DISS. ETH NO. 23433

SECURE END-TO-END COMMUNICATION IN REMOTE INTERNET VOTING

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

MICHAEL SCHLÄPFER

MSc ETH CS

born on 03.08.1978

citizen of Wald AR and Basel BS

accepted on the recommendation of

Prof. Dr. D. Basin
Prof. Dr. S. Čapkun
Prof. Dr. C. Schürmann
Dr. S. Radomirović

2016

Acknowledgements

First of all, I would like to thank Prof. David Basin (ETH Zurich) for his advice and continuing support during my time at ETH Zurich. I am thankful for the chance he gave me to be part of so many different interesting projects regarding research and teaching.

Alongside Prof. Basin, many other outstanding researchers were involved in this research or at least influenced it. To begin with, Dr. Saša Radomirović (ETH Zurich) were largely involved in the development of the formal extended security protocol models and the characterization of secure human-server communication. Prof. Melanie Volkamer (Technische Universität Darmstadt) was involved in the analysis of existing approaches to mitigate the Secure Platform Problem. Prof. Rolf Oppliger (Swiss Federal IT Steering Unit) and Dr. Oliver Spycher (Swiss Federal Chancellery) were technical advisors and supported this research with many fruitful inputs.

The members of the Swiss E-Voting Competence Center and the E-Voting Group of the Bern University of Applied Sciences gave me a lot of feedback and thereby improved this research. Together we came up with different relevant ideas and presented those at some of the most relevant conferences concerned with electronic voting. Special thanks go to my long-time office mate Dr. Andreas Fürst for his encouragement and companionship. Also my colleagues of the Institute of Information Security at ETH Zurich were always helpful whenever I needed support especially regarding using the Tamarin tool. Their feedbacks to this research improved it considerably.

Further, I would like to thank the co-examiners, Prof. Carsten Schürmann (IT University of Copenhagen) and Prof. Srdjan Čapkun (ETH Zurich) for their support and comments to further improve the final version of this thesis.

Finally, I would like to thank my wife Kathrin and our parents for their patience and support during the last years. During my time at ETH my son Lukas was born, to whom I dedicate this work.

Abstract

Electronic communication becomes more and more important in our everyday life. We use various kinds of computer systems and services to write e-mails, to search the Internet for information, and to manage all kinds of data. With this development and the growing rate of Internet users, more and more services are provided over the Internet. A security-critical application in this context is that of electronic voting over the Internet. The corresponding infrastructure is not entirely under the election authority's control. Rather the voters' personal computers and public networks are used to vote and to submit the vote. Many of the current solutions neglect the fact, that a voter's personal computer may be compromised and that the voter may make mistakes while voting. Although these problems are not specific to Internet voting but an inherent problem in any security-critical communication application, only few research results exist with this respect.

In this context, the Swiss Federal Chancellery's *Vote électronique* project aims to gain a deeper understanding of the possible security problems imposed by insecure client computers and erroneous user behavior. The corresponding questions serve as starting points for the research in this thesis. We split the client-side security problems regarding electronic communication applications into two problem areas: *insecure platforms* and *human error*. Regarding these problem areas, we provide formal methods for the automated analysis of communication protocols with respect to the corresponding problem area. We use these formal methods to characterize secure electronic communication using insecure client computers and examine the influence of erroneous user behavior.

Zusammenfassung

Elektronische Kommunikation spielt eine zunehmend bedeutende Rolle und ist aus unserem täglichen Leben kaum mehr wegzudenken. Wir verwenden unterschiedliche Dienste und Systeme um Informationen im Internet zu suchen, um Nachrichten auszutauschen und um Daten zu verwalten. Mit der wachsenden Bedeutung des Internets werden auch immer mehr Programme und Dienste angeboten, welche Daten über öffentliche Netzwerke austauschen. Eine sicherheitskritische Anwendung in diesem Zusammenhang ist elektronisches Wählen über das Internet. Hierbei wird die verwendete Infrastruktur nicht komplett durch die Wahlbehörden kontrolliert. Vielmehr werden die Computer der Wähler und öffentliche Netzwerke verwendet, um die Stimmen abzugeben und zu übermitteln. Viele aktuelle Lösungsansätze ignorieren hierbei den Umstand, dass der Wähler einen kompromittierten Computer verwenden oder auch einfach einen Fehler in der Bedienung machen könnte. Obwohl diese Probleme nicht nur Wahlsystemen, sondern allen elektronischen Kommunikationsanwendungen zu Grunde liegt, wurden sie bislang wenig erforscht.

Vor diesem Hintergrund zielt das *Vote électronique* Projekt der Schweizerischen Bundeskanzlei darauf ab, ein vertieftes Verständnis für den Einfluss unsicherer Computer und fehlerhaften Verhaltens der Benutzer auf die Sicherheitseigenschaften von elektronischen Wahlsystemen zu entwickeln. Die diesbezüglichen Fragestellungen dienen als Grundlage für die vorliegende Forschungsarbeit. Wir teilen die benutzerseitigen Probleme elektronischer Kommunikationsanwendungen in zwei Problembereiche auf: *Unsichere Benutzersysteme* und *menschliche Fehler*. Für beide Problembereiche präsentieren wir formale Modelle um die automatisierte Analyse von Kommunikationsprotokollen im jeweiligen Problemkontext zu ermöglichen. Wir verwenden die formalen Modelle um sichere Kommunikation über unsichere Benutzersysteme zu charakterisieren und den Einfluss fehleranfälliger Benutzer zu untersuchen.

Contents

List of Figures

List of Tables

1 Introduction

Electronic communication plays an important role in our everyday life. We use various kinds of computer systems to write e-mails, search for information on the Internet, and manage all kinds of data. With the development of the Internet and the growing rate of Internet users, more and more services are provided over the Internet. Businesses use these opportunities not only to facilitate and to accelerate their communication but also to automate tasks such as processing orders and offering their customers access to their services day and night. In the same manner governments have started to provide their services to their citizens.

Internet users are accustomed to accessing all these services at any time and, with the increasing importance of mobile devices such as smart phones and tablet computers, at any location. It is therefore natural that citizens and governments ask for services to exercise political rights over the Internet too. However, electronic communication is exposed to several threats, which have to be considered carefully.

A security-critical application in this context is that of electronic voting. Here, the election authority uses electronic devices to record and tally the votes. This is an active research area and several proposals for secure electronic voting exist. However, providing such services over the Internet is even more complex and proposals often neglect the realistic threats imposed by the voters and their insecure personal computers. Although these problems are not specific to Internet voting but an inherent problem in any security-critical communication application, they have not yet been researched on a large scale.

In this context, the *Vote électronique* project of the Swiss Federal Chancellery aims to gain a deeper understanding of the possible problems imposed by the client side and the implications on Internet voting in general. The questions with respect to client-side aspects raised by the Vote électronique project served as a starting point for the research in this thesis.

The need for secure end-to-end communication is not specific to Internet voting but inherent to any security-critical communication application. Therefore, this is an active and growing research area. In the following section we give an overview of current results with respect to secure end-to-end communication and we identify open issues that we will focus on in this thesis.

1.1 Secure End-to-End Communication Research

Security-critical communication applications rely on secure communication channels. These channels are constructed using security protocols that protect the messages exchanged between protocol agents. The properties to protect depend on the application. Typically, the communicated message is required to remain confidential and unchanged.

Designing security protocols is a non-trivial task. One of the reasons for this is the difficulty to assess all effects of a protocol's execution in the presence of an active adversary. That is, of an adversary controlling the network and with the ability to eavesdrop on, block, and modify all messages exchanged. Secure communication between computers over insecure networks has been researched for decades and its fundamental problems are generally well understood. However, for a practically relevant class of communication applications, at least one communication end is a human. Examples of such applications include online banking and Internet voting.

In settings where humans actively participate, secure communication is even more difficult to achieve. Humans are limited with respect to their computational capabilities and therefore must rely on computing devices as shown in [14], such as their personal computers, mobile phones, or tablets. Unless the computing device's hardware and software are trustworthy, information appearing on the device's screen may not faithfully represent the messages communicated with the remote system. Moreover, the computing device may leak information to unauthorized third parties [72, 32].

To circumvent human limitations and the problem that a user's computing device is generally not trustworthy, trusted supporting technology is used. Examples of such supporting technology include security tokens and smart-cards but also paper-sheets containing codes, such as transaction authentication numbers (TAN). A variety of communication systems where humans, computers, and supporting technologies communicate have been proposed and are even used on a large scale. Most prominently, in online banking and Internet voting.

Most existing research on such systems and protocols focuses on particular scenarios, for instance on browser-based security protocols [36, 43], login procedures [47], solutions for online banking [101, 100] or Internet voting [66, 82]. However, the generic problems imposed by insecure computing devices and honest human users are not well understood yet. Therefore, we investigate how to model a system where humans, computers, and supporting technology communicate, and how to analyze such a system's security properties. Since currently no results on necessary and sufficient conditions for trusted supporting technology and corresponding protocols exist, we focus on formal methods and foundation results with this respect.

An attempt to better understand systems where humans and computers communicate is that of security ceremonies [30, 96]. Security ceremonies are a generalization of security protocols. They extend communication protocols to include actors and communication means that belong to the context in which security protocols are executed, but whose properties and behaviors have traditionally not been considered

in conventional security protocol models. For instance, a human and his computer are actors and the visual channel between the human and the computer screen is a communication channel in a ceremony.

With respect to security ceremonies, a variety of models and formal methods with different focal points exist, such as a logic to reason about objects, their location and movement [57] or a division of security ceremonies into layers. The latter approach seems to be promising since the complexity of the problem can be split into smaller problems. Bella and Coles-Kemp provide the first steps with this respect and interpret the nothing-is-out-of-band requirement even more broadly. They extend security ceremonies with technical and social elements such, as a human user's belief system and cultural values [10, 11]. They propose modeling security ceremonies using five layers: (1) the security of the protocol executed by the computers of the communicating partners; (2) the inter-process communication of the operating system; (3) the socio-technical protocol whereby a user interacts with a graphical user interface; (4) the user's state of mind and (5) the influence of society on individuals. In [11], they formalize layer (3), which is responsible for human-computer interaction. There model allows one to verify a user's confidence in the privacy assurance offered by service providers. We will take a similar approach in this thesis in that we identify two problem areas of secure end-to-end communication: *insecure platforms* and *human error*. We examine these areas independently.

Even if a system where humans, computers, and supporting technology communicate is provably secure under the assumption that a honest user correctly executes the protocol, in reality the human user may accidentally deviate from the protocol. This deviation may have a negative effect on the communication system's security properties. For example, a user may not verify transaction details when he is expected to do so. Possible reasons for such deviating behavior are manyfold and often caused by misunderstandings or a lack of knowledge. Because humans are error-prone and unreliable, secure communication applications must take into account the expected end-user's perception or mental model of the application's security features. The application's user interface must focus on usability to favor the user's compliance with the protocol.

To address the problems related to human error, Cranor provides "A Framework for Reasoning About the Human in the Loop" [25]. This framework offers a systematic approach for the design of security-critical systems. Particularly, to identify potential causes for human failure. She proposes a four-step iterative process to identify and mitigate human threats. Whereas Cranor's framework helps in not missing important flaws, one problem is that the task of identifying potential failures depends on the expertise of the secure system designer. Moreover, only known and obvious failures may be identified. However, protocol flaws may remain hidden for a long time before they are unveiled by systematic reasoning. In this thesis we take another approach and we introduce a generic formal human error model that allows one to identify human-error caused flaws in communication applications.

Based on Curzon and Blandford's formal user model [26], Rukšėnas et al. [77]

examine cognitive processes that affect the confidential information flow from the human user into the computer system and the system's resilience to intruder attacks. However, they focus on confidentiality and the adversary's goals must be modeled explicitly for each individual application.

1.2 Research Questions and Methodology

The aim of this thesis is to improve the understanding of security implications with respect to client-side aspects in secure end-to-end communication applications in general and on Internet voting systems in particular. More precisely, we focus on systems where humans, insecure computers and servers communicate. The questions whether or not it is possible to mitigate the problems related to client-side issues and if so, to what extent, build the basis for this research.

Scope of the Thesis

We identify two problem areas of client-side communication: *insecure platforms* and *human error* and study these areas independently. Our main focus lays on secure communication using insecure platforms, where we see the main contributions of this thesis. Figure 1.1 gives an overview of this thesis' scope. In the following we define the corresponding research questions for each of the two problem areas and we point out the applied methodologies we use to answer them.

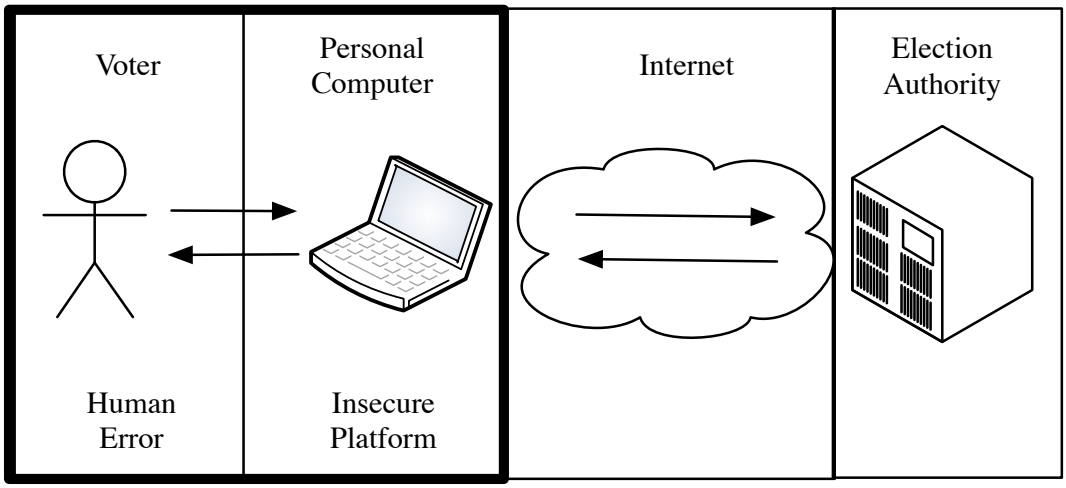


Figure 1.1: Secure end-to-end communication with an emphasis on the client-side.

Insecure Client Platforms

First, unless the personal computer's hardware and software are trustworthy, information appearing on the computer's screen may not faithfully represent the messages communicated with the remote system. Moreover, the personal computer may leak information to unauthorized third parties [32, 72].

It is difficult to model systems where humans, computers, and supporting technologies communicate and a variety of focal points for modeling exist. Our focal points are determined by the following questions.

Research Question 1 *What are the implications of insecure client platforms on communication protocols? How is the trustworthiness of specific protocol steps or function executions related with the protocol's security properties? What are necessary and sufficient conditions for secure human-server communication?*

We consider a setting that is more abstract and more precise than security ceremonies as described in [30]. We provide a formal model where we represent human agents by nodes that have restricted capabilities. Human-human and human-machine channels are simply links between nodes. In contrast to ceremonies, we do not impose a nothing-is-out-of-band requirement but allow to assume that a certain settings has already been set up before the analysis. Hence, our approach allows one to focus on the security properties of a specific part of a protocol.

The model allows one to reason about the security properties of protocols where humans, computers, and supporting technology communicate. Using the model, we completely characterize secure human-server communication and identify the necessary and sufficient conditions for establishing secure channels between a human user and a remote server system.

Human Error

Humans are error prone and their actions may deviate from a protocol designer's expectations. Thereby, the protocol's security properties may not hold. There are various reasons why humans may not comply with a protocol specification. The end-user's perception or mental model of the application's security features may simply be wrong. In this thesis we take a generic approach. We do not reason about the likelihood of deviations, which would require empirical studies for specific applications. Rather, we are interested in general answers to the following questions.

Research Question 2 *How can human users deviate from a security protocol? What are the corresponding negative effects on the protocol's security properties? What are necessary or sufficient conditions for specific security protocols to be robust against the identified human errors?*

We provide a comprehensive study of human error with respect to communication protocols where humans and computers communicate. We define a formal human-error model and analysis methods for different classes of human error. The model

allows one to verify the security properties of a given protocol under the assumption that human users deviate from the protocol. The result is a systematic process to analyze a protocol's robustness against different human-error classes and necessary or sufficient conditions for robustness with respect to certain human errors.

1.3 Contributions

Starting from the open issues in end-to-end security research and the research questions above, this thesis provides contributions in both problem areas introduced above. The main contributions are the results regarding insecure client platforms [6, 82] and human error. Most of these results are independent from Internet voting applications but are general results for any security-critical communication application where humans and computers communicate. A minor contribution is the comprehensive survey on requirements and recommendations regarding Internet voting systems in Section 2.2. In the following we describe the contributions in more detail.

