DEGREE PROJECT IN COMPUTER ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2020

Formal security analysis of authentication in an asynchronous communication model

JACOB WAHLGREN

SAM YOUSEFZADEGAN HEDIN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Formal security analysis of authentication in an asynchronous communication model

JACOB WAHLGREN
SAM YOUSEFZADEGAN HEDIN

Degree Programme in Computer Engineering
Date: 2020-07-06
Supervisor: Mohammad Khodaei
Examiner: Panos Papadimitratos
School of Electrical Engineering and Computer Science
Swedish title: Formell säkerhetsanalys av autentisering i en asynkron kommunikationsmodell

Abstract

Formal analysis of security protocols is becoming increasingly relevant. In formal analysis, a model is created of a protocol or system, and propositions about the security of the model are written. A program is then used to verify that the propositions hold, or find examples of where they do not. This report uses formal methods to analyse the authentication aspect of a protocol that allows private individuals, enterprises, and systems to securely and asynchronously share sensitive data. Unpublished, early drafts of the protocol were studied and algorithms described in it were verified with the help of the formal verification tool Tamarin Prover. The analysis revealed two replay attacks. Improvements to the protocol were suggested based on this analysis. In later versions of the protocol, the improvements have been implemented by the protocol developers.

Keywords: formal verification, Tamarin Prover, formal analysis, information security, authentication

Sammanfattning

Det blir alltmer relevant med formell analys av säkerhetsprotokoll. I formell analys så skapas en modell av ett protokoll eller ett system, och påståenden om modellens säkerhet skrivs. Ett program används sedan för att verifiera att påståendena gäller, eller för att hitta exempel där de inte gäller.

Den här rapporten använder formella metoder för att analysera autentiseringsaspekten av ett protokoll som tillåter privatpersoner, företag och system att asynkront dela känslig information på ett säkert sätt.

Opublicerade och tidiga utkast av protokollet studerades och de algoritmer som beskrivs i protokollet verifierades med hjälp av Tamarin Prover. Analysen avslöjade två återspelningsattacker. Förbättringar till protokollet föreslogs baserat på denna analys. I senare versioner har protokollutvecklarna implementerat förslagen.

Nyckelord: formell verifiering, Tamarin Prover, formell analys, informationssäkerhet, autentisering

Contents

Chapter 1

Introduction

Secure Data Sharing (SDS) [1] is a communication protocol that enables private persons, enterprises and systems to securely share sensitive data in an asynchronous and distributed manner. It is being developed by a Stockholm based startup company. Targets users include both private individuals and enterprises. The security of such a protocol is vital, and the goal of the protocol is to guarantee confidentiality, integrity, authenticity and authorization. This report aims to verify that the protocol ensures authenticity of messages. That is, that it is not possible to masquerade as another user within the system. The limitation to focus on authenticity is due to time constraints.

It is becoming more and more relevant to use formal verification methods to verify the proposed security properties of a protocol. This is because as systems grow more powerful, communication protocols are becoming increasingly complex. The more complex a system gets, the harder it is to reason about its security and reliability. During conventional development one may formulate a set of requirements for the system, though it is seldom proven that these requirements are in fact fulfilled. If a system is to be truly reliable its key requirements should be verified using formal methods.

A formally analysed system may also be a competitive advantage for the company. It may give confidence to potential customers and partners who request security. In addition, a potential exploit in the company's systems may cause a consumer backlash or the termination of partnership agreements. Formal analysis can avoid this by identifying potential security issues before they are ever deployed.

Our work is focused on authenticity, which is a key aspect of the SDS protocol and one of the major information security aspects in general. A message is authentic if it is authored by the apparent sender, and authentication is the

process of ensuring authenticity. Authentication is critical to security because an attacker could otherwise pretend to be another user, and for instance gain unauthorized access to data.

In this report we use formal analysis to examine the protocol's security guarantees. To perform the analysis we first identify key concerns described in the protocol. The system and its algorithms are then modelled using the model checking tool Tamarin Prover, which is specifically built with security protocols in mind. The identified key concerns are translated into lemmas—propositions about the algorithm and system. Tamarin Prover is then used to verify that the propositions hold, or provide counter examples.

Chapter 2 begins by exploring how formal analysis can be used to verify the correctness of a computer system, and its limitations. The chapter also contains a motivation of why Tamarin Prover was used for the analysis of the studied protocol. Lastly, there is an introduction to information security and relevant terminology. The SDS protocol is introduced in chapter 3. There is a high level introduction to how it works and its motivation. The security requirements of the protocol are also explained. Methodology is explained in chapter 4. Assumptions made during the modelling of the protocol are listed and explained. The result of our work is explained in chapter 5. Each modelled part of the protocol and associated lemmas are described. Potential improvements to the protocol are suggested based on the analysis of the lemmas. The report concludes with chapter 6, which contains reflections on model checking and suggestions for future work.

