local measurement tools.”</td>
</tr>
<tr>
<td>Constrained Disclosure</td>
<td>“A target should be able to enforce policies governing which measurements</td>
</tr>
<tr>
<td></td>
<td>are sent to each appraiser.”</td>
</tr>
<tr>
<td>Semantic Explicitness</td>
<td>“The semantic content of attestations should be explicitly presented in</td>
</tr>
<tr>
<td></td>
<td>logical form.”</td>
</tr>
<tr>
<td>Trustworthy Mechanism</td>
<td>“Appraisers should receive evidence of the trustworthiness of the attesta-</td>
</tr>
<tr>
<td></td>
<td>tion mechanisms on which they rely. In particular, the attestation architec-</td>
</tr>
<tr>
<td></td>
<td>ture in use should be identified to both appraiser and target.”</td>
</tr>
</tbody>
</table>

Secret Sharing

Some applications of RA may also require a mechanism for secret sharing as part of the attestation process. Secret sharing could be used to establish a secure channel to the attested device. This is especially useful for architectures that provide attestation for secure enclaves, e.g. Intel SGX where it is used to provision secrets to enclaves [29]. A verifier may want to only provision secrets to a process that is running securely inside an enclave. To achieve this the attestation is tied to the secret key used to set up the secure channel.

Secret sharing for mobile devices through remote attestation is mostly unexplored, despite having potential benefits in the form of confidentiality and integrity of sensitive information. Some work has investigated creating secure trusted channels by combining TLS and remote attestation [4]. Such channels could be used to securely communicate with trusted hardware for any platform. However, this approach requires a large trusted computing base, and is therefore very difficult to formally verify.

3.1.3 Samsung Knox Enhanced Attestation V3

Samsung KNOX is an example of a mobile platform that makes use of the TrustZone architecture in order to provide security functionality to enterprises and users. Samsung provides an attestation service [52] for their KNOX devices that attests the integrity of the device firmware and International Mobile Equipment Identity (IMEI) number.

A diagram of the protocol is shown in Figure 3.2. The protocol is initiated by the verifier who requests a fresh nonce from Samsung’s attestation server. The nonce is used to identify
3.1. Background

Figure 3.2: Enhanced Attestation v3 for Samsung KNOX devices. The first row of rectangles contains the names of agents. An agent can be a device, or only a part of a device. This is the case with the agents named TrustZone, Attestation Agent and Application, as they are all different software running on a Samsung Knox mobile device. The protocol is performed from top to bottom in the diagram. Each arrow represents a message that is being passed between two agents. Additional rectangles between messages describe operations performed by the agent.

Each attestation, and is sent to the device along with an API key. The device utilises the TrustZone architecture in order to perform some integrity measurements and sign them with a unique Samsung Attestation Key. The measurements are sent to the Attestation server over a TLS connection, and the server can from the attestation data produce a verdict. Next, the prover signals to the verifier that the attestation has concluded, after which the verifier can request the verdict from the attestation server.

The Samsung Attestation Key is described as being protected by secure hardware, and in the protocol such hardware is described as “TrustZone”. We can therefore assume that the agent named TrustZone is largely implemented as TEE functionality. The Application on the other hand is a normal android application, and is as such completely situated on the REE.

How to determine whether the functionality of the Attestation Agent is implemented either as REE or TEE functionality, or a combination of both, is more difficult. The Attestation Agent will be discussed more when describing the adversary models later in the thesis (see Section 4.2).

3.1.4 Formal Verification

Up until the 1980s, analysis of cryptographic protocols was mostly carried out informally by experts [7]. Despite thorough analysis, many important cryptographic protocols have been shown to contain weaknesses in the past, demonstrating the difficulty in constructing proto-
Listing 3.1: Example Tamarin rule. Line 1 contains the name of the rule: Example_Rule. Line 2 contains the premise. Line 3 contains an action. Line 4 contains the conclusion. If no actions were to be specified, the brackets on line 3 could have been excluded. All text behind double slashes are comments.

```plaintext
rule Example_Rule:
[ Fact1(x1, x2), Fact2(x3) ] // Both Fact1 and Fact2 must be available
--[ Action(x1, x2, x3) ]->
[ Fact3(x1, x2, x3) ] // One instance of Fact3 is produced
```

cols that meet strict security demands. For example, Lowe [37] found a vulnerability in the original Needham-Shroeder protocol, where an adversary could impersonate another agent. Adrian et al. [2] found that common implementations of the Diffie-Hellman key-exchange protocol in TLS were vulnerable because common implementations used the same standardised groups. Beurdouche et al. [8] used formal verification to discover that common TLS implementations contained design flaws that allowed for some attacks. Many more examples exist.

Automated formal analysis has been shown to be very effective in finding vulnerabilities in cryptographic protocols. Formal methods developed to prove security properties in protocols [44] can be implemented in software, allowing for more efficient computer-assisted proofs.

Unfortunately, it is not possible to have a tool that can verify properties for any protocol in a finite amount of time, since such a tool would be able to solve the halting problem: a provably undecidable problem. Instead, automated formal verification of security properties commonly sacrifice termination in order to provide both sound and complete analysis, meaning verification may encounter cases where the properties cannot be proven in a finite amount of time. For such cases, it may be necessary to guide the verification by specifying heuristics or providing helper lemmas.

**Tamarin Prover**

Tamarin Prover [38] is a tool for formal verification of security protocols. Tamarin checks the satisfiability of provided lemmas, and either proves that a lemma is satisfied, or proves it is unsatisfied while providing a counter-example. Since verification is undecidable in general, and Tamarin proofs are both complete and sound, Tamarin cannot guarantee that verification will terminate.

In Tamarin, protocols are modelled as a set of rules. The rules decide how the system state, the combined state of all agents, can change over time. The system state is tracked using typed variables, known in Tamarin as facts. Rules can both consume or produce facts. The premise of the rule contains facts that are required for the rule to be applied, and that will be consumed when the rule is applied. The conclusion of the rule contains facts that are produced by the rule and that will be available during subsequent application of other rules. In addition, each rule can be marked with statements called actions that provide reference points used to specify constraints and lemmas used for proofs. Listing 3.1 shows an example Tamarin rule.

In Tamarin, the adversary is modelled according to the Dolev-Yao [20] model. In the Dolev-Yao model, the adversary has complete control over the communication channel, and is able to listen to messages, modify messages on the channel and send messages of their own. The built-in facts In and Out model such a channel.

Tamarin supports a number of cryptographic primitives, including symmetric and asymmetric encryption, signature schemes and hash functions. Tamarin treats cryptographic primitives symbolically rather than computationally. This leads to an idealized model for cryp-
3.1. Background

tographic primitives, where encrypted messages are assumed to be secret unless the key is known.

3.1.5 Security Properties

Formal Verification requires that the properties to prove are formalised as formulas in a formal language. Common properties regarding the security of cryptographic protocols are authentication properties and secrecy properties, some of which are presented below.

**Authentication**

The hierarchy of authentication specification described by Lowe [36] provides a basis for evaluating the strength of authentication in a protocol. The hierarchy describes four authentication specifications of increasing strength, namely aliveness, weak-agreement, non-injective agreement and injective agreement. Tamarin lemmas for the authentication criteria specified by Lowe are provided in the Tamarin manual [60].

**Aliveness**  Aliveness describes the weakest form of authentication. It states that if agent $A$, acting as the initiator, completes the protocol with agent $B$ as the intended target, then the agent $B$ must have performed the protocol. If a protocol fails to show aliveness of agent $B$ to agent $A$, there is a failure to provide any authentication. Without authentication, an adversary is able to pose as agent $B$.

**Weak agreement**  For the stronger specifications for authentication, the initiator and respondent need to both agree on parts of the protocol. For weak agreement, the agents need to agree that they performed the protocol with each other. Weak agreement specifies that if agent $A$, acting as the initiator, completes the protocol with agent $B$ as the intended target, then the agent $B$ completed their part of the protocol, and was acting with agent $A$. There is however no requirement that they agreed on any of the data communicated during the protocol.$^1$

**Non-injective agreement**  Weak agreement puts no restrictions on the data as seen by both agents, meaning it is possible for both agents to have different views of the data sent in the protocol. Non-injective agreement instead requires that the data as seen by both agents is the same.

**Injective agreement**  Injective agreement additionally requires that the run by agent $B$ happened at a time point earlier than the final message, and additionally that there was no other run of the protocol with the exact same data. This is important for ensuring that messages cannot be replayed. Injective agreement is typically achieved by generating a unique value that identifies each run, a so-called nonce.

The paper by Lowe additionally discusses the notion of recency and how to specify it for authentication properties, and proposes two approaches, one using fresh variables in the form of nonces, and the other using timed requests.

**Secrecy**

Secrecy describes the notion that some information is known only to a select few agents. For security protocols, secrecy is defined as an adversary never being able to learn the contents

---

$^1$In the paper by Lowe, weak agreement does not require that agent $B$ was acting as the responder. However, in the formula that is from the Tamarin manual, agent $B$ is assumed to be the responder. In this thesis, the convention from the Tamarin manual will be followed rather than the definition made by Lowe since this is more convenient to verify. This makes little difference when considering RA, since the protocol is typically very asymmetric, so there is generally no confusion as to what part each agent performs.
of a message or some variable. More formally, we say that a protocol fulfils secrecy of some variable if it is impossible for the adversary to ever read the variable.

This definition of secrecy does not guarantee that the adversary cannot deduce the contents of the variable through other means. For example, if an agent A makes a choice between values X and Y, encrypts their choice with B’s public key and sends the ciphertext over the insecure channel, then the adversary could try both alternatives and encrypt them with the same public key. This will reveal which of X and Y was sent to B, since the ciphertext will match with one of the encryptions made by the adversary. However, secrecy may still hold since the adversary is never able to read the encrypted message directly, which would require B’s private key.
3.2 Related work

Here previous work on formal verification of similar protocols or TrustZone functionality is presented. The discussed papers are summarised in Table 3.3. The table shows which of the discussed papers covers the areas protocol verification, remote attestation and the TrustZone architecture.

3.2.1 Verification of Direct Anonymous Attestation

Direct Anonymous Attestation [13] is a cryptographic signature scheme that allows an attested device to remain anonymous to the verifier. It has been used in Intel’s Enhanced Privacy ID [30] (EPID) for remote attestation on Intel SGX hardware. Whitefield et al. [63] formally verify a number of security properties using Tamarin. Security properties included authenticity and confidentiality properties, as well as the protocol specific properties user-controlled anonymity and user-controlled traceability. Proving the protocol specific properties required four additional lemmas, and included unlinkability which they proved using the observational equivalence property built into the Tamarin prover. In their model, the attestation is provided through a Trusted Platform Module rather than a TEE implementation. Their adversary has control over all communication channels except for the channel between the Trusted Platform Module and the device, which is assumed to be a secure channel.

Their model revealed that the protocol did not fulfill weak agreement in the event that an adversary could leak keys from a different device. In addition, if one device had its key revealed, it was possible for the adversary to leak the shared secret sent to any platform. To rectify these problems, they proposed that the public key of the intended target device should be included as part of the request, and verified that this change prevented the attacks.

In a follow-up paper, Wesemeyer et al. [62] correct their previous definition of user-controlled anonymity, in addition to providing a detailed specification of Direct Anonymous Attestation according to the Trusted Platform Module 2.0 specification. They verify the new specification using Tamarin. The properties verified were correctness, user-controlled linkability, unforgeability, non-frameability, anonymity, user-controlled unlinkability, and finally authentication.

They use a similar system model to the previous paper, but use a simplification when proving the 12 observational equivalence lemmas in order to avoid the otherwise too large state space. The observational equivalence lemmas instead model the Trusted Platform Module and the device as one single unit. Observational equivalence lemmas were used specifically to prove the anonymity and user-controlled unlinkability properties. Finally, they also provide a reference implementation of Direct Anonymous Attestation written in C++ and evaluate its performance.

While the work by Whitefield at al. and Wesemeyer et al. presents formal verification of a specific remote attestation signature scheme, this thesis instead considers general purpose remote attestation protocols and does not consider specific signature schemes. This thesis puts more emphasis on the specification of requirements and adversary models, rather than specific properties of the Direct Anonymous Attestation signature scheme.