Insecure Client Platform

We provide a taxonomy of approaches to mitigate the problems related to insecure client platforms and we analyze existing approaches with respect to a strong adversary model, where the adversary controls the user's platform. The resulting survey is a structured overview over the mitigation approaches and we identify possible applications for Internet voting.

We introduce a communication topology model on top of an operational semantics for security protocols. Our topology model formalizes the environment in which protocols are executed and allows one to reason about communication systems at different levels of abstraction. We use the model to completely characterize necessary and sufficient conditions for the existence of security protocols that provide secure channels between a human and a remote server using an insecure network and a dishonest platform. This characterization allows one to quickly assess whether a particular protocol design and supporting technology can plausibly offer secure communication.

Our characterization can be used to guide the design of novel solutions for establishing secure channels between humans and a remote server. The developer uses our characterization to first analyze whether the proposed solution satisfies the necessary conditions for secure communication. If it does, the developer obtains from the characterization one or more minimal communication topologies. Afterwards a communication protocol is designed that employs all communication links appearing in the chosen topology. The protocol is then formally specified in our specification language in order to analyze its security properties with the Tamarin [84] verification tool. This design methodology supports the verification of a large class of distributed algorithms running on nodes communicating over links. In particular, it supports the automatic verification of confidentiality and authenticity of information exchanged between nodes in security protocols involving humans and their insecure platforms.

Human Error

We examine existing analysis approaches and apply a systematic approach to identify possible human misbehavior in the context of communication applications. Based on the resulting list of human errors, we introduce a formal model and analysis methods to reason about the negative effects on the security properties of communication applications caused by the identified human errors. Our methods and tools facilitate the task of examining secure communication applications regarding potential human failures. Thereby designers of such applications are made aware of the assumptions regarding human compliance with the protocol and they find possible flaws and fixes with this respect.

We analyze various existing protocols and show that they are not robust against certain human errors and we explain how they may be fixed to improve robustness. We provide foundational results and characterize requirements on human compliance with an application's protocol specification in order to provide human-error robustness.

So far, designers of secure systems had to manually consider possible flaws with respect to human errors. Our methods allow one to automatically compare communication applications with respect to their relative strength regarding different human errors. We provide a protocol specification language and tool support in order to analyze the protocol's security properties with the Tamarin [84] verification tool. We provide a number of examples to demonstrate our approach's applicability.

Related Contributions

During this research we contributed to other research areas related to Internet voting. In [81, 91, 92], we address the computational complexity problem of coercion-resistant Internet voting protocols. We analyze existing approaches and recent improvements with respect to their computational requirements. We developed two different protocols based on the scheme of Juels et al. [50]. Our protocols improve vote authorization and allow the application of coercion-resistant Internet voting in large scale settings. We provide a comprehensive runtime analysis where we relate our protocols to the current state-of-the-art protocols.

1.4 Publications and Collaboration

In the context of this research, the author contributed to a number of research works in collaboration with other researchers. Many of the research results in this thesis are published and were presented at scientific conferences. In the following we provide an overview of the relevant publications and the corresponding collaboration with other researchers. We state what the author's contributions are and how the results are used in this thesis.

A Complete Characterization of Secure Human-Server Communication [6]

Technical report and research paper published at the Computer Security Foundations Symposium (CSF) 2015 in collaboration with D. Basin and S. Radomirović. The conceptual idea of a topology model and a corresponding characterization is based on the author's previous taxonomy of mitigation approaches regarding the secure platform problem. Under supervision and with the co-authors' support, this thesis' author contributed to all parts of the work, with an emphasis on the model, the possibility results, and the application of the Tamarin tool for the corresponding automated proofs. The results build the foundation of Part I of this thesis, where we additionally provide case studies to show our model's applicability for the development and the analysis of security protocols where humans, computers, and supporting technologies communicate.

Efficient Vote Authorization in Coercion-Resistant Internet Voting [81]

Research paper published and presented at the VoteID conference 2012 in collaboration with R. Haenni, R. Koenig, and O. Spycher. The concept of ballot replication for efficient coercion-resistant Internet voting was a joint idea of the paper's authors and it is based on a number of workshops. The author of this thesis was the paper's first author. He developed the detailed protocol description and contributed to all parts of the paper, supported by the co-authors. This thesis' author finally presented the work at the VoteID conference in Tallin, Estonia.

The secure platform problem: Taxonomy and analysis ... [82].

Research paper published and presented at the International Conference on Theory and Practice of Electronic Governance (ICEGOV) 2012 in collaboration with M. Volkamer. The survey of existing mitigation approaches with respect to the secure platform problem is based on the idea and initiative of this thesis' author. He was the first author of the paper, contributed to all parts of it, and presented the work at the ICEGOV conference in Albany, NY, USA. Prof. M. Volkamer supported the author with the description of additional mitigation methodologies, structural improvements, and with proofreading the paper.

A new approach towards coercion-resistant remote e-voting in linear time [91]

Research paper published at the Financial Cryptography and Data Security (FC) conference 2011 in collaboration with R. Haenni, R. Koenig, and O. Spycher. The concept of voter roll indicators and fake-vote generation for efficient coercion-resistant Internet voting was a joint idea of the paper's authors and it is based on a number of workshops. This thesis' author contributed to the development and description of the enhanced Internet voting protocol and to the analysis of the protocol of Juels et al. [50].

Achieving meaningful efficiency in coercion-resistant, verifiable internet voting [92]

Research paper published at the Electronic Voting conference (EVOTE) 2012 in collaboration with R. Haenni, R. Koenig, and O. Spycher. The enhanced protocol is based on a joint development of the paper's authors and it is based on a number of workshops. This thesis' author contributed to the development of the protocol and to the description of the cryptographic primitives.

1.5 Structure of the Thesis

Background

In Chapter 2 we give an overview of Internet voting systems. We explain the voting process and the building blocks before we survey the common security requirements of Internet voting systems. We introduce the client-side threats against Internet voting and general considerations with respect to security.

Chapter 3 introduces background information on formal modeling and summarizes the security protocol model we use to build our formal methods on.

Part I: Insecure Client Platform

This part addresses the problem of insecure client platforms. In Chapter 4, we introduce the *Secure Platform Problem adversary* and explain the implications on the security requirements for Internet voting systems. Afterwards, we provide a taxonomy of mitigation approaches and classify and analyze existing approaches. We then, relate the different approaches to model the problem and give an overview of related work with respect to insecure client platforms.

In Chapter 5, we introduce our extended formal security protocol model. We introduce our formal approach to define channel goals and our protocol specification language.

In Chapter 6, we introduce a novel communication topology model, which we use to completely characterize *human-interaction security protocols* (HISPs). This is the class of security protocols where humans communicate with remote servers using an insecure computer but additional trusted supporting technologies. These foundational results define the necessary and sufficient conditions for communication topologies and protocols to provide secure communication.

We use the extended security protocol model and the characterization in Chapter 7 to introduce a novel methodology to guide the formal development of HISPs. We provide two case studies to illustrate this methodology. Afterwards, we extend the basic security goals introduced for our characterization with *individual verifiability* and analyze a reference Internet voting protocol provided by the Swiss Federal Chancellery.

Part II: Human Error

This part is concerned with the negative effects of human error. In Chapter 8, we introduce human error and resulting implications on security protocols. We identify common failure modes in security protocols with respect to erroneous users and relate our work to existing research work.

Chapter 9 contains the details on our formal human error model as extensions of the formal security protocol model defined in Part I. We provide additional concepts for human knowledge, explicit comparison, and human interaction channels. We provide rewriting rules to model the identified failure modes and introduce our human-error analysis process. Applying the formal human error model, we provide various examples as well as necessary or sufficient conditions for security protocols with respect to robustness against different human errors.

To show the applicability of our analysis process, we analyze a number of existing protocols as examples. We show that these protocols are not robust against certain human errors and how they may be improved regarding human-error robustness.

Conclusion

In Chapter 10, we summarize our work and point on interesting future research directions based on our findings and the tools we created.

Detailed Specification Files

We provide all the detailed specification files referenced in this thesis at `http://www.infsec.ethz.ch/research/projects/hisp`.

2 Internet Voting

In elections and referenda the ultimate goal is to correctly capture the legitimate voters' intention in order to decide what option the majority of the voters prefer and to convince the minority of their defeat, that is, of the correctness of the result.

For many years, only manual procedures were used. For example, voters were provided with a paper ballot with which they expressed their intention. The voters where expected to visit an official election office to authenticate themselves to election officials and to cast their votes in the privacy of a physical voting booth. During the last years, options for casting the votes by electronic means were discussed and even implemented in smaller contexts. The motivations for this effort are manyfold. First, manual tallying is intricate and error prone, thus the use of electronic systems may reduce unintended human errors and fraud during the tallying phase. Second, a society in which almost every procedure and service is automated and accessible using the Internet demands that existing technologies used for online banking and other e-government services are adapted for Internet voting too. Covering the first motivation, proposals for automating the election processes at local election offices were presented, that is, for selecting the candidates, recording the votes, and tallying. The second motivation lead to controversial proposals intending to provide secure voting over the Internet. However, there are some crucial differences between the security requirements for applications such as online banking or e-commerce services and those for Internet voting.

In this chapter we focus on these differences. We first introduce the processes and building blocks of Internet voting systems in Section 2.1. In Section 2.2 we provide a survey of common requirements for Internet voting systems. Finally, in Section 2.3 we identify the threats against Internet voting systems that we focus on in this thesis.

2.1 Internet Voting Systems

Voting Process

There are different ways to divide the electronic voting process into its essential phases. In this thesis we split the voting process into the following phases: *preparation*, *casting*, *recording*, and *tallying*. We depict the process in Figure 2.1 and describe its phases in detail below.

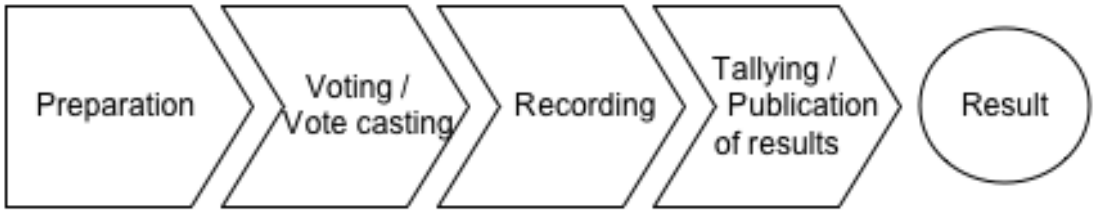


Figure 2.1: Phases of Internet voting.

Preparation. In the preparation phase, the set of eligible voters is determined, the possible candidates or options are defined, the ballots are prepared and the system is initialized. For example, empty ballots and necessary material (such as code sheets) are distributed to the electorate. In some Internet voting schemes a specific registration phase during which the eligible voters register themselves is proposed. Thereby, the electronic ballot may be authorized by the election authority.

Voting / Vote casting. In this phase the voter chooses a candidate or voting option and he commits to this choice by preparing a corresponding electronic ballot. He then casts the ballot to the election authority's server, which presumably receives this ballot.

Recording. The election authority's server stores the received ballot in an electronic ballot box. This ballot box can either be publicly accessible, for example a public bulletin board, or it can remain confidential. In the latter case, the ballot box may still be accessible by eligible officials such as auditors or observers.

Tallying / Publication of results. The ballots in the ballot box are prepared (for example decrypted or re-encrypted) such that they can be tallied to compute the result, which finally is being published.

Building Blocks

In terms of a functional division, Internet voting systems consist of different subsystems. In this thesis we use the following compartmentalization: the *voting server*, the *network*, and the *voting platform*. Furthermore, the voter as a communication endpoint is a part of the voting system too. Each subsystem is actively researched with respect to its security properties and different approaches for analyzing and reasoning about different security issues exist. In this thesis we focus on a combination of the described building blocks towards the implementation of the Internet voting process introduced above in this section. Figure 2.2 depicts the building blocks. In the following, we describe the building blocks in more detail.

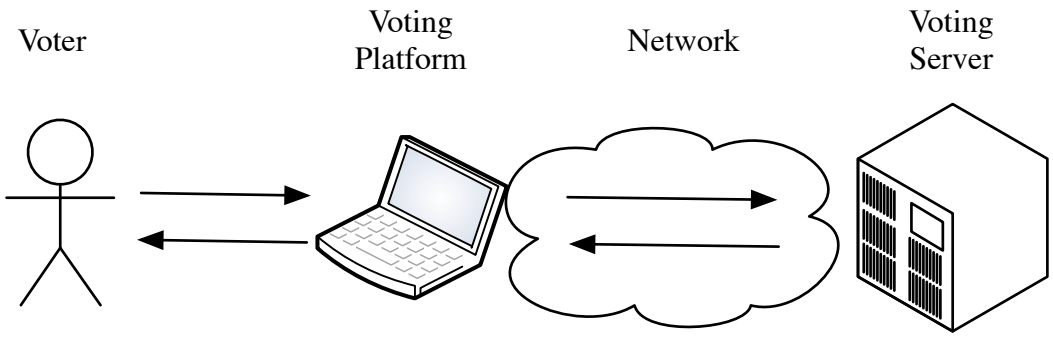


Figure 2.2: Building blocks of an Internet voting system.

Voting Server. The technical infrastructure provided by the election authority represents the voting server system. This infrastructure is used to receive and store (record) the voters' (encrypted) ballots in the casting and recording phases, respectively. In the tallying phase the voting server system provides the services necessary for the validation of the ballots and for the tallying and the publication of the results. Challenges for developing secure voting server systems and protocols include to ensure voter privacy while tallying, that no ballots may be injected (ballot stuffing) or removed unnoticed, and that the ballots and results may not be manipulated by the voting server system or the election authority at any time.

Network. The network represents the entirety of communication channels and devices that are used to initialize the systems and the election, to create, submit, receive, and store the ballots, and to tally and publish the results. While parts of the network may be under the election authority's control, it is inherent to Internet voting that some parts are not. In Internet voting it is completely unclear which routes the submitted ballots take, hence, the part of the network that is not under the election authority's control must be assumed to be controlled by an adversary. Secure electronic communication over insecure networks is being actively researched and many approaches and solutions for various problems with this respect exist. However, we will later see, that a common assumption for these solutions are trustworthy computer systems and that this is not a viable assumption for many applications.

Voting Platform. The voting platform provides the interface between the voter and the Internet voting system. Electronic voting systems with voting platforms under the election authority's control are actively researched and several different proposals exist, e.g., [22, 23, 51, 80]. Some of these solutions propose the use of *direct recoding equipment* (DRE), either in the protected environment of an election office or outside of such an environment in the so-called *kiosk voting* setting. The impact of possibly insecure voting platforms in the context of Internet voting is not yet largely researched.

Voter. The voter as part of the voting system is one of the actual communication endpoints. The voting server system is the other. A voting protocol's goal is to establish a confidential and authentic channel from the voter to the voting server system. Establishing a secure channel between users and computers is mainly researched with respect to usability, e.g., [25, 52]. However, even if a secure and usable channel exists, the voter is error prone and he may be distracted and manipulated. We focus on these problems in Part II of this thesis.

2.2 A Survey on Common Requirements

In general, it is often required that an electronic voting system is as secure as the manual systems that are already in place. For example, in countries where the citizens are offered absentee voting using postal mail, an electronic voting system is required to fulfill similar security requirements as absentee voting. However, the risks of absentee voting in general, no matter whether performed electronically or by postal mail, include several additional threats like voter coercion and vote buying as we will describe in Section 2.3. With this respect, we show in [81, 91, 92] that Internet voting can offer even stronger security properties than those of the systems that are currently applied.

In this section, we provide a comprehensive survey of the common requirements for Internet voting systems as proposed by legal experts and Internet voting researchers. The survey serves as an overview for further research work based on this thesis, for example, to formalize additional security properties. In the remainder of this thesis, however, we will focus on a set of general channel properties. We identify these channel properties in Section 4.1, where we relate them to a number of requirements introduced in this section.

Legal Foundations

The four fundamental principles of any democratic voting procedure [60] are: *universal*, *equal*, *free*, and *secret suffrage*. Many more specific and technical requirements can be inferred from these principles. In the following, we first describe the principles in more detail, before we summarize common recommendations and specific security requirements for Internet voting systems for political elections as proposed by researchers and legal experts.

Universal suffrage Every eligible voter shall have the ability to cast his intention.

Equal suffrage Every voter's vote shall have the same impact on the result.

Free suffrage Voters shall be able to cast their intention absolutely free and without any influence or pressure.

Secret suffrage Every measure shall be taken in order for the voter's choice to remain secret.

Common Recommendations

Several works identify security requirements for electronic voting settings. Many of these works focus on a specific subset of general requirements and propose specific implementation methodologies [42, 88]. However, many recommendations are vague and leave several points open for interpretation.