1.1 Research question

Does the authentication in the Secure Data Sharing protocol satisfy its intended security goals? We analyse authentication in communications between a device and its home server. Other security requirements, such as availability and authorization are not analysed.

Chapter 2

Background

This chapter provides a review of relevant information for understanding the rest of this document. Concepts and terminology introduced here will be used later on.

2.1 Verification

In order to verify a computer system, three different parts are needed.

- A framework for modelling the system.
- A specification language to describe the properties to be verified.
- A verification method that can confirm whether or not the modeled system satisfies the proposed properties.

Approaches to verification can be proof-based or model-based. This report deals with model-based verification techniques. A model-based approach can be described as establishing whether a model satisfies a specification. Model-checking is based on temporal logic. A fundamental idea in temporal logic, and therefore model checking, is that certain properties can be true or false at different points in time, depending on the state of the model [2].

Formal analysis relies on the analyser coming up with appropriate lemmas. This report makes no claims to prove the existence or non-existence of all potential security vulnerabilities in the security protocol, and instead tries to identify critical sections and propositions to verify. Assumptions are made about a system and how it works; thus changed or incorrect assumptions invalidate the model and the lemmas written for it.

The following subsections provide a brief description of some notable verification tools.

TLA+

TLA+ was created for concurrent and distributed systems [3]. Such systems are difficult to verify with the use of runtime tests as differences in timings and hardware make them non-deterministic. A concurrent system can show different characteristics and yield a different result from time to time, even if the input to the system is the same. TLA+ is based on temporal logic, and the execution of a system is represented as a sequence of discrete changes in state [3, 4].

The hosting provider Amazon Web Services use TLA+ to aid in the design of their systems, described in [5]. The use of formal methods has helped AWS find logic bugs, summarized in a table in the paper. However, they have not managed to find design bugs causing availability problems, for instance sustained retry requests when a server is too slow.

ProVerif

ProVerif is a model checking tool created to formally verify cryptographic protocols. It uses a Dolev-Yao adversary [6], a state machine with control of all messages sent over the network [7]. Such a network is represented by state machines that exchange messages. Messages are elements of an algebraic system. ProVerif is taught at the Parisian Master of Research in Computer Science [8]. ProScript [9] is a domain specific programming language for writing cryptographic protocol code that can automatically be translated into a ProVerif model.

ProVerif has been used to analyse the security of the 5G authentication protocol [10], the Signal Protocol used in the popular messaging app [9] and the SPPEAR architecture for participatory sensing systems [11].

Tamarin Prover

Tamarin Prover is a formal verification tool for security protocols developed by researchers at ETH Zürich. It is heavily inspired by ProVerif, and is similarly used for analysis of cryptographic protocols. It also models a network of the Dolev-Yao model [12]. The creators of Tamarin emphasize its ability to easily deal with security properties based on the state of the protocol, in contrast to ProVerif [13].

Tamarin Prover has, like ProVerif, been used to analyse several security protocols. For instance, the 5G authentication protocol [14] and TLS 1.3 [15]. The report on analysis of TLS 1.3 cite Tamarin's support of the Diffie-Hellman protocol, its inherent support of non-monotonic state, its visualizations and graphical user interface as reasons for choosing it over similar tools [15].

Since Tamarin is specifically designed to work with security protocols and has clear visualizations, it was chosen over similar tools for our analysis of the SDS protocol.

Alloy

Alloy is, in contrast to for example TLA+ or Tamarin Prover, a modelling language based on relational logic. It is primarily static (as in, there is no notion of time), and shares similarities with UML and Object Oriented Programming in how it models the world [see 16, File System Lesson I]. Alloy specifications can be verified by Alloy Analyzer, a tool that translates these specifications into a boolean expression. The expression can then be analyzed by a SAT solver in an attempt to find instances that satisfy the expression. Alloy was developed as a general-purpose specification language for software engineering and data modelling. It has been used in many fields, including scheduling, cryptography, and instant messaging [17].

2.2 Introduction to Tamarin Prover

In Tamarin Prover, the system is modeled as a set of states. At any given point, the state of the system can be described as a multiset of facts. In order to transition between states, a rewrite system, consisting of *rules*, is defined. A rule consists of three sequences of facts - a sequence describing what facts must hold in order to invoke the rule, a sequence describing the labels of the transition—*action facts*—and one sequence describing which facts hold after the invocation of the rule. Any of these sequences may be empty [12].

Once a model has been set up, lemmas are written. Lemmas define the security properties to verify and evaluate traces of action facts of an execution [12]. The set of traces can be limited with the use of *restrictions*.

Tamarin can in most cases automatically compute whether a given model satisfies its specification. In other cases, it may be necessary to manually guide the computation. This is because the automatic prover is not guarateed to terminate for all lemmas. For a complete explained Tamarin example, refer to appendix A.