3.2.2 EPID-based Remote Attestation in Intel SGX

In a paper by Sardar et al. [54] the remote attestation protocol provided by the Intel SGX was formally verified against a Dolev-Yao adversary with complete control over all channels. Intel SGX provides enclaves which form TEEs for user applications. The application can then attest that it is indeed located on an enclave through a local attestation process. The result of the local attestation can be sent to a remote challenger through the use of the EPID signature scheme, which provides Direct Anonymous Attestation. Their work does not verify the mechanisms of the EPID scheme itself, instead assuming perfect cryptographic primitives.
The verification was performed using ProVerif. Two confidentiality properties were formally proven for the model: the confidentiality of the shared secret sent by the remote challenger, as well as the confidentiality of the derived report key that is used to sign the attestation data during the local part of the attestation. Their verification proves that the adversary cannot learn either the secret or the report key. In addition, during the process of formalising the model, some discrepancies were found in Intel documentation describing the attestation procedure.

3.2.3 Formal Verification of Remote Attestation in Intel TDX

In a subsequent paper by Sardar et al. [53], attestation on the new architecture TDX proposed by Intel was formally verified using ProVerif. TDX provides a TEE at the virtual machine level, intended to be used to provide stronger isolation for cloud computing. With TDX the virtual machine monitor can be considered untrusted, yet the Trust Domains can still provide confidentiality and integrity to the hosted virtual machines.

Verification of RA on TDX was performed to prove confidentiality of the shared secret, as well as injective agreement for two parts of the protocol using two different properties. The first property proves that any quote that is verified by the challenger must have been generated by the hardware that generates the report. The second property proves that the report sent from the TDX Module matches in part with the report that is accepted by the Quoting Enclave. In their adversarial model, they assume that the virtual machine monitor of the platform is untrusted, but that otherwise the platform components (consisting of the Quoting Enclave, the guest Trust Domain, the TDX module and the CPU hardware) as well as the challenger are trusted. Similarly to the SGX paper, some discrepancies were also found in the Intel documentation of TDX while formalising the model.

The remote attestation protocols on both Intel SGX and Intel TDX contain many platform-specific steps, that may not generalise to other remote attestation protocols. This thesis differs by providing remote attestation protocols that are as general as possible. In addition, this thesis formally verifies a remote attestation protocol on the ARM TrustZone platform, which differs in many ways to the Intel platforms.

3.2.4 Verified Hybrid Attestation

Nunes et al. [16] present a formally verified architecture for RA targeting low-end embedded devices. Their architecture implements a hybrid RA protocol, meaning RA is not performed entirely by trusted hardware, but instead RA is implemented partially in software. This in turn allows RA with minimal hardware support, resulting in a more affordable solution. They implemented and evaluated the performance of their design on a MSP430 microcontroller. The remote attestation was designed to attest the memory contents of the prover using a signature scheme. Hardware support is added to the microcontroller that perform the integrity measurements, and signs them using a key read from an exclusive read-only memory.

Formal verification is performed over the entire architecture. They specified a number of security sub-properties as invariants in Linear Temporal Logic. They then used the model checker NuSMV to prove that all sub-properties held for all of the hardware modules. Next, they proved that the high-level security properties followed form the sub-properties. The high-level security properties consisted of two properties: the first describing that the integrity measurement of the prover should accurately describe the memory contents at the time of attesting, the other a security game asserting that the adversary should never be able to forge attestation data that does not reflect the memory contents of the prover.

In a subsequent paper, Nunes et al. [17] show how to also provide a Proof of Execution as part of remote attestation, by utilizing the previously developed VRASED architecture. A proof of execution proves that a certain software was executed on the platform, and that the result was not modified.
3.2.5 Formally Verified TrustZone TEE

Sun and Lei [59] design and formally verify a TEE implementation for TrustZone by designing a secure monitor that can protect the memory shared by the REE and the TEE. Their TEE was verified both with respect to its code and with respect to secrets stored on the TEE. The implementation shows that TrustZone can be used to protect secrets from a strong adversary with complete control over the REE as well as persistent storage and unprotected memory.

3.2.6 Verification of TLS 1.3 Specification

Cremers et al. [15] perform analysis of a draft for the TLS 1.3 specification. They performed verification using Tamarin. Verification showed that the specification met the requirements for key-exchange. However, a vulnerability was found when considering the proposal for a delayed client authentication mechanism. Both secrecy and authentication properties were considered. The verified secrecy properties were secrecy of session keys and data keys. The verified authentication properties were agreement on nonces and transcripts between clients and peers, as well as between server and client.

<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Protocol Verification</th>
<th>Remote Attestation</th>
<th>TrustZone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whitefield et al.</td>
<td>2019</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wesemeyer et al.</td>
<td>2020</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sardar et al.</td>
<td>2020</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sardar et al.</td>
<td>2021</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nunes et al.</td>
<td>2019</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nunes et al.</td>
<td>2020</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Sun and Lei</td>
<td>2020</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cremers et al.</td>
<td>2016</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adversary Models and Security Requirements

The following chapter presents the adversary models and attestation goals that form a basis for the formal analysis. First, the adversary models which describe the capabilities of the adversary are described in Section 4.2. Next, we describe goals that attestation protocols should fulfil in Section 4.3. This forms the basis for formulating security properties used during formal verification. Next, some additional definitions for authentication properties are introduced in Section 4.4. This is done in order to specify appropriate authentication properties, including one-way authentication properties, along with a stricter version of injective agreement here referred to as recent injective agreement. Finally, the attestation goals are mapped to security properties in Section 4.5.

4.1 Terminology

We refer to the device which attests itself as the prover, or simply as the device. We assume each prover contains a separate TEE and REE, and that the TEE is capable of performing an attestation. We refer to the party which intends to gather information about some device as the verifier, regardless if the verifier produces the final verdict or if it is produced by a remote attestation server. Additionally, for protocols that make use of an attestation server for producing the verdict, we refer to such an agent as the server. The KNOX protocol uses software dedicated to remote attestation. This software is referred to as the Attestation Agent. The Attestation Agent is situated on the prover, and acts as a proxy for the attestation request made by the REE.

4.2 Adversary Models

This thesis will consider two adversary models, one with a very strong adversary with complete control over most communication channels. The other adversary is much weaker, and can only control the communication to and from the user application situated on the prover. Both models consider an adversary that follows the Dolev-Yao model for untrusted channels.

Two adversary models are used because different models may be more appropriate in certain scenarios. For example, a common assumption when modelling platforms with trusted execution environments is to assume a fully compromised REE, while we assume the TEE
4.2. Adversary Models

Figure 4.1: Adversary model A. The adversary controls all communication channels. The Attestation Agent and REE are both assumed to be compromised.

Figure 4.2: Adversary model B. The adversary controls channels in red. Channels in black are fully secure.

will remain secure against any software-based attacks. Proving protocol security for such an adversary model will ensure correct operation even during operating system compromise. However, the Samsung KNOX platform provides additional security [58], such as Trusted Boot and security services running on the TEE. These security services include Real-time Kernel Protection [6], which protects the kernel from run-time attacks. Because of the additional security provided by the KNOX platform, we also consider an adversary model where all parts of the device are assumed to be completely secure, including both the REE and TEE of
the platform. Instead, only network channels are vulnerable to the adversary. This adversary more closely models the security assumptions of the KNOX platform.

For this thesis we unfortunately do not have access to any implementation or documentation of the Attestation Agent. Instead, we make assumptions about the capabilities of the Attestation Agent. Depending on the model, we include or exclude the Attestation Agent in the capabilities of the adversary.

4.2.1 Adversary Model A

For adversary model A, we consider a strong adversary with complete control over all communication channels. Additionally, we consider the adversary to have control over the REE of the prover, as well as the Attestation Agent. We argue that any remote attestation protocol that hopes to detect the presence of an attack on the prover must be able to function during such attacks. In the case of an adversary acting as privileged malware running in the REE of the prover, such adversary could leak or inject their own messages to and from the TEE. Therefore, we have decided to also model the channel between the REE and the TEE as a communication channel controlled by the adversary. Otherwise, we assume that the TEE, verifier and server are all secure. An overview of this adversary model is shown in Figure [4.1]. We note that such an adversary is stronger than the adversary typically considered for the KNOX platform. For the KNOX platform, privileged applications are generally considered protected, and thus trusted.

4.2.2 Adversary Model B

Adversary model B considers a weaker adversary that only controls the communication channel between the REE and the Verifier and between the REE and the Server. All other channels are assumed to be secure channels, including the channel between the server and verifier. This is possible to establish over a network using existing network protocols, for example through the use of Transport Layer Security (TLS). Adversary model B is shown in Figure [4.2].

This models a system where we exclude any malicious privileged software from the REE, since such software could intercept communication with the Attestation Agent or the TEE. Run-time attacks against the attestation application on the REE are also excluded, since run-time attacks could also be used to intercept communication with the Attestation Agent. This essentially models a benign prover, while still assuming a strong Dolev-Yao network adversary listening to the REE.

4.3 Attestation Goals

We now proceed to present a list of attestation goals that we have identified as important for remote attestation protocols. Each goal is described and motivated below. The protocols will be evaluated against these goals and their corresponding security properties. In Section 4.3 the attestation goals are summarised and motivated based on their relation to the requirements presented in Section 3.1.2.

G1: Executability of Protocol

For any protocol we want to ensure that the protocol steps can be executed in the order intended by the developers. Specifically, we want to ensure that it is possible to execute the protocol without any help from any agents such as the adversary. We explicitly state this as a verification goal to be proven in order to give confidence in our models, and in the developed protocols.
4.3. Attestation Goals

**G2: Authenticity of TEE**
For remote attestation to succeed from the point of view of the verifier, it is crucial that the verifier can guarantee the origin of the attestation data. If not, there is no guarantee that an attestation report was not forged by a malicious party, or came from a different device than intended by the verification. Lack of authentication opens up for masquerading attacks, allowing a malicious device to attest itself through the use of a helper device.

**G3: Recentness of Attestation**
The attestation needs to be performed on new values, or else an attacker can reuse a valid attestation response. If the attacker can compromise the device and then reuse existing attestation messages, it could trick the verifier into thinking it was still secure. It is also an issue if the prover can store integrity measurements, and then use them to provide valid attestation data after a compromise has taken place. For any RA protocol, we should require that integrity measurements are at least performed after the verifier has first issued the challenge to the prover.

**G4: Integrity of Attestation Data**
The attestation data should have been produced by the TEE through some trusted mechanism. An attestation protocol must ensure that integrity of the attestation data is preserved, and cannot be forged or modified by the adversary. If the attestation data can be modified by the adversary in transit, the adversary could try to remove any discriminating evidence of device compromise from the attestation data, thereby creating a valid attestation report for an untrustworthy device. In such a case the attestation data cannot be trusted.

**G5: Integrity of Attestation Verdict**
When an attestation server is used, we should prove that the server and verifier agree on what verdict was given during a given attestation in addition to proving the integrity of the attestation data. The adversary should not be able to tamper with the verdict itself in order to trick the verifier into trusting an untrustworthy device.

**G6: Authenticity of Verifier**
For the prover, it is desirable to only provide attestation data to trusted parties. This especially applies for RA protocols that may disclose information about OS version, firmware version or running applications. In addition, some attestation procedures rely on measuring run-time information such as memory contents of the REE. Such run-time information could potentially leak application data. In order for the prover to have trust in the attestation procedure, the prover should be able to ensure that attestation data will not be leaked to an untrusted party. To achieve this, the prover should be able to authenticate the agent for which the attestation will be performed.