Grimm et al. identify a number of security requirements for electronic voting systems [39, 99]. Their results are summarized in a *Common Criteria* protection profile for non-political online voting systems [37]. In [41] they propose to formalize the security properties defined in the protection profile and provide simple examples for their methodology. Higher evaluation assurance levels require the application of formal methods. However a full implementation of their formal approach is questionable because of the complexity of all the proposed requirements. Instead, formal definitions should be defined on distinct security critical parts of electronic voting systems.

The Council of Europe's Committee of ministers adopted a recommendation on legal, operational and technical standards for e-voting [24]. This recommendation has been criticized for incompleteness, redundancy and contradicting requirements [56]. However, in contrast to the aforementioned works, this recommendation covers legal requirements for electronic voting systems in a political context. Therefore, the recommendation builds a corner stone for the implementation of specific Internet voting systems for legally binding elections and referenda.

More specific recommendations for different legal-frameworks exist. For example, [16] and [17] summarize the Swiss legislation's implications on electronic voting and they define corresponding requirements for electronic voting systems and applications in Switzerland.

In general, a multidisciplinary set of common requirements is meaningful and it builds the foundation for specific proposals and implementations. However, legal experts also criticized some of the recommendations because they are often only built on top of legal considerations without considering practical aspects of electronic communication systems. Therefore, some technical requirements contradict others while other requirements are open to broad interpretation and some are even implied by other stronger requirements. In [56] McGaley and Gibson give some examples for such ambiguities in the Council of Europe's Committee of ministers' recommendation and they propose a refined recommendation.

Skagestein et al. [90] criticize legal requirements for often being formulated with certain implementations in mind. Thereby, such formulations may exclude technical solutions that could provide even stronger security properties. They point out that it is desirable not to exclude technical solutions to legal problems by defining overly-specific laws, i.e., by including implementation details into these laws. An example is the principle of *vote-updating*, which allows voters to cast several ballots. This allows a coerced voter to vote again when no longer under the influence of a coercer. However, this might contradict the Council of Europe's recommendation with respect to the one-voter-one-vote requirement. With this respect, the current legal situation in many countries is not yet clear and must be clarified by the corresponding courts.

We structure the widely accepted general security requirements into the following five classes: *democracy*, *privacy*, *fairness*, *accuracy*, and *verifiability*. Next, we describe these classes and the corresponding requirements in more detail.

Democracy-Related Requirements

The goal of the requirements in this class is that only authorized voters can vote and only once.

Eligibility. Exactly the eligible voters are able to cast their vote.

One voter one vote. Every eligible voter is able to cast exactly one vote. The voter's possibility to cast several ballots seems to be important to achieve coercion-resistance and resistance against vote buying. However, it is not clear whether or not the principle of vote-updating contradicts the requirement of one voter one vote, since one can argue that in such a setting the casted ballots are pre-stored on the authority's server and only after closing the electronic vote casting phase, the last submitted ballot is put into the actual electronic ballot box for the tally, i.e., every voter is indeed able to only cast one vote. We recommend to specify this requirement as: One voter one *tallied* vote.

Generality. Equality of voters and equality of candidates, i.e., every eligible voter has the same influence on the final result and every candidate or choice is given the same chance to be chosen.

Privacy-Related Requirements

The voter's choice remains secret and cannot be linked to the voter (anonymity) and the voter cannot prove his individual choice to any third party (receipt-freeness).

Secrecy. The voters' individual votes remain secret, hence it is not possible in any phase to gain information about an individual voter's vote, not even for the election authority. It is only possible to prove secrecy with respect to a predefined adversary. Therefore, secrecy must be defined with respect to a specific threat model conceived from a thorough risk analysis.

Receipt-freeness [12]. No information is given to the voter that allows him to prove any information about his individual choice to any third party. This includes the impossibility to prove what he did not vote for or that he voted at all. Note that this requirement is not explicitly considered by the Swiss law. The system itself must not provide any official information that proves to any third party, what the voter might have voted for. However, in a remote setting and using photo- and videocameras, the only known feasible way to achieve this (at least to a certain extent), is the principle of vote-updating.

Coercion-resistance [49]. The voter cannot be coerced by any third party to cast a vote according to the coercer's instruction or to not participate at all. Coercion-resistance and resistance against vote buying are related. In fact, the vote buying problem is a special case of the coercion problem, where the voter intends to collaborate with the adversary for a compensation. The adversary may be physically present to observe and instruct the voter. Regarding Switzerland, large-scale vote buying is a more serious threat than large-scale physical voter coercion due to the fact, that the level of safety and protection against physical violence is very high.

Manipulation-protection. The voter cannot be influenced or manipulated during the voting process through any source of information. Using multi-purpose computing equipment, it seems to be difficult to enforce this in general, because several applications and the voting application may run in parallel. The voting application itself must be designed to not influence the voters behavior or intention.

Non-observation. The voter cannot be observed during the voting process or at least not during casting the intended final ballot. In a remote setting, it cannot be enforced that the voter cannot be observed during the casting phase. Therefore, this requirement must be made more precise. The principle of vote-updating is an option to decrease the adversary's confidence into the observation, since the voter may vote differently at a later time.

No time pressure. The voter must not be set under time pressure during the voting process. As there is a specific pre-defined deadline for casting the electronic ballots, voters trying to vote just shortly before the deadline could face some time pressure to do so. However, if the system is hybrid and therefore embedded into existing voting procedures, eligible voters will still be able to cast their vote using another channel (e.g., physically at the election office).

Fairness-Related Requirements

At no time during the election any intermediary election result shall be obtained.

No intermediary results. During the election, no information is leaked that allows any party to derive the status quo of the election or referendum. Information used by official observers or auditors during the election must not reveal tendencies with respect to the final result.

Accuracy-Related Requirements

The system behaves correctly, i.e., every eligible voter's vote is used in the tally and cannot be altered or removed. No invalid vote is counted in the tally.

Ballot casting assurance [1]. Every voter's vote is being cast as intended and recorded as cast, hence recorded as intended and voters gain confidence into this fact. The

voters shall be convinced, that their individual vote has been submitted correctly. This can be achieved by the voters' trust into official observers and auditors or through individual verifiability, which is a stronger property and allows the voters to verify the correct processing of their ballots.

Correctness of tallying. The tally of the recorded votes is correctly performed. Universal verifiability or auditability allows to verify all computation steps and intermediate results of the tally. The intermediate computation results can be made public or accessible by a set of trusted auditors.

Valid votes. All valid votes are being counted but only valid votes. Especially ballot stuffing shall not be possible, i.e., to include additional ballots not casted by an eligible voter. Ballot stuffing by the voting system or the election authority itself must be prevented too.

Availability / Reliability. The system has to be accessible at any time during the voting phase for the eligible voters to cast their votes. Generally, availability cannot be guaranteed, since an adversary is able to mount a sophisticated distributed denial-of-service (DDoS) attack. However, a concise system architecture and infrastructure might reduce the risk of a complete system failure. Furthermore, the time window in which eligible voters are able to cast their vote can be set to a date before the election day. In a hybrid Internet voting system, an eligible voter who was unable to cast the electronic ballot properly, may use alternative ways to vote, e.g., by casting the ballot physically at the election office.

System security. The used hard- and software has to be proven secure and any possible measure has to be taken in order to prevent external attacks. In an Internet voting system, the voter's personal computer is part of the Internet voting system. Since it is not under control of the election authority, it will be difficult to enforce this requirement at the voter's side.

Verifiability-Related Requirements

Every step in the election shall be reproducible and the correctness of the whole election can be checked.

System test. Before the election starts, a system test must be performed and the correctness of every involved subsystem and process must be assessed.

Individual verifiability. The voter is given a proof that his vote has been recorded as intended. Measures must be defined and communicated for the case the voter's verification of the proof fails.

Universal verifiability. Every interested individual is provided with all information necessary to verify the correctness of the processing of the ballots and the tally. Usually this requirement is mentioned as preferable over auditability and indeed

it is a stronger requirement because of the fact that everyone interested in it must be provided with all information necessary to verify the correctness of all the involved processes. Acceptance is assumed to be higher. However, due to the lack of knowledge and skills, most voters will not be able to take this opportunity anyway, but have to trust the independent experts who can. Therefore, one could argue that it is reasonable for a voting system to be auditable.

Auditability. In the case where not every step is universally verifiable, eligible election officials or independent auditors shall be able to check the correctness of these steps. Whereas a verifiable system allows the individual voters and any third party to verify the correctness without gaining any information about the individual votes, auditability is a weaker requirement. Only a defined group of trusted experts performs the verification of the system's functioning. For example, auditors could be given access to a random set of casted ballots and check the processing of these ballots from the beginning to the end and thus get partial information of some voters choices.

Monitoring. The entire election process shall be monitored and logged. This allows one to analyze the election at a later time and to detect anomalies during the election. Part of this requirement is as well to define the measures to be taken in case of finding anomalies.

Archiving. All information necessary to reconstruct the voting process must be archived for later analysis of correctness, for forensics, and as a proof of correctness.

Reproducibility. The voting process has to be reproducible and provable but not leaking information about individual voters' choices, even after years.

Transparency. The system and every step shall be transparent, i.e., according to Kerckhoffs' principle the security of the whole system must only rely on the defined secrets, namely the used keys and not on closed source code.

Non-Security Requirements

This thesis focuses on security-related requirements but Internet voting systems are embedded into the entirety of all system requirements. Obviously, the above mentioned security requirements do not only partly contradict themselves, they even more contradict other common non-security-related system requirements. Some examples are the requirements of usability, scalability and cost-efficiency. It is a matter of tradeoffs, what requirements that are implemented and to what degree.

In the following we describe two important non-security requirements in more detail: *acceptance* and *scalability*.

Acceptance. The failures of many countries' advances in introducing nation-wide electronic voting systems for political elections have shown, that the socio-cultural aspects must be taken into account too. Even very sophisticated and provably secure

systems may fail because of a lack of trust of the electorate into it. For example, pilot trials in Austria have shown that it was sufficient to only claim that the system is insecure to undermine the electorate's confidence into the Internet voting system. Therefore, a non-technical but nevertheless important requirement of any voting system is that of acceptance. The electorate must know the system and the voters have to trust it.

Scalability. A practical political voting system must be scalable in different dimensions. First, it must be possible for the entire electorate to use the system to cast the votes. Second, it should be possible to not only cast simple ballots with the ability to choose "yes" or "no", but also very sophisticated ballots such as for national parliament elections. These elections may involve hundreds of candidates from which dozens can be chosen. In such a setting, the short ballot assumption [75] does not necessarily hold and therefore the security requirements may not be satisfied by the system.

2.3 Security Threats

There exist many different threats with respect to security-critical electronic communication systems and it is a matter of a thorough risk analysis to identify the assets and threats, and to determine the resulting risks for a specific system. In this section we identify the threat sources and the corresponding threats that we focus on in this thesis.

Threat Sources

Threat sources in nation-wide legally binding elections or referenda and their motivations are countless. Ranging from unexperienced "script-kiddies", requesting some form of fame, to organized crime and companies, trying to influence the legislation of a country for their own benefit. Moreover, foreign countries may try to destabilize or to control a country by influencing the democratic procedures and by undermining the citizens' trust into the authorities. Depending on the relevance of the system under consideration, it is reasonable to assume the most powerful adversary when analyzing the system's security properties. Internet voting for political elections and referenda is such a system. Hence, we assume the adversary to be more powerful than in other security-critical applications such as for e-commerce or e-government services.

Threats

A threat defines how a threat source may exploit a system's vulnerability to cause harm to an asset [7]. With respect to Internet voting, a number of threats have been identified and many of these threats are inherent to any security-critical electronic communication system. For example, the problem that the network may be controlled by

the adversary. Hence, solutions to analyze security protocols and to mitigate the corresponding threats already exist in other domains. Examples include formal symbolic analysis of cryptographic protocols and encryption schemes to encrypt confidential messages before sending them over an insecure network.

In the following we introduce three challenging problem areas that are related with Internet voting and the insecurity of the users and their computer systems. With respect to these problem areas we identify three main threats: *the adversary can break the secrecy of the ballot*, *he can secretly change the voter's ballot before, during or after casting the vote*, and he can manipulate the voter to vote in a specific way or not at all.

Breaking the secrecy of the ballot. Breaking ballot secrecy allows an adversary to coerce individual voters. There exist several points where an adversary could possibly break secrecy. First, the adversary could try to break the cryptographic scheme used to encrypt the ballot. For example regarding TLS (Transport Layer Security), recently, many vulnerabilities in OpenSSL where uncovered [58]. Second, the adversary could try to gain control over the election authority's server system. This problem and corresponding countermeasures are researched in the domain of system security and is out of scope of this thesis. Last, the adversary could try to gain control over the voter's computer system, i.e., the voting platform. We cover the consequences of this last problem area in Parts I and II, where we focus on confidential channels from the voter to the trusted election authority's server.

Secret change of the vote. This threat assumes either the election authority, its server infrastructure, the voter's personal computer, or any intermediary to influence the integrity of an honest voter's ballot. In contrast to coercion and vote buying, where the voter is aware of the deviation of his intended choice and the cast one, this threat concerns that even without the voter's knowledge the intended vote can be altered arbitrarily while the voter assumes that his vote has been processed correctly [9]. Parts of this threat scenario are as well dishonest election officials, ranging from a single dishonest election official to an entire district cheating. We cover this threat in Parts I and II, where we focus on authentic channels from the voter to the trusted election authority's server, as well as on individual verifiability.

Voter coercion / Vote buying. The fact that the voter performs vote casting remotely introduces additional threats such as voter coercion and vote buying. The first is the problem that the voter can be coerced by a (physically present) adversary. The latter is a special case of coercion, where the voter intends to collaborate with the adversary and to prove how he voted. Coercion-resistance, as we introduced above in this chapter, is currently a very active research area and a number of solutions have been proposed. In the context of this research, we improved an existing coercion-resistant Internet voting protocol [49] and we developed new approaches for efficient vote authorization [81] and coercion-resistant Internet voting in linear time [91, 92].

3 Preliminaries

To answer the research questions introduced in Chapter 1, we develop models and methods that are based on labeled term rewriting. We build our models upon Tamarin's security protocol model [84], which we will refer to as the *Tamarin model* throughout this thesis. The flexibility of Tamarin and its tool support helps us to constructively prove our theorems and propositions in later chapters. Although our extensions are substantial, the Tamarin tool, which performs deductions based on term rewriting, can still be directly applied to analyze our extended protocol models. In this chapter, we summarize the Tamarin model's main features and introduce the formal preliminaries for Parts I and II.

3.1 Notation

We denote the set of sequences of elements from a set S by S^*. For the sequence s, $|s|$ denotes the length of s. The set of indices of s is denoted by $idx(s) := \{1, \ldots, |s|\}$ and we write s_i to refer to the i-th element of s. A sequence s with $|s| = k$ is denoted by $[s_1, \ldots, s_k]$ and the empty sequence by $[\,]$. We denote the concatenation of two sequences s and s' with $s \cdot s'$. The superscript $\_^b$ (bag) is used to denote operations on multisets such as $\cup^b$ for multiset-union. S^b denotes the set of finite multisets with elements from S. $\mathcal{P}(S)$ denotes the powerset of S. For a sequence s, $mset(s)$ denotes the multiset of its elements and $set(s)$ the corresponding set. A set S is also a multiset and for a multiset M, $set(M)$ denotes the corresponding set.

3.2 Term Algebra

We use the term algebra of the Tamarin model. The term algebra is order-sorted with the sort *msg* and two incomparable subsorts *fresh* and *pub*. There are two countably infinite sets $\mathcal{C}_{\mathsf{fresh}}$ and $\mathcal{C}_{\mathsf{pub}}$ of fresh and public constants, respectively, and we denote their union by $\mathcal{C}$. Fresh constants are used to model the generation of nonces, while public terms are used to represent agent names and publicly known values. Let $S := \{fresh, pub, msg\}$. For each sort $s \in S$, there is a countably infinite set $\mathcal{V}_s$ of variables. We write $x{:}s$ to denote that $x \in \mathcal{V}_s$ and we let $\mathcal{V} := \bigcup_{s \in S} \mathcal{V}_s$.

A signature Σ is a set of function symbols, where each function symbol is associated with an arity. The subset of n-ary function symbols is denoted by Σ^n and we set $\Sigma^0 = \mathcal{C}_{\mathsf{fresh}} \cup \mathcal{C}_{\mathsf{pub}}$. Messages are elements of the term algebra $\mathcal{T} = T(\Sigma, \mathcal{V})$, and ground terms are elements of $\mathcal{M} = T(\Sigma, \emptyset)$.