2.3 Information security principles

This section provides a brief review of the basic principles of information security. The terminology described here is used in our specification of the security goals of the SDS protocol.

First, we introduce the CIA triad: confidentiality, integrity and availability. These form the most fundamental principles. We then continue to explain additional concepts such as authenticity and authorization.

Confidentiality

Confidentiality refers to the ability to protect data from being read by unauthorized users [18]. Confidentiality can, for example, be lost if a device is stolen, giving an adversary access to the system. Sending sensitive data unencrypted might also lead to unauthorized users gaining access to it.

A breach of confidentiality is called an interception attack. Interception attacks can be difficult to detect as an end user. From the legitimate user's point of view, everything may seem to be in order [18].

Integrity

Integrity is about the consistency and soundness of the sensitive data [18]. Unauthorized changes, deletions or additions to data should not be possible. The integrity of data can be compromised by an adversary, but it can also be compromised by authorized users making incorrect changes to the data, a corrupt file system, or hardware failure.

Modification attacks tamper with data and are usually considered attacks on integrity [18]. If the data being tampered with is an important configuration file, it may lead to loss of availability or confidentiality.

Availability

Availability refers to the ability to access data when it's needed [18]. Hardware failure or denial of service attacks by an adversary are common situations in which availability is harmed.

Interruption attacks are usually attacks on availability. A denial of service attack is primarily a way to temporarily prevent legitimate users from accessing sensitive resources. However, an interruption attack can potentially lead to data loss or corruption, in which case it can be classified as an attack on integrity [18].

In a fabrication attack, garbage data is generated to either affect the availability of a system through overloading, or the integrity of data, for example by filling a database with fabricated information [18].

Authenticity

The principle of authenticity ensures that a message can be properly attributed to its author. This can be enforced through the use of digital signatures [18]. The process of confirming the authenticity of a message is called authentication.

Lowe's hierarchy of authentication formalizes the concept of authenticity into four levels [19]. These can be used as a framework for analysis of authentication protocols. The following list summarizes the four levels.

1. **Aliveness:** The apparent author has at some point sent a message. This level is achieved by signing a constant value.

2. **Weak agreement:** The apparent author has at some point sent a message to the recipient. This level is achieved by signing the identity of the recipient.

3. **Non-injective agreement:** The apparent author has at some point sent a message to the recipient with the apparent content. This level is achieved by signing the identity of the recipient concatenated with the contents of the message.

4. **Injective agreement:** The apparent author has at *exactly one* point sent a message to the recipient with the apparent content. This level is achieved by including a unique value in the message and signing the identity of the recipient concatenated with the contents of the message.

Any of these levels can be strengthened by additionally ensuring *recentness*. This means that the recipient can verify that the signature was made recently [19]. It is achieved by including a time stamp in the signature. Recent aliveness is for instance a stronger version of the first level of authentication.

A major attack against authenticity is the replay attack. This is when an authentic message is resent out of its original context by an attacker [20]. For instance, if a protocol only ensures weak agreement, an attacker may masquerade as another user by replaying their message, while modifying its content.

Authorization

Authorization is the process of controlling users' access to resources. It differs from authentication in that authentication is about verifying identity, while authorization is about enforcing a policy [21].

Proper authorization relies on correct authentication and integrity of messages. As such, an attack on those aspects may be leveraged to bypass authorization.

2.4 Digital signatures

Digital signatures can be used to ensure integrity, authenticity and non-repudiation. The following section is a primer on how digital signatures work, based on [22].

Digital signatures rely on asymmetric cryptography where each party has an associated private and public key. A message encrypted with the private key can be decrypted using the public key, and vice versa. To sign a message, the sender generates a hash of the message, and then use their private key to encrypt the hash. This result is called the signature. Any recipient can then verify the signature of the message by decrypting the signature with the sender's public key and then compare it with the hash value of the received message.

Chapter 3

Protocol specification

This chapter provides a description of the SDS protocol. The latest version is specified in a publicly available document [1]. During this degree project work, the protocol was still in development. Where relevant, we describe parts of the protocol both at the time of our analysis, and at the time of publishing this report.

The purpose of the protocol is to provide a way for users, or groups of users, to share data while maintaining confidentiality and integrity. Users have fine-grained access control to their data, allowing it to be shared in a secure way. A copy of the data is stored on a server in order for clients to be able to participate asynchronously. In the most secure mode, direct access to a server does not enable viewing or editing the data on it. In other words, only the client devices need to be trusted in order for the data to remain secure.