In addition, it is also desirable to protect from Denial-of-Service attacks through repeated attestation requests. In the scenario that the request is initiated by the verifier, the prover would want to know that the request originated from a trusted verifier, or else an attacker could send repeated attestation requests to the device with the intent of disrupting service on the device. This is needed since performing run-time integrity measurements can be quite detrimental to performance. It is also important that an adversary cannot repeat requests made by the verifier, since this could be used to overwhelm the prover with attestation requests.
4.3. Attestation Goals

Table 4.1: Attestation goals

<table>
<thead>
<tr>
<th>Goal</th>
<th>Attestation Goal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Executability of Protocol</td>
</tr>
<tr>
<td>G2</td>
<td>Authenticity of TEE</td>
</tr>
<tr>
<td>G3</td>
<td>Recentness of Attestation Data</td>
</tr>
<tr>
<td>G4</td>
<td>Integrity of Attestation Data</td>
</tr>
<tr>
<td>G5</td>
<td>Integrity of Attestation Verdict</td>
</tr>
<tr>
<td>G6</td>
<td>Authenticity of Verifier</td>
</tr>
<tr>
<td>G7</td>
<td>Confidentiality of Shared Secret</td>
</tr>
<tr>
<td>G8</td>
<td>Integrity of Shared Secret</td>
</tr>
</tbody>
</table>

Table 4.2: Mapping of Attestation Principles presented by Coker et al. and attestation goals

<table>
<thead>
<tr>
<th>Principle</th>
<th>Attestation Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh Information</td>
<td>G3 &amp; G4 &amp; G5</td>
</tr>
<tr>
<td>Comprehensive Information</td>
<td>out of scope</td>
</tr>
<tr>
<td>Constrained Disclosure</td>
<td>G6*</td>
</tr>
<tr>
<td>Semantic Explicitness</td>
<td>out of scope</td>
</tr>
<tr>
<td>Trustworthy Mechanism</td>
<td>G2 &amp; G6</td>
</tr>
</tbody>
</table>

Table 4.3: Protocol attacks and what attestation goal they are covered by

<table>
<thead>
<tr>
<th>Attack</th>
<th>Attestation Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerading Attack</td>
<td>G2</td>
</tr>
<tr>
<td>Measurement Tampering</td>
<td>G4 &amp; G5</td>
</tr>
<tr>
<td>Replay Attack</td>
<td>G3 &amp; G6</td>
</tr>
<tr>
<td>TOCTOU</td>
<td>out of scope</td>
</tr>
</tbody>
</table>

**G7: Confidentiality of Shared Secret**

For a protocol that implements secret sharing, the secret would only be known to the verifier and the TEE of the attested device. This ensures that data can be securely provisioned to trusted devices, without there being any way for the data to leak to an adversary.

**G8: Integrity of Shared Secret**

The adversary should not be able to modify or replace the shared secret, or else this would compromise the security. Specifically, once the protocol finishes, the verifier who provisioned the secret should be able to be sure that the secret is securely provisioned to the TEE of the attested device. If the adversary can tamper with the provisioned secret, then it is no longer possible to share a key in order to establish a secure channel. This is because the adversary could replace the key with a key they know, and thereby listen to the channel.

**Confidentiality of Attestation Data**

Confidentiality of attestation data is an important requirement of remote attestation, since remote attestation may otherwise reveal sensitive information about the prover device state. However, confidentiality of attestation data was not formally verified in this thesis due to some modelling challenges. The reason is explained in-depth in Section 7.2.2.

**Summary**

The attestation goals are summarised in Table 4.1. Table 4.2 shows the mapping from the properties presented by Coker et al. [14] (see Section 3.1.2) and to the attestation goals. Table 4.3 shows how the attestation goals relate to the remote attestation attacks.
As seen in Table 4.2, attestation goals G2 - G6 can all be related to the attestation principles presented by Coker et al. [14]. In addition, all attestation attacks listed in Table 3.1 are covered by an attestation goal as shown by Table 4.3.

The asterisk next to goal G6 in Table 4.2 is meant to indicate that goal G6 on its own is not enough to provide constrained disclosure, and since confidentiality of attestation data is not explored for the thesis.

TOCTOU vulnerabilities were deemed out of scope for the protocols. Completely protecting against TOCTOU vulnerabilities is hard at a protocol level, since we will always have some latency over the network, which provides a window of time for the adversary to infect a device and then erase its presence before the next attestation. Completely protecting against TOCTOU vulnerabilities instead has to be provided by the measurement procedure itself, which is out of scope for this thesis. For an example of how TOCTOU protection can be provided through an attestation mechanism, see the work by Nunes et al. (2020) [18].

The attestation principles Comprehensive Information and Semantic Explicitness have to do with the integrity measurements and how they are presented to the verifier. Exact attestation mechanisms are deemed to be out of scope for this thesis. The attestation goals that do not relate to any attestation principle or attack are goal G1, which is not attestation specific, and goals G7-G8 which are related to secret sharing which is not covered by the attestation principles.

4.4 Authentication Definitions

In order to provide exact definitions for the authentication properties used, we define two additional authentication properties. First, we relax the agreement properties used by Lowe [36] in order to describe one-way authentication. We motivate why the authenticity requirements for goals G2-G5 are better modelled using such definitions. Finally, we explicitly state the assumption made when proving recency for goal G3.

4.4.1 One-way Authentication

In Lowe’s Authentication Hierarchy (see Section 3.1.5) all agreement properties require that the responder agrees with the initiator on who the initiator was. While this requirement is important for key exchange protocols, where an attacker with the ability to mislead one of the agents could be devastating and compromise the security, we argue it is not always necessary for the purpose of remote attestation. For remote attestation, we consider the goals of the verifier and the prover to be separate, and although they should agree on the attestation data for a particular request, they need not agree on the identity of the responding agent.

For an example where such differences matter, consider the following. The verifier first sends an attestation request intended for a certain prover. An adversary listens to the request, removes it from the channel and sends its own request to the prover. The prover would then believe it performed the request with the adversary. However, as long as the verifier can be certain the prover performed the request, the verifier would still be able to verify the prover as long as the final result can be seen by the verifier. If we were to use the strict definition presented by Lowe, we would have a violation of the security property for weak agreement in such a scenario, since the prover would believe it performed the request with the adversary, while the verifier believed it performed the request with the prover. Yet it is not clear how the violation of the security property would translate into an attack. To differentiate these cases, we specify whether we are using the one-way or two-way authentication property. We introduce definitions for one-way non-injective agreement and one-way injective agreement to indicate that the two agents do not need to agree on who the initiator was. Authentication properties not explicitly stated as one-way implicitly refer to the standard two-way authentication properties. One-way non-injective and one-way injective agreement are defined as follows:
4.4. Authentication Definitions

**Definition: one-way non-injective agreement**  A protocol guarantees one-way authentication with non-injective agreement to an agent $A$ acting as the initiator if $A$ completes a run of the protocol apparently with $B$, then there was an agent $B$ acting as responder not necessarily with $A$, and $A$ and $B$ agreed on the data for this run.

**Definition: one-way injective agreement**  A protocol guarantees one-way authentication with injective agreement to an agent $A$ acting as the initiator if $A$ completes a run of the protocol apparently with $B$, then there was an agent $B$ acting as responder not necessarily with $A$, and $A$ and $B$ agreed on the data for this run, and for each run of $A$ there was a unique run by $B$.

4.4.2 Recent Injective Authentication

Recentness is discussed in the paper by Lowe, and two approaches to specify recentness are presented. Either through using a fresh value in each run that both agents agree upon, thus proving that the protocol was ran recently, or through the use of timing to prevent requests from taking too long.

Using timing as a criterion is infeasible for the method used in this work, since Tamarin does not quantify time. On the other hand, using fresh values is necessarily protocol dependent, and could thus require modification of the protocol itself. Instead, we propose using a slightly modified injective agreement property that also guarantees that the run by the responder happened after the first message of the initiator. This provides the same guarantee that using a fresh value does, namely that the protocol was run recently by the responder. Furthermore, if the protocol already makes use of fresh values, then such property would trivially hold in case injective agreement also holds. We refer to this modified injective agreement property as recent injective agreement. We also define the corresponding one-way authentication property one-way recent injective agreement. When security properties are defined later in Section 4.5 only recent injective aliveness will be used. The definition of (two-way) recent injective agreement is only provided for completeness. We define the properties as follows:

**Definition: recent injective agreement**  A protocol guarantees recent injective agreement to an agent $A$ acting as the initiator if $A$ completes a run of the protocol apparently with $B$, then there was an agent $B$ acting as responder apparently with $A$, and $A$ and $B$ agreed on the data for this run, and the run by $B$ happened after the first message by $A$, and for each run of $A$ there was a unique run by $B$.

**Definition: one-way recent injective agreement**  A protocol guarantees one-way authentication with recent injective agreement to an agent $A$ acting as the initiator if $A$ completes a run of the protocol apparently with $B$, then there was an agent $B$ acting as responder not necessarily with $A$, and $A$ and $B$ agreed on the data for this run, and the run by $B$ happened after the first message by $A$, and for each run of $A$ there was a unique run by $B$.

4.4.3 Authentication Hierarchy

Relation 4.1 illustrates the relative strength of the authentication properties. The one-way authentication properties follow from the corresponding two-way authentication properties. Additionally, the one-way authentication properties follow the same hierarchy as two-way authentication. Do note that even the strongest one-way authentication property does not imply even weak agreement. Also note that the corresponding one-way authentication property for weak agreement is equivalent to aliveness.
4.5 Security Properties

For each of the attestation goals listed in Section 4.3, we select a formal security property. The protocols will be evaluated against lemmas corresponding to each security property. The attestation goals and their corresponding security properties are listed in Table 4.4. Each goal has exactly one security property mapped to it. We also specify the perspective with which we will define the security properties, shown by the columns Target and View. The Target column shows what agent is to be authenticated for the authentication properties, and what variable is the target for the secrecy property for Goal G7. The View column shows what agent the target should be authenticated to for authentication properties, and what agent believes the variable to be secret for Goal G7. For each security property we provide a motivation below.

G1: Correctness Trace

Proving protocol executability is important in order to ensure that the protocol can be performed as intended. It also makes sure that other verification goals do not vacuously hold due to an inability to perform the protocol steps. We have defined a correctness trace as the following:

Table 4.4: Verification goals and their corresponding security properties. The Target column shows what agent should be authenticated, or what variable should be kept secret. The View column shows which agent the security property is applied by.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Verification Goal Name</th>
<th>Security Property</th>
<th>Target</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Executability of protocol</td>
<td>Correctness Trace</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>G2</td>
<td>Authenticity of TEE</td>
<td>One-way Injective Agreement</td>
<td>TEE</td>
<td>Verifier</td>
</tr>
<tr>
<td>G3</td>
<td>Recentness of Attestation Data</td>
<td>One-way Recent Injective Agreement</td>
<td>TEE</td>
<td>Verifier</td>
</tr>
<tr>
<td>G4</td>
<td>Integrity of Attestation Data</td>
<td>One-way Non-Injective Agreement</td>
<td>TEE</td>
<td>Verifier OR Server</td>
</tr>
<tr>
<td>G5</td>
<td>Integrity of Attestation Verdict</td>
<td>One-way Injective Agreement</td>
<td>Server</td>
<td>Verifier</td>
</tr>
<tr>
<td>G6</td>
<td>Authenticity of Verifier</td>
<td>Injective Agreement</td>
<td>Verifier</td>
<td>REE</td>
</tr>
<tr>
<td>G7</td>
<td>Confidentiality of Shared Secret</td>
<td>Secrecy</td>
<td>shared secret</td>
<td>Verifier</td>
</tr>
<tr>
<td>G8</td>
<td>Integrity of Shared Secret</td>
<td>Non-Injective Agreement</td>
<td>TEE</td>
<td>Verifier</td>
</tr>
</tbody>
</table>

1Security property defined by Lowe. [36]
2Security property defined in this work.
4.5. Security Properties

**Definition: Correctness Trace** There must exist a way to execute the protocol so that each protocol step has been executed at least once, with each protocol step containing matching variables, and while each agent has started exactly one run of the protocol.

By also ensuring that each agent performs the protocol in a stateful manner, that is each agent only performs a protocol step if previous steps have been executed, then such a correctness trace also ensures that each agent completed the protocol fully from beginning to end with no repeated steps. This gives a good indication that the protocol can be performed as specified, without requiring help from an adversary.

**G2: One-way Authentication of TEE seen from Verifier with Injective Agreement**

Authenticity of TEE can be proven using authentication properties. Authentication properties give guarantees that the responding agent was the intended attestation target. In order to protect against replay attacks that reuse old attestations, we should require an injective authentication property.

As motivated in Section 4.4.1, two-way authentication is not necessary for the purpose of attestation. We do not require that the TEE agree with who the initiator was. Thus, the strongest authentication property we require is one-way injective agreement, and not (two-way) injective agreement. Using two-way authentication may lead to false positives when searching for attack traces, for cases when the TEE cannot properly differentiate between different verifiers. The one-way property avoids this false-positive without requiring any restrictions on the model itself.