In the Tamarin model, $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2$, where

$$\begin{aligned} \Sigma^1 &= \{\pi_1(\_), \pi_2(\_), \mathrm{h}(\_), \mathrm{pk}(\_)\} \\ \Sigma^2 &= \{\langle \_, \_ \rangle, \mathrm{senc}(\_, \_), \mathrm{sdec}(\_, \_), \mathrm{aenc}(\_, \_), \mathrm{adec}(\_, \_)\}. \end{aligned}$$

For $i > 0$, all functions in Σ^i are of sort $msg \times \cdots \times msg \to msg$. The function $\langle \_, \_ \rangle$ represents the pairing of terms, and π_1 and π_2 are the projections first and second, respectively. The functions $\mathrm{senc}(\_, \_)$ and $\mathrm{aenc}(\_, \_)$ represent symmetric and asymmetric encryption and $\mathrm{sdec}(\_, \_)$ and $\mathrm{adec}(\_, \_)$ represent symmetric and asymmetric decryption, respectively. $\mathrm{h}(\_)$ represents a hash function and $\mathrm{pk}(\_)$ corresponds to the public key for a given secret key. The function $\mathrm{pk}(\_)$ can be applied to any term t to yield the term $\mathrm{pk}(t)$, but t cannot be inferred from $\mathrm{pk}(t)$.

For $a, b \in \mathcal{T}$, we let $\mathcal{E}$ be the following set of equations over Σ:

$$\begin{aligned} \{ \quad & \pi_1(\langle a, b \rangle) = a, \pi_2(\langle a, b \rangle) = b, \\ & \mathrm{sdec}(\mathrm{senc}(a, b), b) = a, \mathrm{adec}(\mathrm{aenc}(a, \mathrm{pk}(b)), b) = a \ \}. \end{aligned}$$

The equational theory $Eq(\Sigma, \mathcal{E})$ is the smallest congruence containing all instances of the equations of $\mathcal{E}$ over Σ.

A position p is a (possibly empty) sequence of natural numbers. The subterm $t_{|p}$ of t at position p is inductively defined by t if p is empty and by $(t_i)_{|p'}$ if $p = i, p'$ and $t = f(t_1, \ldots, t_k)$ for $f \in \Sigma^k$ and $1 \leq i \leq k$. The set of all subterms of t is denoted by $St(t)$. The set of variables of t is denoted by $vars(t) := St(t) \cap \mathcal{V}$.

3.3 Multiset Term Rewriting System

We use a labeled multiset term rewriting system to represent all possible protocol behaviors. The system states are represented as finite multisets of *facts*. Facts are functions over $\mathcal{T}$ whose symbols appear in a signature Σ_{Fact} (disjoint from Σ), which is partitioned into *linear* and *persistent* fact symbols. Linear facts model resources that can only be consumed once, whereas persistent facts, prefixed by "!", model inexhaustible resources that can be consumed arbitrarily often. The set of facts $\mathsf{F}(t_1, \ldots, t_k)$, such that $\mathsf{F} \in \Sigma_{Fact}$ and $t_i \in \mathcal{T}$ for all $1 \leq i \leq k$, is denoted by $\mathcal{F}$ and the set of all ground facts, i.e., $\mathsf{F}(t_1, \ldots, t_k)$ such that $\mathsf{F} \in \Sigma_{Fact}$ and $t_i \in \mathcal{M}$ for all $1 \leq i \leq k$, is denoted by $\mathcal{G}$.

State transitions are effected by labeled multiset rewriting rules. Each such rule is denoted by $l -[\ a\ ]\!\!\to r$ with $l, a, r \in \mathcal{F}^*$. The elements in l, a, r are called the rule's premises, actions, and conclusions, respectively. The transition rewrites the current state by replacing the linear facts in l with the facts in r and is labeled with the facts

in a. The initial system state is the empty multiset. Formally, the labeled transition relation $\rightarrow_{\mathcal{R}} \subseteq \mathcal{G}^b \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^b$ for a set of multiset rewriting rules $\mathcal{R}$ is defined as:

$$\frac{l -[\ a\ ]\!\!\rightarrow r \ \in ginsts(\mathcal{R}) \quad lfacts(l) \subseteq^b S \quad pfacts(l) \subseteq set(S)}{S \xrightarrow{set(a)}_{\mathcal{R}} (S \setminus^b lfacts(l)) \cup^b mset(r)}, \tag{3.1}$$

where $lfacts(l)$ is the multiset of all linear facts in l, $pfacts(l)$ is the set of all persistent facts in l, and $ginsts(\mathcal{R})$ consists of all ground instances of rules in $\mathcal{R}$. Formally, $ginsts(\mathcal{R})$ is the set of all rules $l -[\ a\ ]\!\!\rightarrow r$ for which there exists a rule $l' -[\ a'\ ]\!\!\rightarrow r' \in \mathcal{R}$ with $|l'| = |l|$, $|a'| = |a|$, $|r'| = |r|$, and a substitution $\sigma : \mathcal{F} \rightarrow \mathcal{G}$ such that $\forall i \in \{1, \ldots, |l|\}, j \in \{1, \ldots, |a|\}, k \in \{1, \ldots, |r|\} : \sigma(l'_i) = l_i \wedge \sigma(a'_j) = a_j \wedge \sigma(r'_k) = r_k$. The transition rewrites the current state by replacing the facts in l with the facts in r and is labeled with the facts in a.

A trace tr is a finite sequence of sets of actions $tr_i \in \mathcal{P}(\mathcal{G})$, for $1 \leq i \leq |tr|$. The action sets in the trace label the system's state transitions that correspond to applying a ground instance of a rule in a set $\mathcal{R}$. We denote the set of all traces for the set of rules $\mathcal{R}$ by $TR(\mathcal{R})$ and refer to $\mathcal{R}$ as a *protocol*. Formally, the set of traces $TR(\mathcal{R})$ is defined as:

$$\begin{aligned} TR(\mathcal{R}) := \{ & [a_1, \ldots, a_n] \\ & | \exists S_1, \ldots, S_n \in \mathcal{G}^b. \emptyset^b \xrightarrow{a_1}_{\mathcal{R}} \cdots \xrightarrow{a_n}_{\mathcal{R}} S_n \\ & \wedge \forall i \neq j. \forall x. (S_{i+1} \setminus^b S_i) = \{\mathsf{Fr}(x)\} \Rightarrow \\ & (S_{j+1} \setminus^b S_j) \neq \{\mathsf{Fr}(x)\}\}. \end{aligned} \tag{3.2}$$

Note that Fr facts may only be generated by a distinguished model-specific rule (to be discussed below). Thus, the second conjunct ensures that each instance of the rule for generating Fr facts is used at most once in a trace and therefore each consumer of a Fr fact obtains a different fresh constant. Hence, a trace $tr \in TR(\mathcal{R})$ is a finite sequence of sets of actions $tr_i \in \mathcal{P}(\mathcal{G})$, $1 \leq i \leq |tr|$. We write $b \in tr$ if $b \in tr_i$ for some $1 \leq i \leq |tr|$, that is, when the action b occurs in a set of ground actions in the trace tr.

3.4 Trace Restrictions

The Tamarin model allows one to analyze trace properties with respect to a specific set of traces that hold predefined properties. For example, the following trace restriction represents the set of all traces where an agent is either human or computer but not both.

Trace restriction

$$\forall tr \in TR(\mathcal{R}), A, B \in \mathcal{C}_{\mathsf{pub}} : \mathsf{Human}(A) \in tr \wedge \mathsf{Computer}(B) \in tr \implies A \neq B. \qquad \square$$

3.5 Adversary model

In the Tamarin model, the network is controlled by a Dolev-Yao adversary [29]. The adversary chooses whether to deliver each message. He eavesdrops on, injects, and modifies messages on channels. However, he can neither eavesdrop on confidential (or secure) channels nor inject or modify messages on authentic (or secure) channels. The message deduction rules in $\mathcal{MD}$ represent his capability to receive, construct, and send messages in a protocol execution:

$$
\begin{aligned}
\mathcal{MD} \;:=\; & \{\; [\mathsf{Out}(x)] -[\; !\mathsf{K}(x) \;]\!\rightarrow [!\mathsf{K}(x)], && (3.3)\\
& [!\mathsf{K}(x)] -[\; !\mathsf{K}(x) \;]\!\rightarrow [\mathsf{In}(x)], && (3.4)\\
& [\,] -[\; !\mathsf{K}(x{:}\,pub) \;]\!\rightarrow [!\mathsf{K}(x{:}\,pub)], && (3.5)\\
& [\mathsf{Fr}(x)] -[\; !\mathsf{K}(x) \;]\!\rightarrow [!\mathsf{K}(x)] \;\} && (3.6)\\
\cup\; & \{\; [!\mathsf{K}(x_1), \ldots, !\mathsf{K}(x_k)] -[\; !\mathsf{K}(f(x_1, \ldots, x_k)) \;]\!\rightarrow && (3.7)\\
& \quad [!\mathsf{K}(f(x_1, \ldots, x_k))] \mid f \in \Sigma^k \wedge k > 0 \;\}.
\end{aligned}
$$

The !K fact appearing in all rules of $\mathcal{MD}$ is used to store and observe the adversary's knowledge in a trace and plays a role in specifying secrecy properties[1]. Rule (3.3) allows the adversary to learn all terms that are produced with Out facts and rule (3.4) allows the adversary to input any term in his knowledge into an In fact. The Rules (3.5) and (3.6) represent the adversary's capabilities to learn public and freshly generated constants, respectively. The set of Rules (3.7) allow the adversary to apply any function in Σ^k, for $k > 0$, to known messages.

Our formal models and the corresponding results are based on these preliminaries and we refer to the formal foundations in this chapter later in Parts I and II.

[1] For efficiency reasons, Tamarin distinguishes between !KU and !KD facts. For simplicity, we refer to both of these as !K facts.

Part I

Insecure Client Platform

4 Insecure Client Platforms

In electronic communication applications, users use their multi-purpose personal computer or other devices such as tablet computers or smartphones to communicate. For the rest of this thesis we refer to these platforms by using the term *personal computer* or simply *computer*. With the increasing complexity of the hardware and the corresponding operating systems and other software, a growing number of security flaws is reported. These vulnerabilities offer a broad attack surface for malware such as viruses and trojans. Malware infections are especially problematic in security-critical applications such as online banking and Internet voting. For example, malware can be constructed such that a voter would not even notice that the system voted for a different candidate than it claims to have [9]. Moreover, the voter cannot check the correct processing of his input into his personal computer nor of his computer's output. It is not enough to encourage users to keep their systems up-to-date and to use updated anti-virus software because most of the users are not computer nor security experts and cannot be assumed to handle security configuration instructions correctly. In other security-critical applications such as online banking, several solutions to mitigate this problem have been presented.

In this chapter, we first introduce the *Secure Platform Problem* and we characterize the corresponding adversary in Section 4.1. We then give a taxonomy of existing approaches to mitigate the problem in Section 4.2. Finally, in Section 4.3 we provide an overview of related work on formal modeling and analysis of security protocols in which humans are involved.

4.1 Secure Platform Problem Adversary

In electronic communication applications where humans use their personal computer to securely communicate with a remote communication partner, the *Secure Platform Problem* is the problem where unless the personal computer's hardware and software are trustworthy, information appearing on the computer's screen may not faithfully represent the messages communicated with the remote system. Moreover, the personal computer may leak information to unauthorized third parties [32, 72].

Securing the last few inches of the communication channel (those between the network cable and the human) is difficult, since in contrast to computing devices, most

people's computing and memorizing abilities are insufficient to perform cryptographic computations [13, 14].

In the following we informally characterize the adversary who is able to control the user's computer. We then explain such a powerful adversary's influence on Internet voting and the relevant requirements as introduced in Chapter 2.

Secure Platform Problem Setting

Figure 4.1 illustrates the setting in which a voter communicates with the election authority's server by using his personal computer. The adversary controls the network between the personal computer and the server but also the voter's personal computer. Communication takes place between the voter and the adversary as well as between the server and the adversary. Hence, all communication between the user and the server is sent through the adversary.

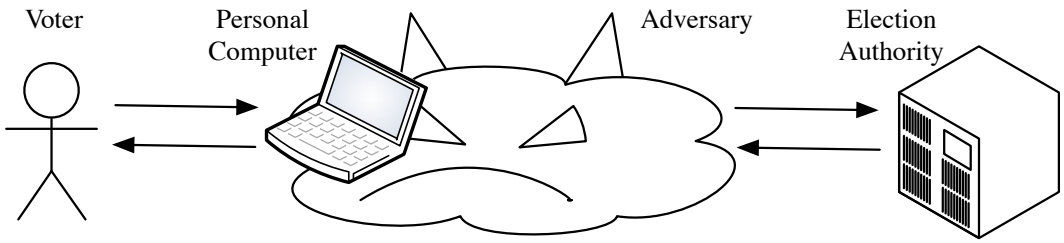


Figure 4.1: Secure Platform Problem in an Internet voting setting.

In this thesis we focus on the client-side problems and we do not consider a malicious election authority or voter. More specifically, we do not consider the case where the election authority or the voters intend to collaborate with the adversary as it would be the case for vote buying. Furthermore we do not take into account a physically present adversary. We examine erroneous user behavior later in Part II.

A protocol's primary security goal in this setting is to securely communicate a vote from the voter to the election authority's server. How this vote is processed on the election authority's side is out of scope of this thesis and builds its own research area.

The following list characterizes the Secure Platform Problem adversary's capabilities. In particular the adversary:

- can learn messages sent from the user to his personal computer;
- can learn messages sent from the server over the network to the personal computer;
- can drop messages and replace them with own messages;

- can manipulate and fabricate arbitrary messages according to publicly known knowledge and previously sent messages;
- can apply every publicly known function;
- knows all implementation details of all used systems;
- can act as a human.

Although the above characterization describes a powerful adversary, the capabilities are limited. In particular, the adversary cannot break cryptography.

Influence on Internet Voting Security

The Secure Platform Problem has an impact on a number of the requirements introduced in Section 2.2 above. In the following we identify a relevant set of these requirements and derive the underlying basic security properties a communication system must provide to fulfill the requirements.

Secrecy. To guarantee voter privacy, the voter's choice must remain secret. With respect to the Secure Platform Problem setting, a confidential channel from the voter to the election authority must be established. Since the adversary controls the voter's personal computer, the voter must not enter his choice directly into the computer.

Manipulation-protection. Since the adversary controls the voter's personal computer, he may influence and manipulate the voter in various ways. An authentic channel from the voter to the election authority is a sufficient condition to protect against manipulation.

No intermediary results. To prevent the adversary from learning intermediary results, the votes must be confidentially communicated. Thus, a confidential channel from the voter to the election authority is a sufficient condition to fulfill this requirement.

Ballot casting assurance. Ballot casting assurance implies the assurance that the vote was cast as intended on the voter's side and recorded as cast on the election authority's side. The Secure Platform Problem adversary influences the first part, i.e., the cast as intended property, but not the recorded as cast property, which is out of scope of this thesis. With respect to the Secure Platform Adversary, an authentic channel from the voter to the election authority is a sufficient condition. We discuss the voter's confidence into this fact next.

Individual verifiability. Whereas authenticity ensures a message's origin for the recipient, individual verifiability ensures authenticity to the sender of the message. This ensures to the sender, that the recipient indeed received the message.

Universal verifiability. One foundation for universal verifiability is that the verification information is authentic and therefore, an authentic channel from the election authority to the voter must exist.

Summarizing the above underlying security properties, an Internet voting system needs to provide confidential and/or authentic channels directly between the voter and the election authority to fulfill the relevant security requirements. To guarantee to the voter that his ballot was correctly submitted and not blocked by the adversary, individual verifiability may be defined as an additional channel property.

We will formally define confidentiality and authenticity as channel goals in our Secure Platform Problem security protocol model in Chapter 5 and individual verifiability in Chapter 7, where we analyze a recently proposed Internet voting system.

4.2 Taxonomy of Existing Mitigation Approaches

The Secure Platform Problem is addressed by a variety of supporting technologies, ranging from simple code sheets [21] to hand-held smart card readers with integrated keypads and displays, commonly used for online banking, such as described in [48].

Following, we provide a taxonomy to give an overview and to classify the different mitigation approaches for the Secure Platform Problem. The classification serves as a survey of current state-of-the-art solution approaches for the problem and we use it later on to define the scope of our models. We identify two main classes of approaches. While approaches in the first class aim to make the platform trustworthy, approaches in the second class accept the user's personal computer to be insecure. See Figure 4.2 for an overview of the classes that are covered in this section.