The protocol can function in two separate modes, end-to-end or basic. Only stateful devices (such as a smartphone application) can use the end-to-end mode. In the mode, data is sent over secure channels and stored encrypted on the server. Therefore, the mode provides the full security guarantees of the protocol. The basic mode supports both stateful and stateless devices (such as a web browser). Data is still sent through secure channels, but it is stored unencrypted on the server. Shortly before publishing this report, basic mode was removed from the protocol.

In parallel to SDS, the startup company is developing an application for exchanging contact information and other data, which is built upon the SDS protocol. This product is a smartphone application where participants verify each other's identity by physical connection. This is implemented by one of the parties scanning a QR code on the other's screen. A nonce included in the QR code is used to initiate a key exchange. The end-to-end mode of SDS then

provides secure transmission, and storage of the data on the company's servers for synchronization purposes.

3.1 System model

The main entities participating in the system are called devices and home servers. End-users use a device to view and edit data in the system. Each user may control multiple devices. A device may also be referred to as a client. Devices may for instance be smartphones or web browsers.

Home servers provide storage and synchronization in the system. A home server may be controlled by an end-user, by the company, or by a third party. In the end-to-end mode, the controlling entity cannot view or edit data stored on the server.

Devices communicate with home servers through a bidirectional channel secured using TLS. The home server has a certificate. Home servers may additionally communicate with each other using TLS as well. TLS stands for Transport Layer Security and is a communication protocol providing confidentiality and integrity of data transfer [23].

3.2 Data model

This section provides a simplified high level overview of the protocol's data model. A complete description is provided by the specification [1]. Relevant details are presented together with the analysis in chapter 5.

In SDS, data is organized into objects. Objects contain an arbitrary number of data blobs called fields. Each object is represented by a stream of events. The messages sent over the network in the protocol contain lists of events. This model allows participants to exchange only parts of objects, and only the new events need to be transmitted. There are several types of events for describing different types of changes to an object.

- Owner event: Change the object's owner. The owner has full access rights to the object.
- Patch event: Update the data in a field of the object.
- Delete event: Remove the object.

- Access event: Grant users additional rights to the object. This can be on the level of read, write or admin and either for one single field or for all fields.
- Reset event: Remove access rights to a field.

The access rights to an object is defined by the owner, access and reset events. Together these allow the users to specify an ACL (access control list).

As an example, an object may represent a user's profile. Each field would contain an element of the profile, for instance a picture or their home address. To share their home address with another user, the profile owner would add an access event granting them read rights to the field containing the home address.

3.3 Security requirements

The trusted computing base of the system is the user's device. The system must maintain confidentiality, integrity, authenticity and authorization. These principles must hold even in the case of an untrusted home server.

There is no requirement of recent authenticity as clients may participate asynchronously and thus receive messages long after they were sent. Anonymity and pseudonymity are not goals for the system.

If the system is not secure, the confidentiality and integrity of sensitive user data could be compromised.

Chapter 4

Methods

In the first part of the study, an overview of relevant literature was conducted. We investigated popular tools for formal analysis, and the theory of information security. The results of the literature study formed the basis for chapter 2, and informed the continued work and the design of the analysis methodology. The principles presented in section 2.3 were used to formulate the security requirements of the protocol.

In the second part of the study, we read unpublished draft versions of the SDS protocol. They contained detailed descriptions of algorithms and data structures used in the system. For example, an algorithm can describe how user authentication is carried out.

The system was then modeled in Tamarin Prover with the use of rules. Sanity checks—lemmas that confirm that the expected behavior is possible within a model—were written. These checks gave confidence that the model was correct, and helped give feedback when something was incorrect.

We identified authenticity requirements of protocol messages from their descriptions in the specification. These properties were expressed in Tamarin Prover using lemmas. Tamarin was then used to analyse the model to prove that the lemmas hold, or find counter examples.

In cases where the authentication was insufficient, we discussed potential changes to the protocol to provide stronger authenticity. Our suggestions were then communicated to the company.

4.1 Modelling assumptions

This section presents some general choices and assumptions made during the modelling process.

- The specification allows for both email and phone numbers to be used. We choose to use email terminology in the model, since they are effectively interchangeable at this level of abstraction. Email addresses were assumed to be public, since we cannot assume an adversary would not be able to find or guess this from other channels.
- The model of the adversary is a Dolev-Yao adversary with complete control over the communication channel between a device and its home server. The adversary can read messages, inject new messages, and modify or delete messages. That is, when analysing the end-to-end mode we assume that the HTTPS channel between client and server is insecure. For instance the server itself may be controlled by an untrusted party.
- We assume clients ignore malformed messages. This includes among other things invalid signatures and unauthorized access changes.
- In the protocol, there may be multiple devices per user participating concurrently. We limit the scope of our model by only considering the case where a user has a single device.

Chapter 5

Results

Version 0.03 and parts of version 0.04 of the SDS protocol were analysed. A model was created with Tamarin Prover and lemmas were written to specify and verify the security properties of the modeled protocol[1].