**G3: One-way Authentication of TEE seen from Verifier with Recent Injective Agreement**

While injective agreement guarantees that a protocol run was unique, it does not guarantee that the protocol runs were recent. To prove G3 we use our definition of recent injective aliveness, as specified in Section 4.4.2. We use aliveness, rather than agreement for the same reason as with goal G2: the TEE does not need to agree on who the request is performed with.

Goal G3 and G2 have overlapping security properties, since proving recent injective aliveness necessarily proves injective aliveness.

**G4: One-way Authentication of TEE seen from Verifier or Server with Non-Injective Agreement**

Integrity of attestation data can be proven using non-injective authentication, since according to the definition such a property will ensure that both parties must agree on the data seen. For attestation, we can specify that the attestation data seen should agree with the attestation data that was produced by the intended prover. We specify the property from the perspective of the agent that is responsible for producing the verdict based on the attestation data: either the verifier or the server.

**G5: One-way Authentication of Server seen from Verifier with Injective Agreement**

If a server is responsible for producing the attestation verdict, we additionally require injective aliveness between the verifier and the server. This is to ensure that the verifier and server agree on the attestation verdict, and that the verdict was not replayed.

**G6: Two-way Authentication of Verifier seen from REE with Injective Agreement**

From the prover perspective we require injective authentication in order to protect against replay attacks. If the adversary can replay requests, they could perform Denial-of-Service
through repeated requests. We also require two-way authentication rather than one-way authentication since the prover should not respond to requests that were originally made to a different prover.

Unlike the properties that are defined from the perspective of the verifier or the server, where we consider the TEE to represent the true identity of the device, for Goal G6 we consider the perspective of the REE rather than the TEE. The motivation being that we expect the REE to be the agent with knowledge of the verifier, rather than the TEE. In addition, the decision to perform or not perform the attestation for a particular verifier must be made early in the protocol. During normal operation, the application that initiates the attestation on the prover would be implemented by the REE.

**G7: Secrecy seen from Verifier**

Secrecy should hold for any secret once it has been provisioned to a TEE. We only guarantee secrecy to the verifier, since the verifier provides the secret. Once the TEE receives some secret, it cannot guarantee that the verifier has not previously leaked the secret that was sent, and so we cannot guarantee secrecy from the perspective of the TEE.

**G8: Two-way Authentication of TEE seen from Verifier with Non-Injective Agreement**

Non-injective agreement ensures that the secret remains unaltered when provisioned to the TEE, and additionally guarantees that the TEE knew what verifier provisioned the secret.
In this chapter, the protocol development process is briefly described, followed by a presentation of the developed RA protocols. A summary of the developed protocols is shown in Table 5.5.

5.1 Protocol Development

Protocols were developed in an iterative manner with regular evaluation in the Tamarin prover. Initial models were made based on intuition, and were evaluated against weak versions of the lemmas. If lemmas were disproved by Tamarin, the protocol was revisited. Often, the initial intuition was incorrect and the protocol had to be redesigned. By utilising the interactive mode of the Tamarin prover, attack traces could be generated. These traces provided examples for how the lemmas could be broken by the adversary, which simplified reasoning about the weaknesses in the protocol. By following the attack trace, it was possible to deduce whether the security property violation was due to insufficient or unsatisfactory usage of cryptographic primitives, or due to modelling or specification errors.

Once the lemmas could be proved, stronger versions of the lemmas were successively added, until all desired properties could be proved. This resulted in strong protocol guarantees as part of the protocol development process.

5.2 Two-Agent Protocol

Figure 5.1 shows the developed two-agent RA protocol, featuring only a prover and a verifier. Table 5.1 describes each of the steps in the figure.

The first part of the protocol involves a negotiation between the agents in order to establish a fresh nonce to be used during attestation. Both the verifier and the prover each generate a unique value for each run; the verifier generates nonce nonce1 and the prover generates nonce nonce2. The nonces are combined in step 5, and this resulting nonce is the nonce that is used to uniquely identify the attestation. The messages from the verifier are signed using the verifier’s private key in order to ensure their authenticity and integrity to the prover.

Messages that are written as $\text{sign}_X(...)$ are messages that are signed with the private key of agent $X$. 

27
5.2. Two-Agent Protocol

Figure 5.1: Two-agent RA protocol. Prover is shown as two agents, with the REE and TEE implementations separated. Text above the arrows are names of the requests, while the text below the arrows are variables that are contained in the request. Steps 1–6 perform the nonce negotiation, while steps 7–10 attest the prover.

Table 5.1: Description of steps in two-agent RA protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Verifier generates a random nonce</td>
</tr>
<tr>
<td>2</td>
<td>Verifier sends the nonce to the prover and signs the message</td>
</tr>
<tr>
<td>3</td>
<td>The REE of the prover generates its own nonce</td>
</tr>
<tr>
<td>4</td>
<td>The REE sends both nonces to the verifier</td>
</tr>
<tr>
<td>5</td>
<td>The verifier combines both nonces into one</td>
</tr>
<tr>
<td>6</td>
<td>The combined nonce is signed and sent back to the prover</td>
</tr>
<tr>
<td>7</td>
<td>The REE of the prover receives the combined nonce, verifies that it matches the earlier request, and forwards the attestation request to the TEE</td>
</tr>
<tr>
<td>8</td>
<td>The TEE of the prover performs some integrity measurements and creates a quote from the measurements and the nonce</td>
</tr>
<tr>
<td>9</td>
<td>The quote is signed with the attestation key and sent back to the REE</td>
</tr>
<tr>
<td>10</td>
<td>The REE forwards the signed quote to the verifier</td>
</tr>
</tbody>
</table>
Once a nonce has been agreed upon, the TEE of the prover performs some measurements of its state, signs the measurements along with the negotiated nonce using its attestation key before sending it to the verifier. Because the public attestation key is known to the verifier, the verifier can be sure the measurements were performed on the specified TEE by verifying the signature. In addition, since both the verifier and the prover generate a unique nonce as part of each run of the protocol, both agents can be sure that each protocol run is unique.

Sending the attestation measurements to the verifier could optionally be sent over a secure channel in order to provide confidentiality of the attestation data. Note that depending on the adversary model used, the adversarial model may assume a compromised REE. For a compromised REE, any data that passes through the REE could be leaked, thus we cannot provide confidentiality with such model.

5.2.1 Secret Sharing

The two-agent protocol was extended to provide a mechanism for secret sharing. This extended protocol is shown in Figure 5.2. Table 5.2 describes the protocol steps that are different from the original protocol.

Steps written as \( \texttt{aenc}_X \{Y\} \) describe asymmetric encryption of \( Y \) using the key \( X \), while steps written as \( \texttt{adec}_X \{Y\} \) describe asymmetric decryption of \( Y \) using the key \( X \). Similarly, \( \texttt{senc}_X \{Y\} \) describe symmetric encryption of \( Y \) using the key \( X \), while \( \texttt{sdec}_X \{Y\} \) describe symmetric decryption of \( Y \) using the key \( X \).

The initial attestation remains the same. However, the quote signed by the TEE contains some additional fields. As part of the attestation, the TEE now generates a symmetric key and encrypts the key using the public key of the verifier. The symmetric key is later used as the encryption key when sending a secret from the verifier to the prover. A symmetric key is used rather than an asymmetric key because asymmetric keys are slower to generate. Generating new asymmetric keys such as RSA takes a lot of computation, and may require careful implementation of check-pointing \[47\] in order to not suspend OS functionality for too long, which would interfere with the user interface.

The TEE also generates a new nonce \textit{sharing nonce} specifically for the purpose of secret sharing. The encrypted sharing key, the sharing nonce and the verifier public key are all included as part of the quote, along with the nonce and the measurements.

Once the verifier receives the quote, it can first verify the measurements, and then decide to provision the secret based on the verdict. If it decides to provision a secret to the TEE, it decrypts the sharing key using its private key and uses the sharing key to encrypt the secret before sending a provisioning request to the prover. The provisioning request includes both the encrypted secret and the sharing nonce, and the message is signed using the verifier’s private key. Once the REE has forwarded the message to the TEE, the TEE can decrypt the secret. The TEE then generates a unique tag so that future requests can refer to this secret. The TEE signs a message which includes the nonce, tag and the sharing nonce using the attestation key in order to confirm that the secret was provisioned. When the verifier receives this message, it can be sure that the secret has been securely stored by the TEE, by verifying that the signature matches the attestation key that performed the attestation.
Figure 5.2: Two-agent RA protocol with secret sharing. Messages 1–10 are similar to the simple two-agent protocol. The secret $x$ is chosen by the verifier in step 11 and is provisioned to the TEE in step 14.
### Table 5.2: Description of steps in two-agent RA protocol with secret sharing

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–6</td>
<td>Same as without secret sharing</td>
</tr>
<tr>
<td>7</td>
<td>The REE of the prover receives the combined nonce, verifies that it matches the earlier request, and forwards the attestation request to the TEE, now also with the public encryption key of the verifier</td>
</tr>
<tr>
<td>8</td>
<td>The TEE of the prover performs some integrity measurements in addition to generating a unique symmetric encryption key and encrypting it with the public key of the verifier. The nonce, measurements, verifier public key and a newly generated nonce are all added to a quote</td>
</tr>
<tr>
<td>9–10</td>
<td>Same as without secret sharing</td>
</tr>
<tr>
<td>11</td>
<td>The verifier makes a judgement based on the received integrity measurements, and if the prover is accepted, the sharing key is decrypted using the private key of the verifier, and a secret is encrypted using the sharing key</td>
</tr>
<tr>
<td>12</td>
<td>The verifier makes a request to store the secret on the TEE containing the nonce, encrypted secret and sharing nonce. The request is signed using the verifier private key</td>
</tr>
<tr>
<td>13</td>
<td>The REE forwards the request to the TEE</td>
</tr>
<tr>
<td>14</td>
<td>The TEE ensures that the sharing nonces are the same, and then decrypts the encrypted secret</td>
</tr>
<tr>
<td>15</td>
<td>The TEE signs a message containing the nonce and sharing nonce and sends it to the REE</td>
</tr>
<tr>
<td>16</td>
<td>The REE forwards the message to the verifier</td>
</tr>
</tbody>
</table>

### 5.3 Three-Agent Protocol

For the three-agent protocol, a separate server is responsible for evaluating the measurements and the signature. The developed protocol is shown in Figure 5.3. Table 5.4 describes the protocol steps. The prover now sends the measurements to the server instead of the verifier. The verifier can then request a verdict from the server. Thus, the attestation data is hidden from the verifier, while the server can still provide the verifier with enough information to evaluate the prover’s state.

This protocol has a similar structure to the Samsung KNOX Enhanced Attestation protocol (see Section 3.1.3). The main differences are the addition of nonce negotiation as part of the protocol, and the addition of a device ID in the final verdict sent from the server. Another difference is that in the Knox protocol, the server is responsible for generating the nonce used for identifying the session. In this protocol, the verifier and prover generate their own nonces.

In both the developed three-agent protocol and the Knox protocol, the attestation data is sent to the server and not to the verifier. The verifier therefore relies on the server to make a verdict about the prover state based on the attestation data. In the developed protocol, the server only gives a binary verdict: either the prover is accepted, or rejected. This verdict is sent along with the prover ID and session nonce in the final step of the protocol.
Three-Agent Protocol

Figure 5.3: Three-agent RA protocol. Attestation data is sent to the server instead of the verifier in step 10. The verifier receives a verdict from the server in step 15.

Table 5.3: Description of steps in three-agent RA protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–9</td>
<td>Same as for two-agent protocol, except requests from verifier now contain a server ID</td>
</tr>
<tr>
<td>10</td>
<td>The REE forwards the signed quote to the server</td>
</tr>
<tr>
<td>11</td>
<td>The server makes a verdict based on the received integrity measurements</td>
</tr>
<tr>
<td>12</td>
<td>The server notifies the REE of the verdict made</td>
</tr>
<tr>
<td>13</td>
<td>The REE notifies the verifier that the server has made a verdict</td>
</tr>
<tr>
<td>14</td>
<td>The verifier requests to see the verdict made by the server, and sends the nonce. The request is signed with the verifier public key</td>
</tr>
<tr>
<td>15</td>
<td>The server responds by sending a message containing the nonce, prover ID and the verdict</td>
</tr>
</tbody>
</table>
5.3. Three-Agent Protocol

5.3.1 Secret Sharing

Secret sharing was added to the three-agent protocol through similar mechanisms as for the two-agent protocol. The developed protocol is shown in Figure 5.4 Table 5.3 describes the protocol steps.