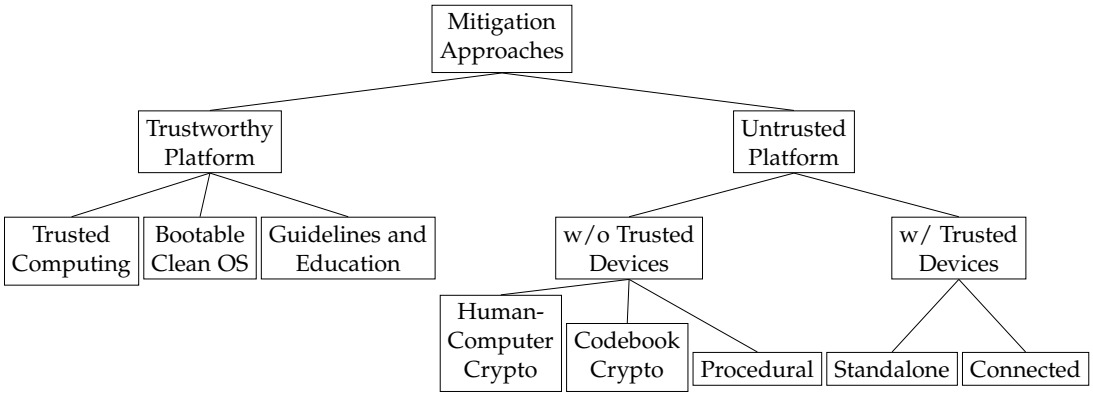


Figure 4.2: Taxonomy of mitigation approaches.

Below we refer to this taxonomy, summarize specific examples, and analyze them informally with respect to the Secure Platform Problem adversary we characterized above in Section 4.1. In particular, in Section 4.2 we first examine approaches to make

the platform trustworthy. We then analyze approaches that distrust the user's platform, where we summarize approaches without trusted devices and approaches based on dedicated trusted devices.

Trustworthy Platform

Approaches in this category aim to improve the personal computer's integrity to make it trustworthy. These approaches can be classified into the following classes: *trusted computing*, *bootable clean operating systems*, and *guidelines and education*.

Trusted Computing

The key idea of applying Trusted Computing [93] techniques to address the Secure Platform Problem is to use an appropriate security architecture based on a security kernel and special Trusted Computing hardware. [2] and [98] examine the application of Trusted Computing techniques for internet voting in detail. Eligible messages can only be created after successful verification of the computer's integrity. Hence, Trusted Computing overcomes the problem of malicious software running on the voter's personal computer. However, there are still open problems with Trusted Computing [97] and corresponding hardware is not yet implemented on a large scale. Some voters may already have a personal computer with an integrated Trusted Platform Module but the security architecture and the security kernel are still missing. It is questionable whether or not this technique will be available on a large scale in the near future.

Bootable Clean Operating System

Otten [69] proposes a specific voting operating system, based on an open source operating system, that boots and runs directly from a read-only data medium such as a CD or DVD. This medium would then be distributed to the electorate. The voter needs to configure his computer to boot from this medium. Additional security checks and secure distribution of the media are required to prevent an adversary from distributing malicious media that, for example, communicates with a malicious server. While this approach mitigates many Secure Platform Problem related issues, it does not protect from a manipulated BIOS. The adversary may load a malicious environment in which the secure operating system is executed, without the voter or the authority noticing it. To avoid this kind of attack, Trusted Computing hardware must be applied as described before. Challenges include the development of a CD or DVD that boots on all the different hardware and software settings around and that all necessary drivers are included in the provided system. This may lead to high development, distribution, and maintenance costs for the election authority. Another difficulty for the authority is to verify that users really use the clean operating system and that it was running on hardware, and not in a virtual machine, which potentially is under the adversary's control.

Guidelines and Education

A simple approach to mitigate the Secure Platform Problem is the provision of special guidelines and the education of the electorate in how they may protect their personal computers from malware infections. Examples include the guidelines provided for the student elections in Austria [94] and those developed by the German society of computer scientists [38]. They include information about software updates, firewall-settings, and the proper verification of SSL certificates. This approach aims to reduce malware infections. However, only standard and well-known attacks can be prevented and it is questionable whether a large fraction of the electorate is able to follow the guidelines to protect their computers. Moreover, voters cannot be forced to apply the proposed security guidelines. Note that in contrast to online banking, where laziness of individual customers affects only the corresponding customer, in Internet voting, such a behavior has an effect on the outcome of an election and therefore on the entire electorate.

Without Trusted Devices

Distrusting the voter's computing platform leads to the need for establishing a secure channel directly between the user and the server. This channel can either be established using cryptography or by an "out-of-band" channel that is not under the adversary's control. The mitigation approaches in this category can be further classified into: *human-computer cryptography*, *codebook cryptography*, and *procedural* approaches.

Human-Computer Cryptography

Human-computer or paper-and-pencil cryptography has a long history and given enough time, pencils, and paper, in theory humans may perform every computation a computer can do. Such approaches aim to provide humans with the capabilities to encrypt and authenticate messages without any supporting technology. However, the vast majority of humans lack sufficient memory, computation power, and knowledge to perform strong cryptographic operations in a reasonable amount of time. Nevertheless, some interesting approaches such as [54] and Schneier's Solitaire algorithm [86] were proposed. For example, in Solitaire, the randomness of a shuffled deck of playing cards is used to encrypt messages. Bertà examined in [14] the human limitations with respect to the encryption and authentication of sufficiently large messages in practical settings. He concludes that for humans no sufficiently strong cryptographic protocol to encrypt or authenticate messages exists. Because of the obvious lack of practicality we do not consider human-computer cryptography as a solution for any practical communication application.

Codebook Cryptography

In codebook cryptography, a codebook is exchanged prior to the communication. This codebook assigns random codes to clear-text messages. The sender chooses a code

that is assigned to the clear-text message he wants to communicate and sends the code instead of the clear-text. Hence, an adversary overhearing the communication does not learn the clear-text and, without the codebook, he cannot replace the code with another valid one that will be accepted by the receiver. In the following we present some voting protocols in this category.

Code voting. Chaum was the first who applied this concept to internet voting [21]. The key idea of Chaum's SureVote is that the voter gets a code sheet together with the general election information via a secure out-of-band channel, such as postal mail. The code sheet links each candidate or party to random alphanumeric codes. In order to cast a vote for a specific candidate, the voter enters the corresponding code into his personal computer. The code is submitted to the election authority that is able to derive the user's choice by mapping the received code back to the corresponding candidate and confirms the reception with a corresponding verification code. Since not in possession of the code sheet, the adversary can neither derive the user's choice from the submitted codes nor can he change the ballot's content to another meaningful code. Helbach and Schwenk [45] as well as Oppliger et al. [67] proposed different improvements like additional finalization codes that are sent back to the server to improve individual verifiability. To overcome the vote-buying problem the authors of [68] propose an additional finalization code. Ryan et al. suggest in [79] and [44] to use code voting in combination with Prêt-à-Voter in order to allow individual verifiability. In this approach, the election authority's server can only send a valid confirmation code to the user if a majority of trustees confirms the correct processing of the voting code. The main disadvantage of code voting concerns user-friendliness, which decreases with the complexity of the ballots, i.e., with the number of candidates to choose. In addition, code voting requires a trusted procedure to generate and distribute the code sheets.

Prêt-à-voter. Another implementation of codebook cryptography can be found in Prêt-à-Voter [78], where the candidate list is permuted and the permutation is encrypted. While Prêt-à-Voter was proposed for a poll-site-based setting, it could be adapted to Internet voting too. The voter receives the ballot paper with the permuted list of candidates and the encryption of the permutation via mail. Then, he enters the index of the chosen candidate together with the encryption of the permutation (which could also serve as a ballot sequence number). The permutation is decrypted by the election authority and is used to derive the voter's choice. However, since using a permuted list of valid choices, the adversary, although not able to learn the voter's choice, can mount a randomization attack by choosing a different index.

CAPTCHAs. Oppliger et al. [68] propose the application of CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) to mitigate the Secure Platform Problem. Candidates are displayed in a random order and represented by visual CAPTCHAs. While people may know this kind of images from other appli-

cations, it is often difficult to figure out what is displayed. Therefore, user-friendliness is questionable. Similarly to Prêt-à-Voter, the adversary can perform a randomization attack by randomly choosing a CAPTCHA different from the one that was chosen by the voter. Moreover, we assume that the adversary has human capabilities too. Hence, he can solve the CAPTCHAs. A more advanced approach is discussed in [73]. In this approach, arbitrary CAPTCHAs, which are not related to the candidates' names, appear next to the actual candidate name, while the candidate order can stay as it is. The voter decodes the CAPTCHA related to his favorite candidate and enters the resulting code into a text-field. This approach prevents the adversary from a simple randomization attack but since we assume that the adversary has human capabilities, he is again able to solve the CAPTCHAs and thus, to manipulate the ballot.

Procedural Approaches

Approaches in this class adapt the voting process to mitigate various problems in Internet voting. Some of these approaches can be applied to mitigate the Secure Platform Problem.

Vote-updating. The general idea of vote-updating is that the voter can update his electronic vote as often as he wants to, ideally from different devices. In hybrid systems, the voter can even update the electronic vote by a traditional paper ballot at the polling station. There are several different approaches to enable vote-updating, while they have different advantages and disadvantages as shown in [40]. Enabling vote-updating, the adversary looses confidence in how a voter really voted. This is due to the fact that he may use another device which is not under the adversary's control or that the voter went to the polling station on election day to cast a paper ballot. The adversary does not know whether his modifications will influence the result or not. Vote-updating is particularly useful in the case where a voter distrusts his personal computer after casting his vote. For instance, this is the case when he misinterpret information presented to them, detect suspicious behavior caused by malware, or if he is not convinced that the ballot was properly transmitted. However, statistics from Estonia, where Internet voting is used for political elections since 2007, indicate that only few people update their vote [31]. Maybe very few incidents happen but more likely most voters are simply not able to notice manipulations and therefore blindly trust their personal computers. Vote-updating is easy to implement. The challenge is to ensure that only one vote is counted and replay-attacks or delays on the network can cause that an earlier or later cast valid vote is counted instead of the intended one. Opponents of vote-updating also argue that this approach influences the value and character of an election. They argue that the act of casting a vote is something special and should not be repeatable because otherwise it gets the character of a game.

Anonymous voting. Anonymous voting allows the voter to cast a ballot without the adversary being able to link the ballot to its origin. The following protocol serves as

a simple example for anonymous voting. Prior to the election the voter receives an official letter from the election authority. This letter contains credentials that allow the voter to cast exactly one eligible ballot and the authority cannot link the credential to the voter. Now the voter uses a public computer, for example in an Internet cafe, that cannot be linked to him. Then he enters the credentials to prove eligibility, chooses his favorite candidate, and submits the vote together with the credentials to the election authority. The adversary learns the candidate whom was voted for but does not learn who voted. Hence, voter privacy holds. It is obvious that in this simple example neither confidentiality nor authenticity holds, unless combined with other approaches. A more advanced example for this category is based on blind signature schemes that have been first proposed in [35] and was later used in many other voting protocol proposals. While this approach is more sophisticated from a cryptographic point of view, it does address the Secure Platform Problem to the same extent as the simple protocol above. Under the assumption that the personal computer does not know the identity of the voter who is using it, voter-anonymity is given but neither confidentiality nor authenticity holds.

Test ballots. Test ballots are dedicated ballots that the adversary cannot distinguish from real ones. The voter casts one real ballot and some test ballots in random order. Since the adversary does not know the real ballot he may manipulate one or more test ballots. After the vote casting phase, the processing of all test ballots is made fully transparent to a group of auditors or even to the public for verification. For successful attacks, the adversary must manipulate a large number of votes and therefore it is likely to detect at least a fraction of the manipulations. The approach does not provide authenticity of the real vote itself but strongly indicates possible manipulations by analyzing the proper handling of the test ballots. To detect manipulations in the context of the Secure Platform Problem, the test ballots could be individually verifiable by the voters. Hence, the test ballots are exposed after the election's vote casting phase and the adversary learns the real ballot.

Trusted Devices

In the context of online banking, a variety of technical solutions for user and transaction authentication have been presented and even deployed on a large scale. In the following we summarize these approaches and we give some hints on how such approaches could be adapted to internet voting.

Solutions based on trusted devices can be classified into *standalone* approaches, where the trusted device is not attached to the user's personal computer, and *connected* approaches, where the trusted device is connected to the adversary, for example, to the user's personal computer. The term trusted indicates that users must trust their devices to offer trustworthy functionality and that they securely store confidential data. If the device additionally stores a specific user's secret information, such as cryptographic keys, we call the device a personal trusted device. An example of personal

trusted devices are smart cards. They provide certain cryptographic operations and store individual secret information, i.e., the user's secret key. Since additional devices are required to access the smart cards, users need to trust these devices too. Hence, in smart-card-based approaches the card readers are also a part of the trusted device.

Standalone Trusted Devices

The user communicates directly with the device and no communication takes place between the trusted device and any other party than the user. Challenge-response authentication protocols for online banking solutions that use a smart card reader with a key pad and a small display are an example for this category of approaches. The bank first sends a challenge code to the user's personal computer. The user enters this code into the card reader together with the bankcard's personal identification number (PIN) to access the card. The card then computes a message authentication code (MAC) of the challenge code. This MAC is displayed to the user through the reader's display. Finally, the user enters the MAC into his personal computer to submit it to the bank. If the received code corresponds to a valid MAC, the bank is confident that the correct card (containing the secret information) as well as a person who knows the correct PIN to access the card are involved in the protocol run. To adapt this approach to internet voting, such a device could be used to encrypt and authenticate the ballot. The voter enters the candidate choice directly into the personal trusted device and in turn receives an encrypted ballot, for example, represented as an alphanumeric code. The voter enters the encrypted ballot into his personal computer to submit it to the election authority. Such an approach can be used to provide "digital" code voting without a dedicated out-of-band channel. The initial vote codes are digitally provided to the users via their personal computers. The vote codes are then used like the challenges in the above described challenge-response authentication example. The voter enters the MAC of the vote code that corresponds to his choice into his personal computer to submit the ballot to the election authority. The authority in possession of the same secret key computes the MACs for all candidate codes and compares them with the received MAC to derive the user's choice. Furthermore, the election authority could again compute the MAC of the received MAC and send it back to the user as a confirmation code that can be verified by the user by again computing the corresponding MAC using his personal trusted device. In practice a more sophisticated scheme is required to prevent the authority from learning individual voters' votes. Furthermore, randomization attacks must be prevented. User-friendliness is questionable since the voter must relay all encrypted communication between the personal trusted device and the election authority. In the case of a few short codes, this may be applicable but not necessarily in complex elections where the voter chooses multiple candidates.

Connected Trusted Devices

In contrast to standalone trusted devices, connected trusted devices are attached to the adversary, either to the user's personal computer or directly to the network, thus there

is a communication channel between the adversary and the trusted device. Simple approaches offer specific functionalities to the computer they are attached to, such as securely storing secret information. More sophisticated networked devices are able to directly communicate with a remote server. Generally, the more functionalities a trusted device offers, the more vulnerable it is. The availability of communication interfaces of a connected trusted device affects the security guarantees of any protocol using it. To make this clear, we summarize all possible communication capabilities of the trusted device, provide examples, and describe the properties of each setting.

No communication between user and trusted device. The trusted device has no interface to communicate with the user and all communication is based on the user's personal computer (which is controlled by the adversary). Particularly the user can only communicate with the adversary and neither confidentiality nor authenticity can be achieved with respect to our the SPP adversary. The only guarantee in this setting is that during the protocol run, the trusted device is attached to a personal computer. Widely deployed examples for this setting are smart cards in combination with class 1 card readers directly attached to the personal computer, for example via USB. The adversary instantly learns every message sent by the user, particularly the PIN that protects access to the card. Hence, after the first time the user enters the PIN, the adversary may use the card and perform whatever functionality the smart card offers, as long as it is attached to an adversary-controlled personal computer. Applying this to Internet voting the voter does not know what the trusted device processes. In a setting where additionally vote updating is allowed, the voter does not even notice if the trusted device is later used to replace his original vote. Another example are hardware tokens like that of mIDentity, developed by Kobil [59]. It is a commercial solution for online banking and other sensitive applications. A browser runs in a "secure" environment on the trusted device. As there is no communication between the trusted device and the user, neither confidentiality nor authenticity is guaranteed. We will prove this later in Chapter 6.

Unidirectional communication from user to trusted device. The TD provides an input interface to the user such as a pin pad and the user can send clear-text messages to the TD, which are encrypted afterwards. Hence, such approaches provide user-to-server-secrecy as well as user-to-server-integrity. Depending on the protocol, the user can even verify the correct reception of the message (individual verifiability). To do so, the user enters her choice together with a random number into the TD, which in turn encrypts these messages with a secret key as well as a hash value thereof. The secret key is shared between the server and the TD. The TD sends the encrypted values to the adversary for submitting them to the server. After reception, the server first decrypts them and verifies their integrity before sending the random number back to the adversary as confirmation. The adversary provides this confirmation to the user and the user is convinced that the server received the correct message if and only if the confirmation corresponds to the previously entered random number. Since the

adversary cannot break cryptography, the best strategy would be to guess the random number, which is not feasible for large enough numbers. In this setting the server cannot send any secret or authentic message to the user because the TD is not able to communicate with the user differently than through the adversary. Examples include smart cards in combination with class 2 readers. These readers provide a pin pad to prevent the platform from learning the PIN since it is directly entered into the trusted card reader. Because the adversary does not learn the PIN, unnoticed access to the smart card's functionality is prevented.