In our Tamarin model, persistent facts allow users to access each other's public keys. Built-in functionality is used to generate signatures and validate them. Their correctness during the execution of an algorithm is enforced with the use of restrictions. Restrictions are also used to keep the set of possible executions relevant. For example, a restriction is used to ensure that Tamarin Prover does not consider scenarios where an user fails to authorize, yet is given access to the system. In the protocol, a user who is known to be unauthorized would not be granted access to the system, and therefore it is not relevant for the analyzer to consider such scenarios.

The analysis identified two replay attacks and verified that the basic mode fulfills its security requirements.

After receiving preliminary drafts of this thesis, the company modified the protocol to implement our proposed improvements. In this chapter we describe our analysis and any resulting changes to the protocol. Each part and its model are described briefly. For a detailed description of the current version of the protocol refer to the specification [1].

5.1 Owner event

A data object, or just object, consists of fields with data. An object can be shared between users, and different users can have different access levels to

[1]The complete code is available at `https://github.com/samhedin/formal_analysis_of_communication_model`

the object. Each object has one single owner, and ownership of an object can be transferred with the use of an owner event.

Owner events are authenticated using a digital signature. In the analysed version of the protocol, 0.03, the signature was computed over the concatenation of the user ids of the previous owner and the new owner, and the object id.

Three Tamarin rules are used to model the owner event. The creation of an object is modeled by the rule `create_object`. When an object is created the owner event signature is sent using the `Out` fact and the ownership is recorded using the fact `LocalOwner`. The rule `receive_owner` takes in a signature and records the stated object owner in a `LocalOwner` fact. Finally, the `change_owner` rule allows the owner of an already existing object to be changed.

Several restrictions are used to remove degenerate edge cases from the model. For instance the `create_once` restriction is used to make sure that an object can be created only once, and the `change_to_new_owner` restriction ensures that the new owner is not the same as the old owner in the `chage_owner` rule.

Lemmas

- `private_key_secret`: Ensure the confidentiality of the private key used for signing the event. This proves that the adversary cannot obtain the signing key and break authenticity that way.

- `local_owner_is_authentic`: Show that the origin of owner events is authentic. In this case, we proved that the signature provides non-injective agreement.

  Non-injective agreement means that the apparent owner at some point actually was the owner, however the signature is not unique per event. Because the signature may be reused, it is possible to fake ownership if one was previously the owner of the object through a replay attack.

  For example, Alice is the original owner and transfers ownership to Bob. If ownership later is transferred back to Alice, Bob can take ownership of the object once more by copying the signature from the first ownership transfer. An example of a replay attack such as this can be seen in Figure 5.1.

  A way to remedy this is to make signatures unique for each event. For

instance, an incrementing serial number could be added to each signature.

Result The signing key cannot be obtained by an attacker. The owner event provided a weaker form of authentication in the studied version of the protocol. In version 0.06 of the protocol, an incrementing number was added to the signature to enforce ordering of owner events and provide stronger authenticity, based on our suggestion. Reset events had a similar issue, which was also resolved in the same manner.

5.2 Access event

An access event modifies access rights to an entire object or for specific fields in that object. The event contains values which describe if access rights should be edited for the entire object, or only for specific fields. It contains information about which users should be given what rights.

Access events are authenticated using a digital signature. In version 0.03 the signature was computed over the concatenation of the affected label, the user id of the user making the change, the device number, a counter value, and the list of users and what access level they are granted.

The model for access events builds upon the model for owner events. Different access levels are not modeled since they do not affect the authenticity of the messages.

Three additional Tamarin rules model the access event. Granting access to an object is modeled by the `send_access_event` rule. It can be executed by a user who already has access to the object, who sends the signature using an `Out` fact. The `receive_access_event` rule is then used to receive and verify the event and signature. The `LocalAccess` fact is used to record the access level of each user. The rule `owner_to_access` acts as a bridge between the access and owner models, it transforms a `LocalOwner` fact into a `LocalAccess` fact so the owner is considered to have access as well.

Lemmas

- `can_receive`: This is a sanity check lemma. It verifies that a user with the correct access rights can grant another user similar access rights. For example, the owner of an object should be able to assign a user as an administrator of that object.

The lemma was not originally intended to test any security properties. However, the trace it generated made us realise that the access event does not provide full agreement. In the generated trace, two objects were created, but the access event signature from one object was copied from the other object. Since the signature did not include the object identifier, it could be used in a replay attack to modify access rights for another object. An example of a similar replay attack can be seen in Figure 5.1.

This is an authenticity problem because the signature may be reused in the wrong context. A consequence of weak authentication may be harmed integrity of access right metadata, as a user could incorrectly be given authorization to read, modify, or even delete an object which it should not have access to. This can be remedied by making the signature in the access event include the object identifier.