The sharing key is still encrypted with the verifier public key. In the three-agent version, the encoded sharing key and sharing nonce are still part of the attestation data, but the data is sent to the server rather than the verifier. This is seen in step 10 of the protocol, where the quote contains the encoded sharing key and sharing nonce. The verifier then receives the sharing key and sharing nonce from the server while receiving the verdict in step 16.

After provisioning, the verifier ensures that the signature matches the attestation key used during attestation by asking the server to verify the signature. This is necessary due to the assumption that only the server knows the public attestation key used by the prover. For example, in the scenario that the device manufacturer provisions attestation keys to devices, and then provides the attestation as a service, only the manufacturer would know what keys were provisioned for each device.

Table 5.4: Description of steps in three-agent RA protocol with secret sharing

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–9</td>
<td>Same as for two-agent protocol with secret sharing, except requests from verifier now contain a server ID</td>
</tr>
<tr>
<td>10–14</td>
<td>Same as for three-agent protocol</td>
</tr>
<tr>
<td>15</td>
<td>The server constructs a message containing the nonce, prover ID, verdict, verifier public key and sharing nonce, all from the quote received from the prover</td>
</tr>
<tr>
<td>16</td>
<td>The message is sent to the verifier. The message is signed using the server private key</td>
</tr>
<tr>
<td>17–22</td>
<td>Same as for two-agent protocol with secret sharing steps 11–16</td>
</tr>
<tr>
<td>23</td>
<td>The verifier asks the server to verify the signature on the received message. The verifier makes a new request that contains the message from the TEE, the prover ID and the nonce and signs it with the verifier private key.</td>
</tr>
<tr>
<td>24</td>
<td>The server verifies that the prover ID matches with the TEE that signed the message, and sends a message containing the nonce, prover ID and message to the verifier, signing it with the server private key.</td>
</tr>
</tbody>
</table>

Table 5.5: Summary of developed RA protocols

<table>
<thead>
<tr>
<th>Name</th>
<th>Agents</th>
<th>Verdict created by</th>
<th>Secret Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-agent Simple</td>
<td>Prover Verifier</td>
<td>Verifier</td>
<td></td>
</tr>
<tr>
<td>Two-agent Sharing</td>
<td>Prover Verifier</td>
<td>Verifier</td>
<td>✔️</td>
</tr>
<tr>
<td>Three-agent Simple</td>
<td>Prover Verifier Server</td>
<td>Server</td>
<td></td>
</tr>
<tr>
<td>Three-agent Sharing</td>
<td>Prover Verifier Server</td>
<td>Server</td>
<td>✔️</td>
</tr>
</tbody>
</table>
5.3. Three-Agent Protocol

Figure 5.4: Three-agent RA protocol with secret sharing
This chapter presents the evaluation method for evaluating the protocols against the attestation goals and the evaluation results. Each protocol was modelled in Tamarin, and evaluated by proving or disproving a set of lemmas. Verification was performed using Tamarin version 1.6.1. Section 6.1 describes how the Tamarin models were built, what Tamarin models were constructed, and the lemmas that each model were evaluated against. Section 6.2 show the final results of the evaluation.

6.1 Evaluation Method

Five protocols are evaluated with respect to the attestation goals: the developed two-agent and three-agent protocol with and without secret sharing, as well as the Samsung KNOX Enhanced Attestation V3 protocol. Evaluation is performed by creating a Tamarin model for each protocol, specifying the lemmas based on the security properties presented in Section 4.5 and proving the lemmas using Tamarin.

6.1.1 Models

In Chapter 5 four protocols were presented, two of them with secret sharing, two of them without. The protocols were evaluated against both of the adversary models that were presented in Section 4.2. The protocols with secret sharing were only evaluated for adversary model A due to time constraints. The versions without secret sharing were evaluated for both adversary models, including the Knox protocol. This leads to a total of eight evaluations, requiring eight Tamarin models. The Tamarin models are listed in Table 6.2. An overview of what adversary model each Tamarin model uses is shown in Table 6.1.

6.1.2 Lemmas

Table 6.3 shows a list of lemmas used during verification. There are 15 lemmas in total. Each lemma is associated with a security property. The rows with goals in the rightmost column mark the security properties which were selected in Section 4.5. Lemmas within the same table segment prove stronger or weaker versions of the same security property. For some of the security properties we use multiple lemmas that verify weaker properties based on
Table 6.1: Tamarin models

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Protocol</th>
<th>Adversary Model</th>
<th>Secret Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>twoagent_A</td>
<td>Two-agent Simple</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>twoagent_B</td>
<td>Two-agent Simple</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>twoagent_secret</td>
<td>Two-agent Sharing</td>
<td>A</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>threeagent_A</td>
<td>Three-agent Simple</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>threeagent_B</td>
<td>Three-agent Simple</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>threeagent_secret</td>
<td>Three-agent Sharing</td>
<td>A</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>knox_A</td>
<td>Knox</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>knox_B</td>
<td>Knox</td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Tamarin model overview. Each check-mark represents a model

<table>
<thead>
<tr>
<th>Protocol Version</th>
<th>Two-agent</th>
<th>Three-agent</th>
<th>Knox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Sharing</td>
<td>Simple</td>
<td>Sharing</td>
</tr>
<tr>
<td>Adversary Model A</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Adversary Model B</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.3: List of all lemmas used during verification

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Security Property</th>
<th>Target</th>
<th>View</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>correctness_trace</td>
<td>Correctness Trace</td>
<td>-</td>
<td>-</td>
<td>G1</td>
</tr>
<tr>
<td>2</td>
<td>aliveness_vf</td>
<td>Aliveness</td>
<td>TEE</td>
<td>Verifier</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>n_i_aliveness_vf</td>
<td>One-way Non-Injective Agreement</td>
<td>TEE</td>
<td>Verifier</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>i_aliveness_vf</td>
<td>One-way Injective Agreement</td>
<td>TEE</td>
<td>Verifier G2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>r_i_aliveness_vf</td>
<td>One-way Recent Injective Agreement</td>
<td>TEE</td>
<td>Verifier G3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>n_i_aliveness_data</td>
<td>One-way Non-Injective Agreement</td>
<td>TEE</td>
<td>Verifier OR Server</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>n_i_aliveness_vf_srv</td>
<td>One-way Non-Injective Agreement</td>
<td>Server</td>
<td>Verifier</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>i_aliveness_vf_srv</td>
<td>One-way Injective Agreement</td>
<td>Server</td>
<td>Verifier G5</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>aliveness_prv</td>
<td>Aliveness</td>
<td>Verifier</td>
<td>Prover</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>w_agreement_prv</td>
<td>Weak Agreement</td>
<td>Verifier</td>
<td>Prover</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>n_i_agreement_prv</td>
<td>Non-Injective Agreement</td>
<td>Verifier</td>
<td>Prover</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>i_agreement_prv</td>
<td>Injective Agreement</td>
<td>Verifier</td>
<td>Prover G6</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>sharing_secrecy</td>
<td>Secrecy</td>
<td>shared</td>
<td>secret</td>
<td>Verifier G7</td>
</tr>
<tr>
<td>14</td>
<td>n_i_agreement_secret</td>
<td>Non-Injective Agreement</td>
<td>TEE</td>
<td>Verifier</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>i_agreement_secret</td>
<td>Injective Agreement</td>
<td>TEE</td>
<td>Verifier G8</td>
<td></td>
</tr>
</tbody>
</table>

the authentication hierarchy. By including weaker properties, we can get a more precise description of what property the protocol fails to provide. Lemmas 13, 14, and 15 are only applicable to models which contain secret sharing. The exact formulation of all lemmas is included in Appendix A.

6.1.3 Tamarin Modelling Procedure

The protocols were modelled in Tamarin by writing protocol steps as rules. Receiving a message and sending a response was for the most part modelled as a single rule by describing the accepted input of the received message in the premise of the rule, and then describing the resulting output of the sent response in the conclusion of the rule.

Additionally, each agent was modelled to track its own agent state by using a unique fact for each protocol step. Such a state fact is generated from the conclusion of a single rule, and
6.1. Evaluation Method

consumed by the fact of the next protocol step for each agent. This ensures that an agent must perform its steps in protocol order.

Messages sent over a channel use the built-in \textit{In} and \textit{Out} facts, unless a secure channel is used. A message sent using the \textit{Out} fact will be sent to an adversary with Dolev-Yao capabilities. The adversary can then construct any message to be consumed using the \textit{In} fact.

The following subsections clarify the assumptions made in order to model the protocols in Tamarin, including the key infrastructure and channel assumptions.

Key Infrastructure

All Tamarin models assume an asymmetric cryptographic key system, with some existing provisioning of public keys. We assume each prover device comes with an attestation keypair provisioned to the TEE, and that the public key is known to either the verifier for two-agent protocols or the server for three-agent protocols. In addition, we assume both the verifier and server have keypairs used for signing messages, and that their public keys are known to all agents.

The models allow for any key to get leaked to the adversary. When verifying the security properties, we generally require that the keys of the agents involved have not been leaked prior to performing the protocol. This is specified in the lemmas, where we require that any agent we believe to be honest upon completing the protocol cannot have had any of its keys leaked.

Network Channels

For connections between secure agents, we allow for secure connections using HTTPS/TLS. For HTTPS connections between a client and a remote server, the server is authenticated, but typically not the client. We model this by allowing anyone to send the first request. The channel for the first request is modelled as a confidential channel, but the sender is not authenticated. We also allow for the adversary to intercept requests and change the sender. Once the connection has been established, any messages sent over the connection are fully confidential and authenticated between the responder and the sender. The Tamarin model of the HTTPS channel is shown in Listing 6.1. We model all of the secure channels over the network this way. For insecure network channels, we instead use the built-in In and Out facts.

TrustZone Assumptions

We model the REE and TEE as separate agents, both with their own protocol state. The REE and TEE share a named variable that identifies the prover. Only the TEE is allowed to ever access the attestation key used for signing attestation data.

For the channels between the REE and TEE, we either model a completely insecure channel in adversary model A or a completely secure channel for adversary model B. The secure channel used for adversary model B is shown in Listing 6.2. The insecure channel used for adversary model A is shown in Listing 6.3.

Knox Modelling Decisions

We make some simplifications of the Attestation Agent when modelling the Knox protocol. Instead of modelling the REE, Attestation Agent, and TEE as separate agents, we merge the Attestation Agent with either the REE or the TEE depending on the adversary model. For adversary model A, where the adversary controls the attestation agent, we model it as if it is a part of the REE. The motivation being that the REE and attestation share the same capabilities in adversary model A, and are both insecure. Therefore, any insecure communication with the Attestation Agent could also be modelled as insecure communication between the REE
6.1. Evaluation Method

Listing 6.1: TLS channel modelled using confidential and secure channels in Tamarin

```plaintext
/*
The initial message is modelled as a confidential channel.
Any device can make a request over the channel, therefore
the adversary can also send requests using the channel.
*/

//Initial request over confidential channel
rule ChanOut_TLS_start:
  [ Out_TLS_1(A,B,x) ] //A sender, B receiver, x data
  -->
  [ !TLS_req(B,x), Out(A) ] // Adversary can see sender

//Initial request on receiver's end
rule ChanIn_TLS_start:
  [ !TLS_req(B,x), In(A) ] // Adversary can modify sender
  -->
  [ In_TLS_1(A,B,x) ]

//Initial request if initiated by adversary
rule ChanIn_TLS_adv:
  [ In(<A, B, x>) ] // Adversary can send own message
  -->
  [ In_TLS_1(A,B,x) ]

/*
Once a connection has been established, any subsequent messages are
assumed to be sent over a secure channel to the initiator.
*/

//Response message
rule ChanOut_TLS_reply:
  [ Out_TLS_2(A,B,x) ] // A sender, B reciever, x data
  -->
  [ TLS_data(A,B,x) ] // Message not modifiable or repeatable

//Response received by initiator
rule ChanIn_TLS_reply:
  [ TLS_data(A,B,x) ]
  -->
  [ In_TLS_2(A,B,x) ]
```

Listing 6.2: Secure TrustZone channel modelled in Tamarin

```plaintext
//Secure channel between prover components
rule ChanOut_TZ:
  [ Out_TZ(A,B,x) ] //Message sent from component
  -->
  [ Sec_TZ(A,B,x) ] //Message cannot be modified or repeated

rule ChanIn_TZ:
  [ Sec_TZ(A,B,x) ]
  -->
  [ In_TZ(A,B,x) ] //Message received to component
```
and the TEE. For adversary model B, where the prover is secure, we model the Attestation Agent as if it is a part of the TEE. The reasoning is the same: because communication with the Attestation Agent is secure we can model it as secure communication between the REE and TEE, and then merge the capabilities of the TEE and the Attestation Agent. This simplification also allows for a comparison of the Knox protocol with the developed two-agent and three-agent protocols since their models do not contain an Attestation Agent.