Unidirectional communication from trusted device to user. The trusted device provides an output interface such as a display or a speaker. The adversary learns every message sent by the user. Therefore, the adversary may again learn the PIN to access the trusted device's functionality. It is possible to avoid this, for example, by displaying a randomly permuted key pad on the untrusted personal computer's display on which the user has to enter the PIN [95]. The corresponding permutation is displayed on the trusted device's display. Every time the trusted device is accessed it challenges the user, with a new permutation. Moreover, the trusted device's output interface can be used to verify that the remote server received the correct message. To do so, the server sends an encrypted confirmation message, that the trusted device decrypts and displays to the user. Hence, depending on the implementation, confidentiality and authenticity from the user to the server as well as vice versa can be achieved. Thus, such solutions successfully mitigate the Secure Platform Problem but they require sophisticated user interfaces to assure confidentiality. Therefore, these approaches lack user-friendliness for practical applications. Examples are smart cards and readers with only a confirmation display. The display can be used to verify the data that will be processed by the smart card. Another example. Which is already widely applied in online banking is transaction authentication. Confirmation information for transactions entered by the user is sent over a dedicated channel to the user. The user verifies the correctness and authorizes the transaction. One concrete implementation is based on short messages and mobile phones as trusted devices. The user enters the desired transaction instruction into his personal computer and sends it to the bank. The bank generates a transaction authentication number (TAN) and sends it as a short message together with the beneficiary's account number and the amount to the user's mobile phone. The user verifies the correctness of the transaction information. If this verification is successful, the user authorizes the transaction by entering the TAN into his personal computer which submits it to the bank. The bank compares the received TAN with the expected one and accepts the transaction if they match. This concept could be used for Internet voting to allow individual verification but it is only secure as long as the adversary cannot break confidentiality and authenticity of the short messages sent by the election authority's server. Unfortunately today's implementations cannot prevent either. Moreover, it is questionable whether it is reasonable to assume that mobile phones are trustworthy [32]. Moreover, in near future communication platforms may conflate and voting as well as short messaging may be performed using one and

the same device.

Bidirectional communication between trusted device and user. The trusted device provides an input as well as an output interface to communicate with the user. In this setting the user may enter confidential messages directly into the trusted device. In contrast to the setting where communication exclusively takes place from the trusted device to the user, user-friendliness is improved due to the fact that security-critical messages can directly be entered. These approaches also allow individual verifiability based on the trusted device's output interface to the user. Examples are smart cards in combination with class 3 readers. These devices offer a high degree of security and a convenient way for the user to securely communicate with the remote server. In more complex solutions the trusted device communicates directly with the remote server by using standard network communication protocols. For example, the *Zone Trusted Information Channel* (ZTIC) [4, 101] was developed by IBM and is widely deployed by a popular Swiss bank. The ZTIC is a networked USB-attached smart card reader for entity and transaction authentication in online banking. The device consists of a small display and a few buttons. Transactions with yet unknown beneficiaries must be verified by the user and authorized by pressing the corresponding button on the device. The device communicates directly with the bank's server using the personal computer as a network proxy. Adapting such a solution for Internet voting would offer a high degree of security while being convenient with respect to user-friendliness. Borchert et al. propose the use of camera-equipped mobile phones for user and transaction authentication in online banking. They provide a variety of ideas and prototype implementations [95].

4.3 Modeling the Secure Platform Problem

In classical formal symbolic protocol analysis, the assumed adversary controls the communication network and a security protocol's goal is to establish a confidential or authentic channel, or both, between the computers in such a hostile environment. To do so, the computers perform cryptographic operations. The adversary is commonly modeled as the Dolev-Yao intruder we introduced above in Section 3.5. He cannot break cryptography but is a legitimate protocol participant and controls the entire network between all the computers. This adversary's capabilities cover the Secure Platform Problem adversary's capabilities only partly. Under the realistic assumption that an adversary is able to gain part or full control over the voter's personal computer used for selecting and casting the vote, the adversary has to be modeled more powerful, i.e., not only having control over the network but as well over the personal computers of the voters.

It is therefore natural to ask how one might model a system where humans, computers, and supporting technologies communicate, and analyze its security properties. As described in our taxonomy above in Section 4.2, most existing work on such systems and protocols focuses on particular scenarios, for instance on browser-based se-

curity protocols [36, 43], login procedures [47], solutions for online banking [101, 100] or Internet voting [66, 82]. A general approach to modeling and reasoning about such systems are security ceremonies [30]. These extend communication protocols to include actors and communication means that belong to the context in which security protocols are executed, but whose properties and behaviors have traditionally not been considered in conventional security protocol models. For instance, a human and his computer are actors and the visual channel between the human and the computer screen is a communication channel in a ceremony. Due to the open-ended, informal characterization "there is nothing out-of-band to a ceremony" [30], it is difficult to decide what such a model should include and a comprehensive, yet useful formal model of security ceremonies appears to be difficult to attain. This challenge has lead to a variety of models with different focal points, such as a logic to reason about objects, their location and movement [57] or a division of ceremonies into layers [10, 11].

In Section 4.4 we discuss these and other approaches and relate them to our model, that we introduce below in Chapter 5.

4.4 Related Work on Modeling End-to-End Security

Formal symbolic methods help to understand and to systematically detect logical flaws in security protocols. In contrast to cryptographic methods, messages are represented as terms instead of bit-strings. This abstraction does not focus on cryptographic primitives but assumes them to be perfect (perfect cryptography assumption).

Some progress towards a generic formal model in contexts where the adversary adaptively gains access to a protocol agent's secrets has been shown by Basin and Cremers [5, 8].

Security Ceremonies

Security ceremonies were informally introduced by Ellison [30, 96] as a generalization of security protocols. According to Ellison, ceremonies distinguish themselves from security protocols in the following ways. First, nothing is out-of-band to a ceremony. Second, ceremonies include a greater variety of "network connections" such as human-human and human-machine channels. Third, human nodes have different capabilities from computer nodes. In this part, we consider a setting that is more abstract and more precise than Ellison's description of security ceremonies. We represent human agents in our model by nodes that have restricted capabilities. Human-human and human-machine channels are simply links between nodes. In contrast to ceremonies, we do not impose a nothing-is-out-of-band requirement.

Bella and Coles-Kemp interpret the "nothing-is-out-of-band" requirement more broadly than Ellison. They extend security ceremonies with technical and social elements such, as a human agent's belief system and cultural values [10, 11]. They propose modeling security ceremonies using five layers: (1) the security of the protocol executed by the computers of the communicating partners; (2) the inter-process

communication of the operating system; (3) the socio-technical protocol whereby a user interacts with a graphical user interface; (4) the user's state of mind and (5) the influence of society on individuals. In [11], they formalize layer three, which is responsible for human-computer interaction, and they give a case study. The case study demonstrates the verification of a user's confidence in the privacy assurance offered by a service provider in an example ceremony. In contrast to Bella and Coles-Kemp's model, we do not consider societal and cultural influences on agents. We focus instead on the information exchanged between human and non-human agents in a ceremony as well as their computational abilities and channel properties.

Carlos et al. sketch a method to formalize human knowledge distribution in security ceremonies [19]. In subsequent work [20], they aim for a realistic verification of a Bluetooth pairing ceremony by restricting the capabilities of a standard Dolev-Yao adversary. Their results are, however, specific to Bluetooth pairing ceremonies. In contrast, we work in the standard Dolev-Yao adversary model and extend the communication model with channels that guarantee security properties such as authenticity and confidentiality.

Actor Networks

Meadows and Pavlovic propose a logic of networks involving humans, devices, and computers. They analyze various authentication protocols [70] with respect to claimed security guarantees, but they do not provide a formal adversary model. Their formalism is comprehensive, but complex. In subsequent work, they extend their logic to a "logic of moves" and use it to analyze physical airport security procedures [57]. Similarly to Meadows and Pavlovic, we provide a graphical model for the communication topologies of security ceremonies. However, our abstraction is simpler while supporting the modeling of the communication topologies of security ceremonies in arbitrary detail. The level of abstraction we use is both intuitive to understand and straightforward to verify with existing protocol verification tools. Moreover, we provide a comprehensive formal adversary model for the verification of security properties of protocols involving humans, devices, and computers.

Insecure Terminals

Various other approaches to modeling systems consisting of humans, insecure platforms, servers, and supporting technologies have been investigated independently of the work on security ceremonies. In particular, the problem of ensuring that the user's computing platform faithfully executes a security protocol and does not leak confidential information to any unintended third party is also known as the *problem of untrusted terminals* [13]. In the context of Internet voting, [65, 66, 82] provide overviews of different approaches to establishing a secure channel between a human, using a dishonest platform, and a remote server. In online banking, several concrete solutions have been presented, such as [100, 101]. However, to our knowledge, there exists no formal modeling and verification approach for this problem. Our model allows one to rea-

son about the security properties and the required assumptions of such solutions in general.

Trusted Paths

Another related research area concerns *trusted paths* [33, 74, 103], which is the problem of providing secure channels from an input device to a trusted application and onward to an output device. Research on trusted paths focuses on implementation details at the system level. However, a secure channel from the *human* to a remote entity can also be seen as a trusted path. Our topology model can be used to characterize the low-level communication channels and computationally restricted system parts in the context of trusted paths. However, this has not been the focus of this project.

Channel Abstraction

Regarding our formalization of insecure, authentic, and confidential channels, Mödersheim and Viganò provide a security protocol model [61] based on abstract channels as assumptions and goals. Their *ideal channel model* is closely related to our channel rules in that it provides an abstract notation for sending messages via authentic and confidential channels. Whereas Mödersheim and Viganò implement their abstract channels using asymmetric cryptography, our channel rules directly specify the adversary's interaction with the abstract channels in our model.

5 Secure Platform Problem Model

In this chapter we introduce our formal model to examine communication systems claiming to solve the Secure Platform Problem as introduced in Chapter 4. Our model is based on the Tamarin model that we summarized in Chapter 3 above and constitutes the formal underpinning of our communication topology model that we will introduce in Chapter 6. We summarize our model's main features and several extensions, such as the notion of communicating knowledge. Note that although our extensions are substantial, the Tamarin tool, which performs deductions based on term rewriting, can still be directly applied to analyze our extended protocol models.

5.1 Extended Security Protocol Model

Next, we describe our security protocol model. We extend the Tamarin model with the following facts and rules.

Model Facts and Rules

We define a fixed set of fact symbols and rules to model security protocols. The following equations summarize all facts used in the model.

$$\begin{aligned}
\Sigma_{Fact} :=& \Sigma^1_{Fact} \cup \Sigma^2_{Fact} \cup \Sigma^3_{Fact}, \text{ where} \\
\Sigma^1_{Fact} :=& \{\mathsf{Fr}, \mathsf{Out}, \mathsf{In}, \mathsf{!K}, \mathsf{Agent}, \mathsf{Honest}, \mathsf{Dishonest}, \mathsf{Trust}\}, \\
\Sigma^2_{Fact} :=& \{\mathsf{!Auth}, \mathsf{!Conf}, \mathsf{Fresh}, \mathsf{Comm}, \mathsf{Learn}\}, \\
\Sigma^3_{Fact} :=& \{\mathsf{Snd_I}, \mathsf{Rcv_I}, \mathsf{Snd_A}, \mathsf{Rcv_A}, \mathsf{Snd_C}, \mathsf{Rcv_C}, \mathsf{Snd_S}, \mathsf{Rcv_S}\} \\
& \cup \{\mathsf{!Sec}, \mathsf{Secret}, \mathsf{Authentic}, \mathsf{Verify}, \mathsf{AgentState}\}.
\end{aligned}$$

The set of all facts $\mathcal{F}$ is therefore

$$\mathcal{F} := \left\{ f(t_1, \ldots, t_k) \mid f \in \Sigma^k_{Facts} \wedge t_1, \ldots, t_k \in \mathcal{T} \right\}.$$

We use Agent, Honest, Dishonest actions to indicate agents in a trace. Honest agents are indicated with Honest, dishonest agents with Dishonest. Once an agent is indicated to be honest it cannot become dishonest or vice-versa. This is enforced in Tamarin with an axiom. Trust is used to indicate agents which are assumed to be honest for the

purpose of security properties, see Definition 5.7 below. These are agents whose roles are marked honest in the communication topology.

We distinguish between model and protocol specification rules, denoted by $\mathcal{R}_{Model}$ and $\mathcal{R}_{Spec}$ respectively, where $\mathcal{R}_{Model} \cap \mathcal{R}_{Spec} = \emptyset$ and $\mathcal{R} = \mathcal{R}_{Model} \cup \mathcal{R}_{Spec}$.

The former are a fixed set of rules introduced in the following and the latter specify the security protocol. There are four types of model rules: Rules for generating fresh constants $\mathcal{FR}$, message deduction rules $\mathcal{MD}$ modeling a standard Dolev-Yao adversary [29], channel rules $\mathcal{CH}$ modeling channel properties as assumptions, and dishonest agent rules $\mathcal{DA}$ modeling the behavior of dishonest agents. The message deduction rules and the fresh constant generation rule are adapted from [84], while the remaining rules are specific to our model. Thus, our set of model rules $\mathcal{R}_{Model}$ is defined as

$$\mathcal{R}_{Model} := \mathcal{FR} \cup \mathcal{MD} \cup \mathcal{CH} \cup \mathcal{DA}.$$

The only rule producing fresh constants and thereby creating Fr facts is Rule (5.1). Recall that due to Equation (3.2), every fresh constant is produced at most once in a trace. Fresh constants can be obtained (generated) by honest agents using Rule (5.2). The adversary can generate fresh constants for dishonest agents using Rule (3.6).

$$\mathcal{FR} := \{\ [\,]\text{–}[\ ]\rightarrow[\mathsf{Fr}(x : \mathit{fresh})]\ , \tag{5.1}$$

$$[\mathsf{Fr}(x)]\text{–}[\ \mathsf{Fresh}(A, x), \mathsf{Honest}(A)\ ]\rightarrow[\mathsf{Fresh}(A, x)]\ \} \tag{5.2}$$

Dishonest Agents

Tamarin provides two facts that model the Dolev-Yao adversary's ability to receive and send messages. The adversary learns all terms in Out facts and injects messages from his knowledge using In facts. We distinguish between the adversary's ability to control communication channels and his ability to control dishonest agents, such as a malware-infected personal computer. We model agents explicitly with $\mathsf{AgentState}(A, c, n)$ facts, where A is a public term representing an agent's name, c refers to the role step the agent is in, and n is the agent's knowledge at that step. The set of agents appearing in a trace tr, denoted by $\mathit{Agents}(tr)$, is the set of all public constants A such that $\mathsf{Agent}(A)$, $\mathsf{Honest}(A)$, or $\mathsf{Dishonest}(A)$ appears in tr. The subset of honest agents, denoted by $\mathit{Honest}(tr)$, is the set of all agents A such that $\mathsf{Dishonest}(A)$ does not appear in tr.

We model dishonest agents with the $\mathcal{DA}$ rules shown in Figure 5.1. These agents are marked with a Dishonest action. Rule (5.3) models that a dishonest agent may leak all information in its state to the adversary. Rule (5.4) models the adversary's capability to arbitrarily modify a dishonest agent's internal state. Finally, Rule (5.5) models that a dishonest agent's fresh constants may be chosen by the adversary.

Channel Assumptions

We extend the Dolev-Yao message deduction rules of the Tamarin model that pertain to insecure channels with rules for confidential, authentic, and secure channels. The

$$\mathcal{DA} := \{\ [\mathsf{AgentState}(A,c,n)] -[\ \mathsf{Dishonest}(A)\ ]\to [\mathsf{Out}(\langle A,c,n\rangle)], \tag{5.3}$$

$$[\mathsf{In}(\langle c',n'\rangle)] -[\ \mathsf{Dishonest}(A)\ ]\to [\mathsf{AgentState}(A,c',n')], \tag{5.4}$$

$$[\mathsf{In}(x')] -[\ \mathsf{Dishonest}(A)\ ]\to [\mathsf{Fresh}(A,x')]\ \} \tag{5.5}$$

Figure 5.1: Dishonest agent rules.

set of channel rules $\mathcal{CH}$ models how protocol agents access insecure, authentic, confidential, and secure (i.e., authentic and confidential) channels shown in Figure 5.2. Rules (5.6) and (5.7) represent insecure channels. The sending of messages over an insecure channel is labeled with the Snd_I action and produces an Out fact, which represents the adversary's capability to learn messages by eavesdropping. Rule (5.7) is annotated with the Rcv_I action and represents the adversary's capability to insert arbitrary messages into insecure channels whenever a protocol agent intends to receive a message from an insecure channel (In).