Result Because we discovered an issue with authentication during the modeling phase, we did not design lemmas to prove authenticity of access events. In version 0.06 of the protocol, the object identifier was added to the signature, based on our suggestion. Thus the identified vulnerability no longer exists.

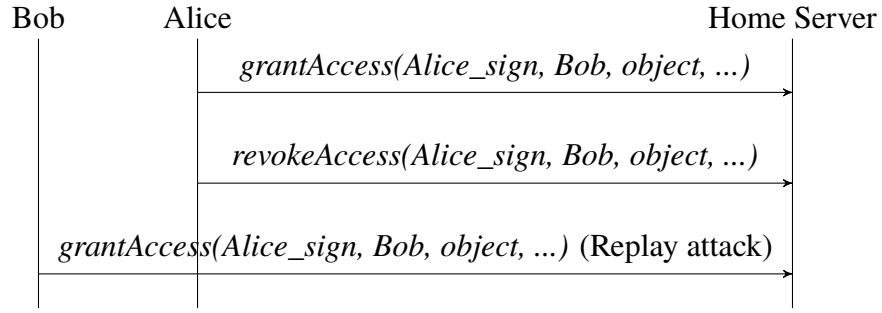

Figure 5.1: The diagram shows an example of a possible exploit which exists when signatures can be re-used between events. Alice wishes to grant some level of access of an object to Bob. Alice does this by creating and signing the event *grantAccessEvent* with *Alice_sign*. Shortly thereafter, Alice revokes the previously granted access with *revokeAccess*. Later, Bob copies Alice's signature and uses it to grant himself illegitimate access to the object.

5.3 Registration and authentication in basic mode

In the basic mode, users can register and authenticate with both stateless and stateful devices. Public key cryptography is not used, instead a user's identity is their email address or phone number. For the sake of brevity we will assume that email is used. The purpose of the following procedure is to verify that API requests to the home server originate from the claimed user. In this case, with lower security requirements, the email channel and HTTPS channel between server and client are assumed to be secure.

Registration and authentication starts with the email address being sent to the home server. The home server responds by sending a verification link to the specified email address. The link expires within 10 minutes. When the user clicks the link, the server can match the URL to the user id, and thus conclude the request to be authentic. In that case, the server returns a session key which is used to authenticate further requests.

Note that the security requirements for basic mode differ from the end-to-end mode. In the basic mode, the home server is assumed to be honest.

```
rule create_user:
  [ClickOnLink(link),
   ServerSentEmail($email, link),
   Fr(~sessionKey), Fr(~userid)]

  --[SessionKey(~userid, ~sessionKey),
     NewUser('t')]->

  [Session(~userid, ~sessionKey),
   !User(~userid, $email)]
```

Figure 5.2: Rule to create a new session key, once a user has clicked a verification email.

Seven Tamarin rules are used in the model. There is a direct correspondence between the rules `fresh_verification_code`, `click_on_link`, `verify_user` and `create_user` and the steps described in the specification. The additional rules `read_email` and `write_email` were used to test what would happen if the adversary could read or write to the email inbox.

The rule `reveal_user_email` gives the adversary knowledge of the email address for a given user id.

Lemmas

- `can_not_compromise_session_key`: The lemma states that if the adversary can not obtain the session key. This assumes that the adversary does not have access to the email inbox until after the session is established. Since the session key is used as a form of authentication in subsequent operations of the protocol, this ensures that authenticity is upheld.

Result Under the assumption that the user's email is secure during the procedure, an attacker cannot gain access to the session key. Thus, authenticity is upheld. This level of security is similar to services with email password reset.

Chapter 6

Conclusion

In this chapter, we answer our research question and present some of our general learnings from the project, as well as future work.

Our analysis of the end-to-end mode revealed two replay attacks on the authentication of an unpublished draft version of the SDS protocol. We can thus conclude that the authentication in the analysed version did not completely satisfy its intended security goals. The identified issues have since been resolved based on our suggestions.

Our analysis of the basic mode proved the security of the authentication process, assuming that the user's email is not compromised at the time of authentication. We can thus conclude that it satisfies its intended security goals.

Since we were able to identify real vulnerabilities in the SDS protocol, we believe that the presented methodology is a viable way to analyse security protocols in general. However, as the model grew larger, the automatic analysis of lemmas would often fail to terminate. This was partly remedied with the use of restrictions and by dividing the model into separate files, all analysed independently, but it is not clear that these methods are always viable. An experienced user of Tamarin Prover might be able to restrict and guide the analysis even further, but a beginner might find themselves at a loss for solutions for an analysis that refuses to terminate.

In some cases, the automated verification may not even be the greatest value of the methodology. Modeling a protocol in Tamarin Prover requires great attention to its details, and we could sometimes identify issues during modeling, before even running the analysis tool.