The Knox protocol leaves the implementation of the verifier and prover application to the Mobile Device Management service provider. This allows for different implementations. In order to create a model for the protocol, some assumptions about the implementation must be made. We assume the prover implementation authenticates all requests from the verifier. We assume the protocol is initiated by the verifier who requests a nonce from the server and forwards it to the prover, rather than being initiated by the prover. When forwarding the nonce to the prover, we assume such message contains the identity of the prover. This is used to indicate the intended attestation target. These assumptions are not necessarily described in the documentation of the protocol. However, making these assumptions is helpful in order to provide the desired security goals. We make these assumptions in order to prove properties for a strong implementation of the protocol, rather than a weak implementation.

We assume the server and verifier are both static publicly known endpoints, and will not change between runs. The reasoning being that the protocol is designed to be used by a Mobile Device Management service taking the role as the verifier. The implementation on the prover can thus assume it will always communicate with a given Mobile Device Management server. Similarly, the server will always be a given Samsung server endpoint, and so we do not need to consider multiple servers. We model this using public constants in Tamarin. The server and verifier are both named using public constants, meaning their names are known to all agents including the adversary, and they will never change.

For the developed protocols we do not assume static endpoints. This models a more general protocol, where any device can act as a verifier, prover or server.

The modelling decisions made are summarised in Table 6.4.
Table 6.4: Summary of tamarin modelling decisions

<table>
<thead>
<tr>
<th>Modelling Decision</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>All agents have provisioned asymmetric keys</td>
<td>All agents can produce or have provisioned keys</td>
</tr>
<tr>
<td>Keys can get leaked</td>
<td>Leaked keys are considered a special case in lemmas</td>
</tr>
<tr>
<td>Agents can communicate over secure channels in model B</td>
<td>HTTPS can provide secure channels</td>
</tr>
<tr>
<td>Adversary can send messages on the secure channel</td>
<td>Any device can initiate a HTTPS request</td>
</tr>
<tr>
<td>REE and TEE have separate states</td>
<td>Execution on the REE and the TEE is independent</td>
</tr>
<tr>
<td>Knox Attestation Agent is modelled as REE for adversary model A</td>
<td>Model A considers a strong adversary</td>
</tr>
<tr>
<td>Knox Attestation Agent is modelled as TEE for adversary model B</td>
<td>Model B considers a weak adversary</td>
</tr>
<tr>
<td>Prover authenticates requests from the verifier</td>
<td>Possible due to key infrastructure</td>
</tr>
<tr>
<td>Verifier initiates protocol</td>
<td>Agrees with protocol specifications</td>
</tr>
<tr>
<td>Prover identity is included in initial request</td>
<td>Specify intended attestation target</td>
</tr>
<tr>
<td>Verifier and server names are public constants in Knox Models</td>
<td>The endpoints are known to the attestation application beforehand</td>
</tr>
</tbody>
</table>

6.2. Results

Table 6.5 shows a summary of the lemmas proven for the models. Checkmarks indicate that the lemma could be successfully proved by Tamarin. Crosses indicate that Tamarin disproved the lemma for the model. Lemmas that are not applicable for the given model are marked with "."

For the two-agent and three-agent protocols, all applicable lemmas were proven for both adversary models, and both with and without secret sharing. This is to be expected since they were designed specifically to fulfil all security properties.

All lemmas except injective_agreement_pv were proven for the Knox protocol when considering adversary model B. This lemma was the strongest security property when considering Goal G6. When considering adversary model A, some lemmas concerning goals G2, G3 and G5 were disproved by Tamarin for the Knox protocol in addition to lemma injective_agreement_pv.

Tamarin additionally provides examples that show how the security properties were violated in the form of attack traces. Such examples, along with a discussion of their cause will be presented in Section 7.1 found in Chapter 7.
### 6.2. Results

Table 6.5: Verification results for protocols with adversary model A

<table>
<thead>
<tr>
<th>Goal</th>
<th>Lemma</th>
<th>Two-Agent</th>
<th>Three-Agent</th>
<th>Knox</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simple</td>
<td>Sharing</td>
<td>Simple</td>
</tr>
<tr>
<td>G1</td>
<td>correctness_trace</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G2</td>
<td>i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G3</td>
<td>r_i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G4</td>
<td>n_i_aliveness_data</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_aliveness_vf_srv</td>
<td>-</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>G5</td>
<td>i_aliveness_vf_srv</td>
<td>-</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>aliveness_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>w_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G6</td>
<td>i_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G7</td>
<td>sharing_secrecy</td>
<td>-</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>aliveness_secret</td>
<td>-</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>w_agreement_secret</td>
<td>-</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>n_i_agreement_secret</td>
<td>-</td>
<td>✔</td>
<td>-</td>
</tr>
<tr>
<td>G8</td>
<td>i_agreement_secret</td>
<td>-</td>
<td>✔</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.6: Verification results for protocols with adversary model B

<table>
<thead>
<tr>
<th>Goal</th>
<th>Lemma</th>
<th>Two-Agent</th>
<th>Three-Agent</th>
<th>Knox</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simple</td>
<td>Sharing</td>
<td>Simple</td>
</tr>
<tr>
<td>G1</td>
<td>correctness_trace</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G2</td>
<td>i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G3</td>
<td>r_i_aliveness_vf</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G4</td>
<td>n_i_aliveness_data</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_aliveness_vf_srv</td>
<td>-</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>G5</td>
<td>i_aliveness_vf_srv</td>
<td>-</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>aliveness_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>w_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>n_i_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>G6</td>
<td>i_agreement_pv</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

41
In this chapter, methods and results are discussed. First, results from the evaluation are discussed in Section 7.1. In Section 7.2, the method used is motivated and criticised with respect to alternative evaluation methods, modelling decisions and attestation goals. Finally, we discuss the work in a wider context by lifting some potential ethical concerns regarding remote attestation in Section 7.3.

7.1 Results

Because all lemmas were proven for all protocols when considering adversary model B, except for Goal G6 for the Knox protocol, all protocols can achieve most attestation goals when only considering a weak adversary. However, when considering the strong adversary in adversary model A, the Knox protocol fails to provide some additional attestation goals. The following subsections discuss why this may be the case, and how these issues were solved in the developed three-agent protocol.

7.1.1 G2 Authenticity of TEE

The model for Knox Enhanced Attestation protocol fails to provide aliveness seen from the verifier when considering adversary model A. This indicates that the Knox protocol does not fulfil attestation goal G2: Authenticity of TEE when considering a strong adversary. Because all authentication lemmas hold when considering the weaker adversary in adversary model A, we can determine that the failure to provide authentication depends on what adversary model we consider.

When a lemma is disproved by Tamarin a counter-example is produced. Such a counter-example consists of an attack trace: an execution of the protocol that will violate the lemma. The attack trace that violates aliveness for adversary model A reveals that a relay attack is possible. In this attack trace, one prover can attest itself through the use of a helper device. An example of such an attack is shown in Figure 7.1. Note that the intended prover prover1 never runs the protocol on the TEE (which for the Knox protocol translates to the TrustZone agent), which is why this attack violates aliveness. This execution is not possible for adversary model B, because the adversary cannot compromise any helper device to perform the attestation on.
7.1. Results

Figure 7.1: Relay attack on Enhanced Attestation V3 by strong adversary. The verifier intends to perform attestation with prover1, however the adversary with control over the REE of both prover1 and prover2 is able to forward the request to prover2. The verifier is not able to detect that the attestation was performed by a different prover than intended. The agents shown in red are controlled by the adversary.

There are two interpretations for the attack trace which we should consider. We describe them as scenario one and two.

**Scenario one:** An adversary with control over a rooted device in the role Prover1, tries to hide their presence by redirecting the attestation request to another device Prover2. Then it would seem to the verifier as if Prover1 had the configuration of Prover2. If Prover2 can be made to perform the request without being compromised in a way that is detectable by the attestation procedure, then Prover1 can have any configuration while still being considered benign by the verifier.

**Scenario two:** An adversary in the role of Prover2 is trying to disrupt the attestation procedure of another device by intercepting the attestation request intended for Prover1. In the case that Prover2 fails attestation, it would seem to the verifier as if Prover1 was compromised.

Both these scenarios provide a potential threat in case the prover application can be compromised. However, the analysis using Tamarin has been performed using simplified models of the Knox protocol, and in practice there are several mechanisms that protect against these attacks. We first note that the attestation procedure is able to detect whether a device is rooted. As such, we cannot root either device in order to perform the attack in the first scenario if the adversary wants to evade detection. In addition, the run-time protection of kernel integrity through Real-time Kernel Protection further protects the Normal World operation. This limits the attack surface further to user applications, since only user applications lack run-time protection.

Another protecting mechanism is that the verdict from the Attestation Server contains the application name and signature of the application used by the prover in order to perform the attestation. This prevents the very simplest attack, which would be implementing an attestation application on a helper device that bypasses all checks of verifier identity. Such
helper device could perform attestation on behalf of the adversary-controlled device. However, the attestation application would have a different signature, which would be noticeable by the verifier when receiving the verdict form the server. Instead, sophisticated run-time attacks would need to be performed on the otherwise benign helper device in order to force an attestation intended for a different device using the same application as intended.

From this we conclude that the practical attack surface consists of the user attestation application, and so ensuring security of the prover application would effectively prevent both of the attack scenarios described. Samsung have been made aware of both these scenarios. The attacker model that we have considered for this attack is stronger than what is usually considered for the KNOX platform. Because the prover application considered in the attestation scenario is usually an Enterprise Mobile Management application, and usually has more permissions compared to normal applications, it is generally assumed to be secure and trusted. Samsung does not consider the scenarios presented as threats to remote attestation, since they would require a stronger attack which is able to compromise trusted normal-world applications.

We further emphasise the importance of correct implementation of the prover attestation application, since the prover application needs to protect against simple relay attacks, where requests from the verifier are sent unmodified to a different device than intended. We also note that scenario two only requires that the nonce intended for Prover1 can be leaked, since this is what decides if Prover2 can interfere by performing their own attestation. Thus a confidential channel, for example using HTTPS, is needed in practice for the communication between the prover application and verifier.

Similar attacks on RA have been discussed in previous work. Parno [41] refers to the attack on RA described in the first scenario as a cuckoo attack. A similar attack is described by Asokan et al. [5] in 2003, showing how compositions of protocols may be vulnerable when using different authentication mechanisms for each. Salowey and Hanna [51] show how this attack also applies to a RA protocol. The Trusted Computing Group specifically mentions in the Trusted Attestation Protocol standard that "Additionally, it is up to the protocol specifications to provide a means to mitigate against an Asokan attack" ([27], p. 7), referring back to the paper by Salowey and Hanna.

Note that the cause is not lack of authentication on the server side, as shown by the proof of lemma $n_i \_aliveness \_data$. The server has no issue authenticating the prover, since the server receives the signature and certificate chain. However, because the response that the server sends to the verifier does not contain any identification of the prover, this information is never relayed to the verifier. We suggest that some device ID is included in the final response from the server in order to strengthen the protocol. If such a device ID is included, there is no issue for the protocol to authenticate the prover to the verifier, even in the case of a strong adversary. The developed three-agent protocol uses such a mechanism: the server replies with the identity of the prover in the final request.

### 7.1.2 G5 Integrity of Attestation Verdict

Because adversary model A allows for the server requests to get replayed, it is possible to repeat the verdict given by the server to the verifier in the Knox protocol. The adversary can listen to the channel, read the nonce and store it in order to send it to the same verifier once the verifier tries to perform another request. Because of this, the lemma $i \_aliveness \_vf \_srv$ fails for the Knox protocol when considering adversary model A. This does not mean the adversary can modify the verdict itself, as shown by the proof of lemma $n_i \_aliveness \_vf \_srv$.