Rules (5.8) and (5.9) model authentic channels. In Rule (5.8), the adversary learns the message (Out). The auxiliary !Auth fact ensures that in Rule (5.9) the adversary can neither alter the message nor its sender. The !Auth fact is persistent, which reflects the adversary's capability to replay authentically transmitted messages. The rules are annotated with the corresponding Snd_A and Rcv_A actions.

Confidential channels are modeled using Rules (5.10)–(5.12). Rule (5.10) creates an auxiliary !Conf fact and the adversary does not learn the message. Rule (5.11) represents the case where the adversary passes the (unknown) confidential message m to the intended recipient, possibly pretending that it stems from another sender (In). The !Conf fact is persistent, which reflects the adversary's capability to replay confidentially transmitted messages. Rule (5.12) represents the adversary's capability to access the confidential channel to deliver any message from his knowledge.

Rules (5.13) and (5.14) model secure channels. In Rule (5.13), the adversary learns nothing and an auxiliary !Sec fact is generated, which models that the adversary can neither alter the message nor its sender. Rule (5.14) models receiving a message from a secure channel. The !Sec fact is persistent, allowing the adversary to replay securely transmitted messages.

5.2 Channels as Goals

In the preceding section, we defined communication channels as a means for agents to communicate. Here we define the notion of a communication channel as a protocol goal. We use this not only to analyze the security of protocols, but also to reason about the *existence* of protocols that provide a communication channel with a given security property.

$$\mathcal{CH} := \{\ [\mathsf{Snd_I}(A,B,m)] \dashv [\ \mathsf{Snd_I}(A,B,m)\ ] \mapsto [\mathsf{Out}(\langle A,B,m\rangle)], \tag{5.6}$$
$$[\mathsf{In}(\langle A,B,m\rangle)] \dashv [\ \mathsf{Rcv_I}(A,B,m)\ ] \mapsto [\mathsf{Rcv_I}(A,B,m)], \tag{5.7}$$
$$[\mathsf{Snd_A}(A,B,m)] \dashv [\ \mathsf{Snd_A}(A,B,m)\ ] \mapsto [!\mathsf{Auth}(A,m), \mathsf{Out}(\langle A,B,m\rangle)], \tag{5.8}$$
$$[!\mathsf{Auth}(A,m), \mathsf{In}(B)] \dashv [\ \mathsf{Rcv_A}(A,B,m)\ ] \mapsto [\mathsf{Rcv_A}(A,B,m)], \tag{5.9}$$
$$[\mathsf{Snd_C}(A,B,m)] \dashv [\ \mathsf{Snd_C}(A,B,m)\ ] \mapsto [!\mathsf{Conf}(B,m)], \tag{5.10}$$
$$[!\mathsf{Conf}(B,m), \mathsf{In}(A)] \dashv [\ \mathsf{Rcv_C}(A,B,m)\ ] \mapsto [\mathsf{Rcv_C}(A,B,m)], \tag{5.11}$$
$$[\mathsf{In}(\langle A,B,m\rangle)] \dashv [\ \ ] \mapsto [\mathsf{Rcv_C}(A,B,m)], \tag{5.12}$$
$$[\mathsf{Snd_S}(A,B,m)] \dashv [\ \mathsf{Snd_S}(A,B,m)\ ] \mapsto [!\mathsf{Sec}(A,B,m)], \tag{5.13}$$
$$[!\mathsf{Sec}(A,B,m)] \dashv [\ \mathsf{Rcv_S}(A,B,m)\ ] \mapsto [\mathsf{Rcv_S}(A,B,m)]\ \} \tag{5.14}$$

Figure 5.2: Channel rules.

Our use of channels as goals has three aspects we highlight here. First, we consider the *communication of knowledge* rather than the transmission of messages over a network. We formally define this concept in Definition 5.1 and illustrate an application thereafter. Second, to avoid protocols that trivially satisfy security properties by never communicating a useful message, we require that there exists a trace in which security-relevant knowledge is communicated from one honest agent to another. We therefore define and later use the notion of *providing a communication channel*. Finally, we consider as a special case protocols in which a fresh constant generated by the sender can be communicated. This is coarse, but for our purposes sufficient, to differentiate between protocols that allow for the communication of an arbitrary message and protocols that impose limits on the communicated message, such as that it be a yes/no vote.

We define what it means for knowledge to be communicated as follows. We say that an agent S communicates a message m in a trace, if the action $\mathsf{Comm}(S,m)$ appears in the trace. This merely implies that S knows m, but there is no guarantee that m is sent on the network. We say that an agent R learns a message m in a trace, if $\mathsf{Learn}(R,m)$ appears in the trace. This, too, implies that the agent R knows m, but there is no guarantee that R did not know m earlier in the trace. To say that m is communicated from S to R in a trace means that $\mathsf{Comm}(S,m)$ occurs before $\mathsf{Learn}(R,m)$ in the trace. In other words, the agents S and R know m and S performs a protocol step labeled $\mathsf{Comm}(S,m)$ before R performs a protocol step labeled $\mathsf{Learn}(R,m)$.

Definition 5.1 *A message m is* communicated *from an agent S to an agent R in a trace tr, denoted* communicate(tr,S,R,m) *if*

$$\exists\, tr', tr'' \in \mathcal{P}(\mathcal{G})^* : tr = tr' \cdot tr'' \wedge \\ \mathsf{Comm}(S,m) \in tr' \wedge \mathsf{Learn}(R,m) \in tr''.$$

Communicating a message from an agent S to an agent R is more general than transmitting a message from S to R. If R receives a message m from S, then S has communicated m to R. However, a message can be communicated without being sent, as the following example shows.

Example 5.2 *Consider a code sheet consisting of pairs of distinct fresh terms that is shared between S and R. If (x, y) is such a pair, then S can communicate x to R by sending y.*

A protocol in which the sender communicates a message by sending its code limits the sender's communication channel to the messages on the code sheet. This is useful for applications like code voting [21], but it is cumbersome for an email application where senders communicate arbitrary messages. For email, the shared code sheet would be better used to establish a shared cryptographic key for securing subsequent email communication. This, however, is a different protocol and is not an option for humans who cannot perform encryption without supporting technology. For this reason we distinguish between protocols that allow for the communication of a fresh constant generated by the sender and protocols that do not. Thus we use the generation and subsequent communication of a fresh constant as a symbolic representative for the ability to communicate an arbitrary message without requiring an encoding or performing another computational task.

Definition 5.3 *We say that a message m* originates *with an agent A in a trace tr, if m is a fresh constant that A generates, that is, if* $\mathsf{Fresh}(A, m) \in tr$.

We now define what it means for a protocol to provide a particular type of channel. A channel property is a pair of predicates (p, q) each of which has domain $\mathcal{P}(\mathcal{G})^* \times \mathcal{C}_{\mathsf{pub}} \times \mathcal{C}_{\mathsf{pub}} \times \mathcal{M}$. A protocol provides a channel with a property defined by (p, q) if there exists a trace, two honest agents, and a message, such that p is satisfied and if for all traces, agents, and messages, q is satisfied. The existential requirement p ensures that the protocol provides some given functionality, such as communicating messages. The universal requirement q specifies a safety property, for example confidentiality. In order to reason about the (im-)possibility of secure communication, we need both of these requirements.

Definition 5.4 *Protocol $\mathcal{R}$* provides a channel with the property defined by (p, q) *if*

$$\exists\, tr \in TR(\mathcal{R}), S, R \in \mathit{Honest}(tr), m \in \mathcal{M}\colon\ p(tr, S, R, m) \wedge$$
$$\forall\, tr \in TR(\mathcal{R}), S, R \in \mathit{Honest}(tr), m \in \mathcal{M}\colon\ q(tr, S, R, m).$$

We now define several channel properties, starting with the properties related to Definitions 5.1 and 5.3 above and concluding with security properties. A *communication channel* is defined by the property $(p_{\text{comm}}, q_{\text{comm}})$, where

$$p_{\text{comm}}(tr, S, R, m) := \text{communicate}(tr, S, R, m)$$
$$q_{\text{comm}}(tr, S, R, m) := \top.$$

The predicate $\top$ (true) places no additional requirement on the set of traces. We say that a protocol provides a communication channel if the protocol satisfies the communication channel property. Intuitively, this states that the protocol is indeed a functioning communication protocol: it allows an honest agent to communicate a message to another honest agent. We will use analogous terminology for the channel properties to be defined in the remainder of this section.

An *originating* channel is defined by the property $(p_{\text{orig}}, q_{\text{orig}})$, where

$$p_{\text{orig}}(tr, S, R, m) := \mathsf{Fresh}(S, m) \in tr$$
$$q_{\text{orig}}(tr, S, R, m) := \top.$$

This says that a protocol with this property allows agents to generate fresh constants. We use this property together with the communication channel property to model an agent's ability to communicate an arbitrary message.

We say that a protocol *combines* channel properties (p_1, q_1) and (p_2, q_2) if it satisfies the property $(p_1 \wedge p_2, q_1 \wedge q_2)$. In this case we combine the adjectives used to describe the channel properties. For instance, we say that a protocol provides an originating communication channel if it combines an originating channel with a communication channel.

The two channel properties defined above concern the functionality of protocols. We now turn to confidentiality and authenticity of messages, which are safety properties. A channel has the confidentiality property if the adversary does not learn a specified message. To identify the messages m that are intended to remain confidential in a protocol, we annotate a protocol rule with a $\mathsf{Secret}(S, R, m)$ action.

Definition 5.5 *The* confidentiality property *is defined by* (p_{conf}, q_{conf})*, where*

$$p_{conf}(tr, S, R, m) := \mathsf{Secret}(S, R, m) \in tr$$
$$q_{conf}(tr, S, R, m) := \mathsf{Secret}(S, R, m) \in tr \rightarrow \mathsf{!K}(m) \notin tr.$$

A channel has the authenticity property for the agents S and R, if whenever R learns m, then m has previously been communicated by S. To identify messages m that are intended to be authentically communicated in a protocol, we annotate the protocol rule in which such a message is learned with an $\mathsf{Authentic}(S, R, m)$ action.

Definition 5.6 *The* authenticity property *is defined by* (p_{auth}, q_{auth})*, where*

$$p_{auth}(tr, S, R, m) := \mathsf{Authentic}(S, R, m) \in tr$$
$$q_{auth}(tr, S, R, m) := \mathsf{Authentic}(S, R, m) \in tr \rightarrow \mathsf{communicate}(tr, S, R, m).$$

We call the combination of a confidential channel and an authentic channel a *secure* channel.

From now on we will only consider protocols that provide a communication channel combined with additional channel properties. Therefore we will leave out the adjective "communication" for the channels provided by the protocols.

We distinguish between the trust assumptions for confidentiality and authenticity and therefore define two properties.

Definition 5.7 *We define the* trust assumption for confidentiality *by the property* (p_{ctrust}, q_{ctrust})*, where*

$$\begin{aligned} p_{ctrust}(tr, S, R, m) := \exists\ & T \in \mathcal{C}_{\mathsf{pub}}, i \in \{1, \ldots, |tr|\} : \\ & \mathsf{Trust}(T) \in tr_i \wedge \mathsf{Secret}(S, R, m) \in tr_i \\ & \wedge\ T \in \mathit{Honest}(tr) \\ q_{ctrust}(tr, S, R, m) := \forall\ & T \in \mathcal{C}_{\mathsf{pub}}, i \in \{1, \ldots, |tr|\} : \\ & \mathsf{Trust}(T) \in tr_i \wedge \mathsf{Secret}(S, R, m) \in tr_i \\ & \rightarrow T \in \mathit{Honest}(tr). \end{aligned}$$

The trust assumption for authenticity *is defined by the property* (p_{atrust}, q_{atrust})*, where*

$$\begin{aligned} p_{atrust}(tr, S, R, m) := \exists\ & T \in \mathcal{C}_{\mathsf{pub}}, i \in \{1, \ldots, |tr|\} : \\ & \mathsf{Trust}(T) \in tr_i \\ & \wedge\ \mathsf{Authentic}(S, R, m) \in tr_i \\ & \wedge\ T \in \mathit{Honest}(tr) \\ q_{atrust}(tr, S, R, m) := \forall\ & T \in \mathcal{C}_{\mathsf{pub}}, i \in \{1, \ldots, |tr|\} : \\ & \mathsf{Trust}(T) \in tr_i \\ & \wedge\ \mathsf{Authentic}(S, R, m) \in tr_i \\ & \rightarrow T \in \mathit{Honest}(tr). \end{aligned}$$

The two properties state that if a confidentiality or authenticity action occurs with a $\mathsf{Trust}(T)$ action, then agent T is honest. We can use these properties to express that whenever a confidentiality or authenticity claim is made, the specified intended communication partners are assumed to be honest. We achieve this expression with a relativization.

We say that a protocol provides the channel property (p_1, q_1) relative to the channel property (p_2, q_2) if it satisfies the property $(p_1 \wedge p_2, q_1 \vee \neg q_2)$. That is, both existential predicates need to be satisfied, and the universal predicate q_2 implies q_1. For instance, the property $(p_{\text{conf}} \wedge p_{\text{ctrust}}, q_{\text{conf}} \vee \neg q_{\text{ctrust}})$ specifies that a protocol provides a confidential channel *if* the sender's trusted communication partners are honest.

We also use relative channel properties to specify that a communication channel should be provided from a specific protocol role to another.

Definition 5.8 *Let* $\text{RoleMaps}(\mathcal{R}, tr)$ *be the set of all functions that assign to each role of* $\mathcal{R}$ *an agent executing that role in trace tr. The* channel from a role A to a role B *is defined by the property* (p_{role}, q_{role})*, where*

$$\begin{aligned} p_{role}(tr, S, R, m) &:= \exists\ \varphi \in \text{RoleMaps}(\mathcal{R}, tr) : \\ &\qquad \varphi(A) = S \wedge \varphi(B) = R \\ q_{role}(tr, S, R, m) &:= \forall\ \varphi \in \text{RoleMaps}(\mathcal{R}, tr) : \\ &\qquad \varphi(A) = S \wedge \varphi(B) = R \end{aligned}$$

5.3 Protocol Specification

Protocols are specified by a finite number of rules in $\mathcal{R}_{Spec}$. They consist of setup rules and rules defining the behavior of a set of *roles*. The setup rules serve to initialize the protocol roles. They create the agents' initial knowledge, generate initial AgentState facts for all roles, and may mark some agents as honest or dishonest. Recall that AgentState facts are of the form $\mathsf{AgentState}(A, c, n)$, where A is a public term representing an agent's name, c refers to the role step the agent is in, and n is the agent's knowledge at that step. The setup rule containing a $\mathsf{AgentState}(A, c, n)$ fact must contain one of the actions $\mathsf{Agent}(A)$, $\mathsf{Honest}(A)$, or $\mathsf{Dishonest}(A)$. Agents are typically marked with $\mathsf{Agent}(A)$ and are considered honest unless they are corrupted and become dishonest by one of the rules in Figure 5.1. $\mathsf{Honest}(A)$ indicates that an agent A is honest, $\mathsf{Dishonest}(A)$ indicates that the agent is dishonest. Agents cannot be both honest and dishonest. Agents that have been marked with $\mathsf{Honest}(A)$ or $\mathsf{Dishonest}(A)$ cannot change from being honest to become dishonest or vice versa. Formally, a setup rule $l -[\, a \,]\!\to r$ is a rule where:

S1 Only Fresh and Fr facts occur in l.

S2 For every $\mathsf{AgentState}(A, \_, \_)$ fact in r, an $\mathsf{Agent}(A)$, $\mathsf{Honest}(A)$, or $\mathsf{Dishonest}(A)$ action exists in a.

A role consists of a set of protocol rules, specifying the sending and receiving of messages, branching and looping conditions, and the generation of fresh constants. In what follows, we only allow protocols where after the setup phase all information is exchanged using the channels defined in our channel abstraction model above. That is, information may not flow from one agent to another in any way other than by one of the channels defined in $\mathcal{CH}$.

Rules defining a role's behavior must contain a single $\mathsf{AgentState}(A, c, n)$ fact, zero or more Fresh facts, and zero or more receive facts in their premise. The rules must contain zero or more send and $\mathsf{AgentState}(A, \_, \_)$ facts in their conclusion. The protocol rules may be annotated with the actions $\mathsf{Learn}(A, m)$ or $\mathsf{Comm}(A, m)$, where m is a term derivable from A's knowledge n, public constants, and terms received or freshly generated in the protocol rule. Such Learn and Comm actions occur in the protocol's trace and are used to specify security goals as defined in Section 5.2.

A protocol rule $l -[\, a \,]\!\to r$ is a rule such that the following 6 conditions are satisfied.

P1 The facts in l, a, and r do not contain elements of $\mathcal{C}_{\text{fresh}}$ as subterms.

P2 Only $\mathsf{Rcv_I}, \mathsf{Rcv_A}, \mathsf{Rcv_C}, \mathsf{Rcv_S}$, Fresh, and AgentState facts occur in l.