During the analysis work, we also learned the importance of a clear understanding of the security requirements of the studied protocol. Without the terminology and framework of concepts such as the CIA triad and Lowe's hi-

erarchy it is hard to do a relevant analysis. Such analysis would be easier if all protocol specifications included detailed statements of intended security guarantees, but that is probably unlikely in practice due to the amount of work required.

6.1 Future work

Our work lead to several changes in later versions of the protocol. To ensure the new authentication is correct, another analysis like the one we performed may be carried out on the current protocol version.

For a more complete picture of the security of the SDS protocol, additional security aspects should be analysed. The protocol is, apart from authenticity, also intended to provide confidentiality, integrity and authorization. These can all be analysed using the same method as in this report.

For a more thorough analysis, the protocol may wish to receive a Common Criteria Certification. With common criteria, a system is evaluated against a standardized document with security requirements. Evaluating a system with the use of standard security requirements makes it easier for potential users to compare a system with its competitors. [24].

Another direction is to see if this type of analysis can be made easier. For instance, if it is possible to implement the SDS protocol using ProScript [9] or other types of verification tools. Such a model could be used to verify our results or expand the scope by analysing other security aspects of the protocol.

Bibliography

[1] David Broman. *Secure Data Sharing Protocol, DRAFT-v0.09*. Available from https://www.consecio.com/. 2020.

[2] Michael Huth. *Logic in computer science : modelling and reasoning about systems*. Cambridge U.K. New York: Cambridge University Press, 2004. ISBN: 9780521543101.

[3] Leslie Lamport. *TLA+ Course Lecture: Introduction to TLA+*. http://lamport.azurewebsites.net/video/intro.html. (Accessed on 24/03/2020).

[4] Nuno Macedo and Alcino Cunha. *Alloy meets TLA+: An exploratory study*. 2016. eprint: arXiv:1603.03599.

[5] Chris Newcombe et al. "How Amazon Web Services Uses Formal Methods". In: *Communications of the ACM* 58.4 (2015), pp. 66–73. ISSN: 0001-0782. DOI: 10.1145/2699417.

[6] Bruno Blanchet. "From Secrecy to Authenticity in Security Protocols". In: *9th International Static Analysis Symposium (SAS'02)*. Ed. by Manuel Hermenegildo and Germán Puebla. Vol. 2477. Lecture Notes in Computer Science. Madrid, Spain: Springer, Sept. 2002, pp. 342–359.

[7] D. Dolev and A. C. Yao. "On the security of public key protocols". In: *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*. Oct. 1981, pp. 350–357. DOI: 10.1109/SFCS.1981.32.

[8] ENS Paris-Saclay. *Cryptographic protocols: formal and computational proofs*. https://wikimpri.dptinfo.ens-cachan.fr/doku.php?id=cours:c-2-30. (University course description, accessed on 03/04/2020). 2019.

[9] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. "Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach". In: IEEE, 2017, pp. 435–450. ISBN: 978-1-5090-5761-0. DOI: 10.1109/EuroSP.2017.38.

[10] Jingjing Zhang et al. "Formal Analysis of 5G EAP-TLS Authentication Protocol Using Proverif". In: *IEEE Access* 8 (2020), pp. 23674–23688. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2969474.

[11] Stylianos Gisdakis, Thanassis Giannetsos, and Panos Papadimitratos. "SPPEAR: Security and Privacy-preserving Architecture for Participatory-sensing Applications". In: *ACM Conference on Security & Privacy in Wireless and Mobile Networks (ACM WiSec)*. Oxford, United Kingdom, 2014, pp. 39–50. ISBN: 978-1-4503-2972-9. DOI: 10.1145/2627393.2627402.

[12] David Basin et al. *Tamarin Prover Manual*. https://tamarin-prover.github.io/manual/index.html. (Accessed on 18/03/2020). 2020.

[13] Simon Meier et al. *The TAMARIN Prover for the symbolic analysis of security protocols*. http://people.inf.ethz.ch/basin/pubs/siglog-tamarin.pdf. (Accessed on 03/04/2020).

[14] David Basin et al. "A Formal Analysis of 5G Authentication". In: vol. 14. June 27, 2018, pp. 1383–1396. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243846. URL: http://arxiv.org/abs/1806.10360 (visited on 03/19/2020).

[15] Cas Cremers et al. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1773–1788. ISBN: 9781450349468. DOI: 10.1145/3133956.3134063. URL: https://doi.org/10.1145/3133956.3134063.

[16] Rob Seater et al. *Alloy Tutorial*. https://alloytools.org/tutorials/online/frame-FS-0.html. (Accessed on 24/03/2020).

[17] Tefvik Bultan. "A guide to alloy". (Accessed on 25/03/2020). URL: https://sites.cs.ucsb.edu/~bultan/courses/267/lectures/l18-1.pdf.