We should also note that the assumption that the adversary can repeat requests that are sent between the server and the verifier is inaccurate if we allow for secure channels using TLS between the agents, since such connections ensures recentness of the messages sent. By using HTTPS when communicating with the server from the verifier, we can ensure that
nonces cannot be repeated, and thus the attack-trace found here would not be usable for an attack that repeats verdicts.

7.1.3 G6 Authenticity of Verifier

The motivation for including of the negotiation of a nonce between the prover and the verifier for all of the developed protocols was in order to provide injective agreement seen by the prover. By allowing the prover to also provide a unique value for each protocol run, the prover can ensure that the request from the verifier has not been repeated. The prover thus gets a guarantee that the verifier is actively performing the protocol.

The Knox protocol does not contain such a negotiation, and we can see that the Knox protocol fails injective agreement seen from the prover when considering both adversary models. However, it would be possible to incorporate a negotiation between the prover and the verifier as part of the protocol implementation made by the service provider in order to provide injective agreement if desired. Non-injective agreement is provided regardless, meaning the prover can at least reject requests from verifiers it does not trust.

Just like for Goal G5, using HTTPS when communicating with the verifier would also prevent requests from getting repeated by the adversary.

7.1.4 Preventing Repeated Requests

In both Section 7.1.2 and Section 7.1.3 we describe an issue with providing a unique value for each request, resulting in a loss of injective authentication. This can be attributed to the weak channel assumptions where we allow the adversary to repeat requests. We can therefore not trust that the nonce either agent receives is a fresh nonce.

An alternative method for preventing repeated requests to either agent is to force the agent to only accept unique nonces. By rejecting any nonce the agent has seen before, the agent can still guarantee that the request is new. This has the downside of requiring a log of all nonces seen. Depending on how frequently RA is applied, this may be a suitable alternative.

In conclusion, the attack-traces found for the Knox protocol for goals G5 and G6 are both preventable during implementation of the prover application and verifier.

7.2 Method

In this section the chosen method is discussed. The modelling choices made are discussed in Section 7.2.1. Some alternative attestation goals are discussed in Section 7.2.2. Alternate methods for formal verification are presented in Section 7.2.3. Finally, the method for collection sources is described and discussion on said sources is presented in Section 7.2.4.

7.2.1 Modelling Decisions

The validity of the verification hinges on the modelling decisions made. Modelling of the Knox attestation protocol had to rely on several assumptions of the protocol implementation, since we could not use a reference implementation in our analysis. The resulting model is best-effort, but may not be entirely accurate.

Similarly, no reference implementation was used during the modelling of any of the developed protocols. Therefore, there is a risk that a feasible implementation of the protocols is not possible due to incorrect assumptions of the system capabilities. For example, the adversary model of the device only considers three agents: the REE, Attestation Agent and the TEE. This very simplified model does not capture the subtle interactions between layers of the software stack.

Depending on the concrete TEE implementation, it may be more accurate to consider the TEE as entirely stateless. Unless there is dedicated secure persistent storage available for the
7.2. Method

TrustZone platform there is no guarantee for the TEE that data can be securely stored to persistent memory. The TEE writing to persistent storage is often implemented using encrypted blobs of data that are sent to the REE where it can be stored using OS functionality. The Android Keystore for example is implemented this way [26].

Formal verification is only meaningful when using appropriate adversary models and security properties. The adversary models considered may only be relevant for special cases of RA and may not be applicable to all RA protocols. For example, RA which cannot be used to detect privileged malware may not be able to function regardless of when such compromise has occurred. In addition, there is a lack of standardisation in terms of what security properties are appropriate to consider for RA protocols. A large difficulty in this thesis consisted of specifying reasonable requirements, formalising security properties and finally specifying Tamarin lemmas to verify. Subtle differences, such as the difference between the two-way or one-way authentication properties, could affect whether a property would be proved or disproved. A lot of effort was put into specifying lemmas that could reach a conclusion about the attestation goals.

7.2.2 Alternative Attestation Goals

The following subsection describes some attestation goals that are relevant but were deemed out of scope.

Confidentiality of Attestation Data Notably missing from the attestation goals is confidentiality of the attestation data itself. Because many proposed and existing protocols for RA rely on the prover revealing some information about the software running on the device, it may be considered a vulnerability if such information is disclosed to an adversary. We should then design RA protocols such that attestation is performed in a way that does not disclose the attestation data to an adversary. Unfortunately, confidentiality was put out of scope due to time constraints following issues with modelling and proving confidentiality.

Two approaches for proving confidentiality were considered: secrecy and observational equivalence. Proving secrecy in Tamarin limits us to using fresh variables that are unknown to the adversary. This models the attestation procedure poorly, since in reality the expected attestation contents are likely to be predictable when using a known integrity measurement. Then encrypting the attestation data with for example the verifier public key would not suffice, since the adversary could do the same to all possibilities and match the ciphertexts with the sent measurement.

It is reasonable to assume that an adversary with knowledge of the attestation procedure should be able to produce attestation data that is valid. What is more important is that the adversary should not be able to deduce exactly what attestation data was sent for a particular attestation, since there may be multiple different configurations, and thus different attestation data that could be considered valid. This property is better represented by observational equivalence. The idea is to model two different systems, where each system corresponds to a choice of attestation data. If the systems exhibit observational equivalence, then the adversary cannot learn what choice was made, and thus cannot learn what attestation data was sent.

However, attempts of proving observational equivalence were unsuccessful, and the property could not be proved or disproved for the models. Proving observational equivalence in Tamarin is challenging for a number of reasons. First, the proof requires making an additional model, where the choice between two attestation integrity measurements is made. This creates two separate systems, one system for each choice. Next, proving the lemma requires that Tamarin can match all executions of one system with the other system. This requires a considerably larger state space exploration than proving other lemmas, since they can usually be proven as state reachability. Proving observational equivalence required considerably more computational power. While this could have been solved by using stronger com-
putational resources, Tamarin also unfortunately encountered cases where it would never terminate. Due to time constraints, observational equivalence was excluded from the thesis.

**Comprehensive Information:** The attestation needs to prove to the verifier that the prover is trusted. This is a very difficult problem in practice. Most research on remote attestation is mainly concerned with finding appropriate measurements in order to prove that not only did the prover load the desired software, but said software is still running correctly and has not been compromised. This thesis instead refers to the literature on remote attestation described in Section 3.1.2 in order to motivate that such measurements are feasible to perform as part of the protocol.

**Anonymity of Attestation Device:** In some applications of RA it may be desirable for the prover to remain anonymous to the verifier in order to avoid tracking of the device. As demonstrated by existing work on Direct Anonymous Attestation (see Section 3.2.1) achieving anonymity is possible. In this thesis, Direct Anonymous Attestation was not investigated further: however, the possibility to perform anonymous attestation can be very valuable for RA implementations. Direct Anonymous Attestation could avoid many of the difficulties of providing both comprehensive information and providing constrained disclosure.

**Minimality of TEE:** In order to provide a trusted execution environment, it is desirable to keep the attack surface of the environment minimal. Since we rely on a trusted environment in order to provide functionality for remote attestation, there is a trade-off between provisioning most of the functionality for remote attestation to the TEE, but increasing the size trusted computing base as a result, or to provision parts of the protocol to the REE to keep the trusted computing base minimal, but possibly opening up for more ways for an adversary to attack the protocol.

### 7.2.3 Alternative Verification Tools

There are other tools that have been used for the analysis of cryptographic protocols. Sardar et al. [54] [55] used ProVerif [11] during the formal verification of RA on Intel SGX and Intel TDX. Lowe [37] used Failures Divergences Refinement Checker (FDR) when finding an attack on the Needham-Schroeder Public-Key Protocol, and when formally verifying the corrected protocol. Hess and Mödersheim [28] discuss the use of Isabelle/HOL to prove security properties for protocols. In the end, Tamarin was chosen for the possibility for interactive proofs and for the great tutorial, documentation and code examples.

### 7.2.4 Sources

Sources were found using Google Scholar by searching for keywords and then prioritising sources from highly reputable conferences or journals. The literature found also provided additional relevant sources through the references used. Some sources were also found by investigating what other sources had cited the same paper, through the use of Google Scholar. Most sources in background and related work have been selected from conferences and journals ranked highly by the CORE ranking.

Only a few sources are from non-peer-reviewed sources, such as online articles or online documentation. The article by Gayde [24] describes the reasons for why ARM processors have become the popular choice for mobile and IoT devices. This article is only meant to provide additional context and is not critical to the thesis. The references that follow in Section 7.3 are to be seen as less credible sources. They however provide an important discussion to the topic at hand.
Finally, there are some sources from the ARM, Samsung and Intel developer documentation. These are all believed to be credible sources, although they are expected to have some bias towards their own products.

### 7.3 The Work in a Wider Context

Remote Attestation, and Trusted Computing [40] in general has been criticised by some. Concerns have been raised regarding the anti-competitive nature of the dedicated hardware security extensions. Anderson [3] voiced some concerns in 2003 about the trusted computing features that were being developed at the time by Microsoft. Anderson made a comparison to the practice of using chips in ink cartridges in order to limit the usability of third-party ink cartridges in printers, and feared that similar practices would become prevalent also for forcing software to only run on certain platforms. There was concern that trusted computing would be used mostly as a means to provide Digital Rights Management for large companies in for example music or film industries.

An article by Schoen [55] from 2003 specifically discusses RA and how allowing third-party policy enforcement could restrict access to services for devices. The fear being that if a service provider can decide to only allow access to resources from devices that have certain software installed, anti-competitive behaviour becomes a lot more practical for large corporations.

Concerns have been raised that RA will enable identification of end-users on the internet [61]. Such concerns have mostly been lifted through the introduction of Direct Anonymous Attestation. However, platforms which still do not use such schemes for attestation keys remain linkable to individual hardware devices, and by extension to individual end-users.

While most criticism of trusted computing seems to have fizzled out over the years, RA has seen limited interest from end-users. More secure platforms have historically been seen as cumbersome for end-users, and have been viewed as restricting free use. As an example, the introduction of secure boot caused some trouble for Linux users [43]. This coupled with the large, opaque specification of Trusted Platform Modules, and the closed, proprietary implementations of Trusted Execution Environments makes it difficult for end-users to evaluate the added security. All the same large companies are pushing for the deployment of Trusted Computing resources. For example, computers running Windows 11 will be forced to activate the Trusted Platform Module [39].

In order for RA to be used for the benefit of end-users, there needs to be more emphasis put on providing user control over RA procedures. RA specifications should be open standards, and users should be in control over allowing or disallowing RA. Ideally, RA would be provided as a service to the user in order to verify the integrity of the device, rather than to be used as a mechanism for a third-party to force certain platform configurations before accepting requests. This was part of the motivation for also developing a two-agent protocol, since such scenario can be used when the prover and verifier are both in the hands of the end-user who wants to ensure the absence of malicious rootkits on a mobile device.
8 Conclusion

This thesis has performed formal verification of Remote Attestation protocols using the Tamarin prover. Requirements for Remote Attestation were gathered and translated into formally verifiable security properties. The security properties were used to analyse the Samsung Knox Enhanced Attestation V3 protocol, and two adversary models with different capabilities were considered. The protocol model was constructed based on our limited understanding of the protocol, using several assumptions about the protocol implementation. According to our best-effort modelling, the analysis indicates that the protocol can only fulfil the strong security properties for authentication when considering the weaker adversarial model.

In addition, two general-purpose remote attestation protocols were developed. Protocols were developed with the help of Tamarin in order to reason about the necessary requirements of the protocol. The developed protocols fulfilled all goals during formal verification. Finally, these protocols were extended to also include a data provisioning mechanism. These extended protocols were formally verified against some additional security properties.

This work further exemplifies the usability of automated formal methods for the purpose of security protocol analysis. The thesis also shows how formal verification can be used alongside protocol development in order to provide strong security guarantees.

8.1 Research Questions

Below we summarise the answers to the research questions that were formulated in Section 2.3.

What security properties are necessary to uphold in a protocol for remote attestation?

The security properties deemed necessary for Remote Attestation were presented in Section 4.5. These were formulated based on the requirements presented in Section 4.3 in the form of goals that are reasonable to expect from a Remote Attestation protocol. These include goals required by the verifier in order to make an appropriate judgement of the prover state. They also include goals required by the prover in order to provide responsible information disclosure.
8.2. Future Work

How can a protocol for remote attestation be designed for platforms with Trusted Execution Environments in order to guarantee the security properties?

The developed protocols were presented in Chapter 5. Both developed protocols fulfill all security properties even when considering a strong adversary with control over both network communications and some prover capabilities.

How can remote attestation protocols be formalised for the purpose of formal verification using Tamarin?

Formalising protocols in Tamarin involves writing rules to represent each message that is sent. The messages are sent either over the adversary-controlled channel, or over a secure channel, depending on the adversary model used and on the channel in question. In order to model remote attestation protocols in Tamarin, a number of assumptions had to be made. These assumptions are described in detail in Section 6.1.3. Importantly, we assume an existing key infrastructure consisting of asymmetric keys, and explicitly state how the secure channels are modelled.

How can remote attestation be extended in order to provide secret sharing to mobile devices?

We show how to extend Remote Attestation in order to provide secret sharing for both a Remote Attestation protocol with two agents and one with three agents. The protocols are shown in Section 5.2.1 and Section 5.3.1. For the developed protocols, we verify additional security properties to prove secrecy and integrity of the shared secret.

8.2 Future Work

Confidentiality of attestation data is important in order to provide security to end-users. Formally proving observational equivalence of attestation data for RA would therefore further prove the overall security of RA protocols. Future work should investigate methods for proving observational equivalence for RA protocols.

The protocols developed were never tested through concrete implementations. Evaluating an implementation on a TrustZone platform would increase confidence in their applicability in real world scenarios.

Secret sharing was only briefly discussed in this thesis, and more work is required to provide a deeper analysis of its benefits and challenges.
Bibliography


[27] Trusted Computing Group. TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0. Ver. 1.0, Rev. 0.36. Last accessed 4 May 2022. 2019. URL: https://trustedcomputinggroup.org/wp-content/uploads/TNC_TAP_Information_Model_v1.00_r0.36-FINAL.pdf


[39] Microsoft. Enable TPM 2.0 on your PC. 2021. URL: https://support.microsoft.com/en-us/windows/enable-tpm-2-0-on-your-pc-1fd5a332-360d-4f46-ae1e7-ae6b0c90645c


This appendix lists the Tamarin lemmas used during the verification of the security properties.

Listing 9.1: Lemma 1: Correctness of Protocol. All Tamarin models have a different number of steps, and so each lemma is different for each model. This example shows correctness trace lemma for Tamarin model twoagent_A.

```plaintext
// Correctness Trace for model twoagent_A
lemma correctness_trace:
  exists-trace
  // There exists a trace where...
  *Ex VF prover nonce1 nonce2 vf_public_key SML id_VF id_REE id_TEE
    #a #b #c #d #e #f #g #n #o #p.
    // ...Steps 1-7 can occur...
    Step1(VF, prover, nonce1, vf_public_key) @a &
    Step2(VF, prover, nonce1, nonce2, vf_public_key) @b & (a < b) &
    Step3(VF, prover, nonce1, nonce2, vf_public_key) @c & (b < c) &
    Step4(VF, prover, <nonce1, nonce2>, vf_public_key) @d & (c < d) &
    Step5(prover, <nonce1, nonce2>, SML) @e & (d < e) &
    Step6(VF, prover, <nonce1, nonce2>, SML) @f & (e < f) &
    Step7(VF, prover, <nonce1, nonce2>, SML) @g & (f < g) &
    // ...And each agent was instantiated...
    Create(id_VF, VF) @n &
    Create(id_REE, prover) @o &
    Create(id_TEE, prover) @p
    // ...And no other run was performed by any other agent
    & not(Ex id agent #s. Create(id, agent) @s & not(#s = #n)
      & not(#s = #o) & not(#s = #p))
    // ...And no keys were leaked to the adversary
    & not(Ex A #r. PrivateKeyRevealed(A) @r)
```
// Aliveness
lemma w_aliveness_verifier:
  // For all possible executions
  "All VF prover nonce #i.
  // If the verifier completes its part of the protocol
  Commit(VF, prover, nonce) @i
  // Then the prover must have performed the protocol
  ==> ( Ex nonce2 #j. AttestStarted(prover, nonce2) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Non-Injective Aliveness
lemma n_i_aliveness_verifier:
  // For all possible executions
  "All VF prover nonce #i.
  // If the verifier completes its part of the protocol
  Commit(VF, prover, nonce) @i
  // Then the prover must have performed the protocol
  using the same nonce as the verifier
  ==> ( Ex #j. AttestStarted(prover, nonce) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Injective Aliveness
lemma i_aliveness_verifier:
  // For all possible executions
  "All VF prover nonce #i.
  // If the verifier completes its part of the protocol
  Commit(VF, prover, nonce) @i
  // Then the prover must have performed the protocol
  using the same nonce as the verifier
  ==> ( Ex #j. AttestStarted(prover, nonce) @j )
  // And verifier finished after the prover
  & j < i
  // And there was no other run by any verifier
  // using the same nonce
  & not (Ex VF2 prover2 #i2. Commit(VF2, prover2, nonce) @i2
  & not (#i2 = #i))
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
Listing 9.3: Lemma 5: Freshness of Attestation Data

```plaintext
// Recent Injective Aliveness
lemma r_i_aliveness_verifier:
  // For all possible executions
  "All VF prover nonce #i.
  // If the verifier completes its part of the protocol
  Commit(VF, prover, nonce) @i
  // Then the prover must have performed the protocol
  using the same nonce as the verifier
  ==> ( Ex #j #k. AttestStarted(prover, nonce) @j
  // and the verifier started before the prover ran
  & NonceConfirmed(VF, prover, nonce) @k
  & k < j
  // And verifier finished after the prover
  & j < i
  // And there was no other run by any verifier
  & using the same nonce
  & not(Ex VF2 prover2 #i2. Commit(VF2, prover2, nonce) @i2
  & not (#i2 = #i))
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```

Listing 9.4: Lemma 6 for two-agent: Integrity of Attestation Data. Lemma for two-agent is similar to the version used by the three-agent model: the verifier is replaced by the server for the three-agent model.

```plaintext
// Non-Injective Aliveness
lemma n_i_aliveness_verifier_data:
  // For all possible executions
  "All VF prover nonce SML #i.
  // If the verifier completes its part of the protocol
  Commit_SML(VF, prover, nonce, SML) @i
  // Then the TEE created the measurements
  ==> ( Ex #j. QuoteCreated(prover, nonce, SML) @j)
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```

Listing 9.5: Lemma 6 for three-agent: Integrity of Attestation Data

```plaintext
// Non-Injective Aliveness
lemma n_i_aliveness_data:
  // For all possible executions
  "All AS prover nonce SML #i.
  // If the server completes its part of the protocol
  Commit_SML(AS, prover, nonce, SML) @i
  // Then the TEE created the measurements
  ==> ( Ex #j. QuoteCreated(prover, nonce, SML) @j)
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```
Listing 9.6: Lemma 7–8: Integrity of Attestation Verdict

```plaintext
// Non-Injective Aliveness
lemma n_i_aliveness_vf_srv:
  // For all possible executions
  "All VF AS prover nonce verdict #i.
  // If the verifier receives a verdict
  VerdictRecieved(VF, AS, prover, nonce, verdict) @i
  // Then the server sent the same verdict for the same nonce
  ==> ( Ex AS #j. VerdictCreated(AS, prover, nonce, verdict) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Injective Aliveness
lemma i_aliveness_vf_srv:
  // For all possible executions
  "All VF AS prover nonce verdict #i.
  // If the verifier receives a verdict
  VerdictRecieved(VF, AS, prover, nonce, verdict) @i
  // Then the server sent the same verdict for the same nonce
  ==> ( Ex AS #j. VerdictCreated(AS, prover, nonce, verdict) @j
  // And the verdict was sent before it was received
  & j < i
  // And there was no other run by any verifier
  // using the same nonce that received the same verdict
  & not (Ex VF2 AS2 #i2.
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```
Listing 9.7: Lemmas 9–12: Authenticity of Verifier

```plaintext
// Aliveness
lemma aliveness_prv:
  // For all possible executions
  "All prover VF nonce #i.
  // If the protocol finishes for the REE
  Running(prover, VF, nonce) @i
  // Then the verifier sent a nonce to some prover
  ==> ( Ex prover2 nonce2 #j. NonceConfirmed(VF, prover2, nonce2) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Weak Agreement
lemma w_agreement_prv:
  // For all possible executions
  "All prover VF nonce #i.
  // If the protocol finishes for the REE
  Running(prover, VF, nonce) @i
  // Then the verifier sent a nonce to the REE
  ==> ( Ex prover nonce2 #j. NonceConfirmed(VF, prover, nonce2) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Non-Injective Agreement
lemma n_i_agreement_prv:
  // For all possible executions
  "All prover VF nonce #i.
  // If the protocol finishes for the REE
  Running(prover, VF, nonce) @i
  // Then the verifier sent the same nonce to the REE
  ==> ( Ex #j. NonceConfirmed(VF, prover, nonce) @j )
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Injective Agreement
lemma i_agreement_prv:
  // For all possible executions
  "All prover VF nonce #i.
  // If the protocol finishes for the REE
  Running(prover, VF, nonce) @i
  // Then the verifier sent the same nonce to the REE
  ==> ( Ex #j. NonceConfirmed(VF, prover, nonce) @j
  // And the nonce was sent before the prover ran
  & j < i
  // And there was no other execution by some REE
  // using the same nonce
  & not (Ex prover2 VF2 #i2. Running(prover2, VF2, nonce) @i2
  & not (#i2 = #i)))
  // Or some key of a participating agent was leaked
  | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```
Listing 9.8: Lemma 13: Confidentiality of Shared Secret

```plaintext
// Secrecy
lemma sharing_secrecy:
  // For all executions and all sharing secrets x
  "All x #i. Secret(x) @i
   // It is not possible for the adversary to learn x
   ==> (not (Ex #j. K(x)@j))
   // Or some key of a participating agent was leaked
   | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```
Listing 9.9: Lemmas 14–15: Integrity of Shared Secret

```plaintext
// Aliveness
lemma aliveness_secret:
    "For all possible executions
    "All prover VF x tag vf_public_key #i.
    // If the verifier believes the secret x was provisioned
    SecretStoredVF(VF, prover, x, tag, vf_public_key) @i
    // Then some TEE provisioned some secret
    ==> ( Ex x2 tag2 vf_public_key2 #j. 
        SecretStoredTEE(prover, vf_public_key2, x2, tag2) @j )
    // Or some key of a participating agent was leaked
    | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Weak Agreement
lemma w_agreement_secret:
    "For all possible executions
    "All prover VF x tag vf_public_key #i.
    // If the verifier believes the secret x was provisioned
    SecretStoredVF(VF, prover, x, tag, vf_public_key) @i
    // Then the same TEE provisioned some secret
    ==> ( Ex x2 tag2 #j. SecretStoredTEE(prover, vf_public_key, x2, tag2) @j )
    // Or some key of a participating agent was leaked
    | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Non-Injective Agreement
lemma n_i_agreement_secret:
    "For all possible executions
    "All prover VF x tag vf_public_key #i.
    // If the verifier believes the secret x was provisioned
    SecretStoredVF(VF, prover, x, tag, vf_public_key) @i
    // Then the same TEE provisioned x
    ==> ( Ex #j. SecretStoredTEE(prover, vf_public_key, x, tag) @j )
    // Or some key of a participating agent was leaked
    | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"

// Injective Agreement
lemma i_agreement_secret:
    "For all possible executions
    "All prover VF x tag vf_public_key #i.
    // If the verifier believes the secret x was provisioned
    SecretStoredVF(VF, prover, x, tag, vf_public_key) @i
    // Then the same TEE provisioned x
    ==> ( Ex #j. SecretStoredTEE(prover, vf_public_key, x, tag) @j
    // And it was provisioned before the verifier was done
    & j < i
    // And there was no other TEE
    // that provisioned the same secret
    & not (Ex prover2 vf_public_key2 tag2 #i2.
        SecretStoredTEE(prover2, vf_public_key2, x, tag2) @i2
        & not (#i2 = #j))"
    // Or some key of a participating agent was leaked
    | (Ex A #r. PrivateKeyRevealed(A) @r & Honest(A) @i)"
```