[18] Jason Andress. “Chapter 1 - What is Information Security?” In: *The Basics of Information Security*. Ed. by Jason Andress. Boston: Syngress, 2011, pp. 1–16. ISBN: 978-1-59749-653-7. DOI: https://doi.org/10.1016/B978-1-59749-653-7.00001-3. URL: http://www.sciencedirect.com/science/article/pii/B9781597496537000013.

[19] G. Lowe. “A Hierarchy of Authentication Specifications”. In: IEEE, 1997, pp. 31–43. ISBN: 978-0-8186-7990-2. DOI: 10.1109/CSFW.1997.596782.

[20] Sreekanth Malladi, Jim Alves-Foss, and Robert B. Heckendorn. “On Preventing Replay Attacks on Security Protocols”. In: 2002. DOI: 10.21236/ada462295.

[21] Barbara Y. Fraser. *RFC 2196: Site Security Handbook*. IETF, 1997. URL: https://tools.ietf.org/html/rfc2196 (visited on 05/27/2020).

[22] Jason Andress. “Chapter 5 - Cryptography”. In: *The Basics of Information Security*. Ed. by Jason Andress. Boston: Syngress, Jan. 1, 2011, pp. 63–80. ISBN: 978-1-59749-653-7. DOI: 10.1016/B978-1-59749-653-7.00005-0. URL: http://www.sciencedirect.com/science/article/pii/B9781597496537000050 (visited on 06/12/2020).

[23] Eric Rescorla. *REF8446: The Transport Layer Security (TLS) Protocol Version 1.3*. IETF, 2018. URL: https://tools.ietf.org/html/rfc8446 (visited on 07/06/2020).

[24] *Common Criteria for Information Technology security evaluation, introduction and general model*. https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf. (Accessed on 06/10/2020).

Appendix A

Tamarin example

This appendix will provide an example of Tamarin usage. It is meant to serve as a quick introduction for readers who are not familiar with the tool. We use the Diffie-Hellman key exchange protocol to illustrate the basics, because it is fairly simple.

First, we establish a model of the protocol using *rules*. Each rule has a name, and then a specification of the rewriting rule. The first bracketed list is the prerequisites, followed by optional action facts, followed by the resulting facts.

```
rule Gen_secret:
    [ Fr(~n) ]
  --[ Secret(~n) ]->
    [ Nonce(~n) ]

rule First1:
    [ Nonce(~n), Fr(~tid) ]
  -->
    [ Out('g'^~n), First_waiting(~tid, ~n) ]
```

In the example, we use the rule *Gen_secret* to specify that a nonce is generated from a fresh random value. We also label the value as secret. This will later be used in the proof. The rule *First1* defines the first part of the protocol. The first actor generates a nonce n and sends g^n out to the network. The value g is a public constant, which is denoted by surrounding it with apostrophes. We also use the fact *First_waiting* to remember in the state that the first actor is waiting for a response, and tie this fact to the unique thread id *tid*.

```
rule Second:
    [ Nonce(~m), In(fp) ]
```

```
  --[ SharedKey(fp^~m), Role('s') ]->
    [ Out('g'^~m) ]

rule First2:
    [ First_waiting(~tid, ~n), In(sp) ]
  --[ SharedKey(sp^~n), Role('f') ]->
    []
```

In the next step of the protocol, the second actor receives the message from the first actor and generates its own nonce. At this stage the second actor can calculate the shared key, which is stored as an action fact. Then the value g^m is sent back to the first actor. Now the first actor receives the message and calculates the same shared key.

With the model specified, we now proceed to specify the properties we want to prove. It is good practice to begin with a sanity check to make sure that the protocol as specified is executable. Here we say that there is a trace where both actors end up with a shared key. *Ex* is the existential qualifier and *#i* and *#j* are temporal variables, where *@i* means "at time *#i*".

```
lemma can_end_up_with_same_key:
    exists-trace
    "Ex k #i #j.
      Role('s') @i & SharedKey(k) @i &
      Role('f') @j & SharedKey(k) @j"
```

Next, we specify what the adversary cannot know. The lemma *secrecy* says that any value marked with the *Secret* action fact is not known by the adversary. The action fact *K(n)* denotes that the adversary knows *n*. In our case, we have said that the nonce is a secret.

For the shared key we have used another approach. Here we say that whenever there is a shared key, it is unknown by the adversary.

```
lemma secrecy:
    "All n #i.
      Secret(n) @ #i ==>
      not(Ex #j. K(n) @ #j)"

lemma key_secret:
    "All k #i #j.
      SharedKey(k)@i & Role('f')@i &
      SharedKey(k)@j & Role('s')@j ==>
        not(Ex #r. K(k) @r)"
```

This theory is now complete. It can be verified using the *tamarin-prover* executable.