P3 Only $\mathsf{Snd_I}, \mathsf{Snd_A}, \mathsf{Snd_C}, \mathsf{Snd_S}$, and AgentState facts occur in r.

P4 Exactly one AgentState fact occurs in l, zero or more AgentState facts occur in r.

P5 If $\mathsf{AgentState}(A, c, n)$ occurs in l, then

a) every $\mathsf{Rcv_I}$, $\mathsf{Rcv_A}$, $\mathsf{Rcv_C}$, $\mathsf{Rcv_S}$, Fresh fact is of form $\mathsf{Rcv_I}(B, A, x)$, $\mathsf{Rcv_A}(B, A, x)$, $\mathsf{Rcv_C}(B, A, x)$, $\mathsf{Rcv_S}(B, A, x)$, $\mathsf{Fresh}(A, x)$ where $B, x \in \mathcal{T}$,

b) every Learn, Comm, Trust, Secret, Authentic, Verify, $\mathsf{Snd_I}$, $\mathsf{Snd_A}$, $\mathsf{Snd_C}$, $\mathsf{Snd_S}$ fact is of the corresponding form $\mathsf{Learn}(A, x)$, $\mathsf{Comm}(A, x)$, $\mathsf{Trust}(B)$, $\mathsf{Secret}(A, B, x)$, $\mathsf{Authentic}(B, A, x)$, $\mathsf{Verify}(A, B, x)$, $\mathsf{Snd_I}(A, B, x)$, $\mathsf{Snd_A}(A, B, x)$, $\mathsf{Snd_C}(A, B, x)$, $\mathsf{Snd_S}(A, B, x)$, where $B \in \mathcal{C}_{\mathsf{pub}}$, $x \in \mathcal{T}$ and x is derivable from terms in $\mathcal{C}_{\mathsf{pub}}$, terms in Fresh and $\mathsf{Rcv_I}, \mathsf{Rcv_A}, \mathsf{Rcv_C}, \mathsf{Rcv_S}$ facts occurring in l, and terms in n.

c) every AgentState fact in r is of the form $\mathsf{AgentState}(A, c', n')$, where $c' \in \mathcal{C}_{\mathsf{pub}}$ and n' derivable from terms in $\mathcal{C}_{\mathsf{pub}}$, terms in Fresh and $\mathsf{Rcv_I}, \mathsf{Rcv_A}, \mathsf{Rcv_C}, \mathsf{Rcv_S}$ facts occurring in l, and terms in n.

P6 $vars(r) \subseteq vars(l) \cup \mathcal{V}_{\mathsf{pub}}$.

Remark. *A protocol rule that contains a receive fact in its premise and a send fact in its conclusion models the reception and sending of messages as an* atomic *protocol execution step. If the agent executing the protocol step is dishonest, then the adversary may not be able to influence the message to be sent. To model the general situation where reception and subsequent sending of messages are not atomic, two separate rules need to be specified, one for the reception of messages and a corresponding update of the receiver's state, and another one to specify the sending of messages. The adversary may then reveal and modify a dishonest agent's state after the dishonest agent receives a message and before the agent sends the subsequent message.*

To be able to reconstruct all system states from a trace, we add a unique action R_i to every rule in $\mathcal{R}$. Formally, we do this as follows. Let q be a sequence of all rules in $\mathcal{R}$ such that every rule in $\mathcal{R}$ occurs exactly once in q. The action R_i contains all variables of the rule q_i in q as an argument. To this end, we must map the elements of the set of variables in the premises and conclusions to an ordered list. We denote such a map by *list*. Thus the set of rules that allows us to reconstruct all system states from a trace for a given protocol specification $\mathcal{R}$ is given by

$$\begin{aligned}\{\, l \mathrel{-[} a \mathrel{]\!\!\to} r \mid \exists i \in \{1, \ldots, |q|\} :\ & l \mathrel{-[} a' \mathrel{]\!\!\to} r = q_i \wedge \\ & a = a' \cdot [\mathsf{R}_i(list(vars(l) \cup vars(r)))] \,\}.\end{aligned} \tag{5.15}$$

For ease of reading, we represent protocols in an extended Alice & Bob notation from which the corresponding protocol rules can be easily obtained. The extension concerns the symbols representing insecure $\circ\!\!\to\!\!\circ$, authentic $\bullet\!\!\to\!\!\circ$, confidential $\circ\!\!\to\!\!\bullet$, and secure $\bullet\!\!\to\!\!\bullet$ (i.e., authentic and confidential) channels. This channel notation is adapted from Maurer and Schmid's channel calculus [55]. The solid dot "$\bullet$" indicates exclusive access to the channel, in contrast to the empty dot "$\circ$" which does not indicate such a guarantee. For instance, we write $A \circ\!\!\to\!\!\circ B : m$ to express that a message m is to be sent from an agent executing role A to an agent executing role B over an insecure channel. To express that the message is sent over an authentic channel, we write $A \bullet\!\!\to\!\!\circ B : m$, whereby only A can send messages using the authentic channel.

In general, an Alice & Bob specification leaves room for different interpretations [18]. When such ambiguities arise, we indicate both the message sent and the message pattern expected to be received and separate them with " / ", as in $A \circ\!\!\rightarrow\!\!\circ\ B : m\ /\ m'$. The variables in m' determine how the received message is parsed by an agent executing role B. To express the initial knowledge m of an agent executing role A, we write $A : \text{knows}(m)$. To express that the agent generates one or more fresh constants $m_1, \ldots, m_k$, we write $A : \text{fresh}(m_1, \ldots, m_k)$ or $A \circ\!\!\rightarrow\!\!\circ\ B : \text{fresh}(m_1, \ldots, m_k).m$ when the generation is followed by a send event.

The above introduced model allows us to examine systems where humans, computers, and servers communicate. We use this model in Chapter 6 to prove our theorems regarding the necessary and sufficient conditions for secure human-server communication.

6 Secure Communication Using Insecure Platforms

We use the formal model defined in Chapter 5 to classify communication topologies of electronic communication systems such as Internet voting systems. The result is a complete characterization of the necessary and sufficient conditions a communication topology must satisfy to provide the confidential and authentic transmission of messages such as ballots in an Internet voting setting. We identify minimal communication topologies with respect to the availability of communication links, channel assumptions, and trusted protocol agents.

6.1 Communication Topologies

The communication topology model we introduce in this section specifies nodes and links. The specification provides assumptions relative to which a communication protocol's security properties are analyzed. Every node in the topology corresponds to a unique role in the protocol specification. The role specifies the node's behavior. The communication topology specifies the node's capabilities, initial knowledge, honesty, and the available communication channels. We define a class of communication topologies which is of particular interest in the remainder of this report. It concerns the set of protocols where a human user securely communicates with a remote server using a potentially compromised computer.

Communication Topology Model

A communication topology (relative to a signature Σ) is an edge- and vertex-labeled directed graph (V, E, η, μ), where V is the set of vertices, $E \subseteq V \times V$, and η and μ are functions assigning labels to vertices and edges respectively. We call a sequence of vertices $[v_1, \ldots, v_{k+1}] \in V^*$, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k$, a *path from* v_1 *to* v_{k+1} *of length* k or simply a *path*. The path is *acyclic* if $v_i \neq v_j$ for all $1 \leq i < j \leq k+1$. We denote the transitive closure of E by E^+, i.e., we write $(v_i, v_j) \in E^+$ if there is a path from v_i to v_j.

The set of vertices V represents a protocol's roles. For $A, B \in V$, an edge $(A, B) \in E$ denotes the existence of a link from a node representing role A to the node represent-

ing role B. The vertex labeling function $\eta : V \to \mathcal{P}(\Sigma) \times \mathcal{P}(\mathcal{T}) \times \{\mathsf{honest}, \mathsf{dishonest}\}$ assigns capability, initial knowledge, and trust assumptions to role names. The elements in $\{\mathsf{honest}, \mathsf{dishonest}\}$ indicate the trust assumptions placed in a role. Agents marked dishonest are assumed to be controlled by the adversary via the dishonest agent rules in $\mathcal{DA}$ shown in Section 5.1. Agents marked honest are assumed to faithfully execute the security protocol. Agents always have a finite initial knowledge. The initial knowledge assumption is indicated as a subset of $\mathcal{T}$. It indicates the *maximal* initial knowledge an agent is allowed to have. An empty set indicates that the agent is assumed to have no initial knowledge, while $\mathcal{T}$ indicates that no restrictions are placed on the agent's initial knowledge other than that it is a finite set. The capability assumption is indicated as a subset of Σ. It consists of the function symbols available to honest agents executing the role that is represented by the node. The edge labeling function $\mu : E \to \{\circ\!\!\rightarrow\!\!\circ, \bullet\!\!\rightarrow\!\!\circ, \circ\!\!\rightarrow\!\!\bullet, \bullet\!\!\rightarrow\!\!\bullet\}$ assigns channel assumptions to links.

Graphical Representation

We graphically represent a communication topology (V, E, η, μ) as follows. Vertices $A \in V$ are drawn as simple, concentric, or dashed circles depending on the labeling η. To express that a role $A \in V$ is assumed to be executed by a dishonest agent, i.e., $\eta(A) = (\Sigma_A, K_A, \mathsf{dishonest})$ for $\Sigma_A \subseteq \Sigma$, $K_A \subseteq \mathcal{T}$, we draw concentric circles. A dashed circle indicates that an honest agent executing the role A has restricted capabilities, i.e., $\Sigma_A \subsetneq \Sigma$. Note that our vertex representation does not distinguish between different types of restricted capabilities and knowledge. This limitation suffices for the present paper. Edges $e \in E$ are drawn as arrows connecting the circles and are labeled according to μ. The edge labels are written next to the arrows representing the corresponding edges.

Figure 6.1 depicts an example of a communication topology (V, E, η, μ) with $V = \{A, B, C, P, Q, R\}$. In this example, the role A is assumed to be executed by an honest restricted agent, thus $\eta(A) = (\Sigma_A, K_A, \mathsf{honest})$ for some $\Sigma_A \subsetneq \Sigma$ and $K_A \subseteq \mathcal{T}$. The role B is assumed to be executed by a dishonest agent, thus $\eta(B) = (\Sigma, K_B, \mathsf{dishonest})$ for some $K_B \subseteq \mathcal{T}$. The remaining roles are assumed to be executed by honest (and unrestricted) agents. The set of edges E and their labeling can be read off of Figure 6.1. For example, $(A, B) \in E$, $(B, A) \notin E$, and $(B, A) \in E^+$. The link from A to B is secure and the link from B to C is insecure.

6.2 Human-Interaction Security Protocols

In the following we introduce the class of security protocols where humans intend to securely communicate with a remote server. We assume that humans have no initial knowledge and are limited to the following functions. They may send, receive, and compare terms. Moreover, humans may pair and select (project) terms. We do not impose any restriction on human memory and we assume that humans can generate fresh values. Thus, humans are assumed to be able to remember all terms received

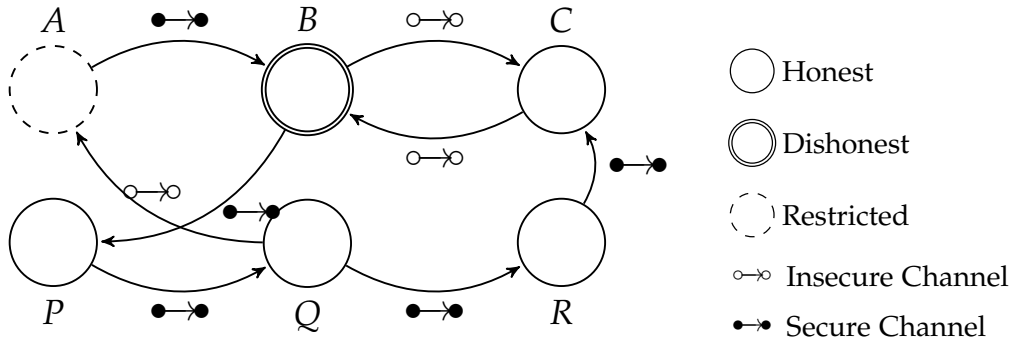


Figure 6.1: Communication topology example.

on any channel and to input any term in their knowledge to any other channel. However, they cannot perform cryptographic operations without the help of supporting technology.

Consider now the set of protocols that provide a secure communication channel between a human and a server. We can model these protocols' topology by (V', E', η', μ'), where $V' = \{H, S\}$, S denotes the remote server's role, H is the human's role, and $\mu'(H, S) = \bullet\!\!\rightarrow\!\!\bullet$. However, this communication topology is too abstract to reason about the requirements a protocol must satisfy to provide a secure channel from the human to the server. A natural step in making this model more concrete is to assume that the human cannot directly communicate with the remote server. The human must instead use a computing platform P that communicates with the remote server over an insecure network. The resulting refinement of the initial topology is $(V'', E'', \eta'', \mu'')$, where $V'' = \{H, S, P\}$, $\mu''(H, P) = \mu''(P, H) = \bullet\!\!\rightarrow\!\!\bullet$, and $\mu''(P, S) = \mu''(S, P) = \circ\!\!\rightarrow\!\!\circ$. If we assume that the computing platform P is honest, then this topology represents the well-known problem of establishing a secure communication channel between two agents over an insecure network.

We are instead interested here in the case where the computing platform is compromised, i.e., $\eta''(P) = (\Sigma, K_P, \mathsf{dishonest})$. We also want to explicitly model that humans are computationally limited and that H and S initially do not share any fresh knowledge. It is evident (and follows from Theorems 6.3 and 6.4 in Section 6.3) that secure communication between H and S is not possible in such a topology. Thus we model that the human has access to a trusted device D by including D in the topology. Examples of such devices include a list of one-time passwords, a code sheet, or a smart card with a corresponding card reader.

Protocols to establish secure communication between the human and a remote server under these circumstances are highly relevant in practice, most prominently in online banking and Internet voting. We call such a protocol a *human-interaction security protocol*, or *HISP* for short, and the corresponding communication topology a *HISP topology* defined in the following.

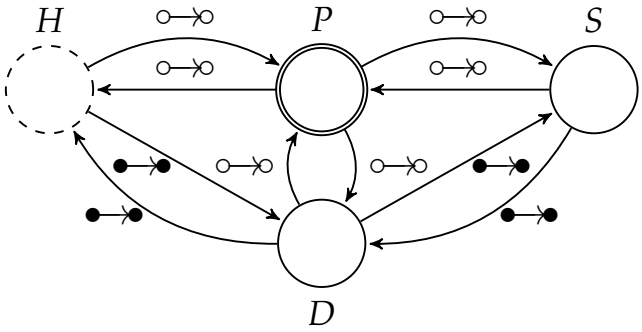


Figure 6.2: The supergraph of all HISP topologies.

Definition 6.1 *A* HISP topology *is a topology* (V, E, η, μ)*, where the set of roles is* $V = \{H, D, S, P\}$ *and the set of links is* $E \subseteq \{(H,P),(P,H),(D,P),(P,D),(P,S),(S,P),(H,D),(D,H),(D,S),(S,D)\}$*. The vertex labeling is given by* $\eta(H) = (\Sigma_H, \emptyset, \mathsf{honest})$*,* $\eta(D) = (\Sigma, \mathcal{T}, \mathsf{honest})$*,* $\eta(S) = (\Sigma, \mathcal{T}, \mathsf{honest})$*, and* $\eta(P) = (\Sigma, \mathcal{T}, \mathsf{dishonest})$*, where* $\Sigma_H = \{\langle \_, \_ \rangle, \pi_1(\_), \pi_2(\_)\} \cup \mathcal{C}_{\mathsf{pub}} \cup \mathcal{C}_{\mathsf{fresh}}$*. The edge labeling is given by* $\mu(e) = \circ\!\!\rightarrow\!\!\circ$*, for* $e \in E_1 \cap E$*, and* $\mu(e) = \bullet\!\!\rightarrow\!\!\bullet$*, for* $e \in E_2 \cap E$*, where* $E_1 = \{(H,P),(P,H),(D,P),(P,D),(P,S),(S,P)\}$ *and* $E_2 = \{(H,D),(D,H),(D,S),(S,D)\}$*.*

The definition states that a HISP topology consists of a human H, a server S, and a device D, which are assumed to be honest, and a computing platform P, which is assumed to be dishonest. There are no restrictions on capabilities or initial knowledge of S, P, and D. H is restricted to the functions in Σ_H and assumed to have no initial knowledge. Figure 6.2 shows the supergraph of all HISP topologies (V, E, η, μ) and indicates the edge labels. Since the edge labels are constant, we omit them in graphical representations of HISP topologies in the remainder of this paper. Such a representation is shown in Figure 6.3. Its description is given in Example 6.16.

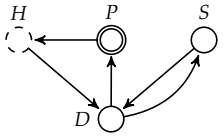
As another example suppose a bank provides its customers with verification messages via short message service. If both the cellular network and the customer's mobile phone are assumed to behave honestly, then the cellular network is represented by $(S, D) \in E$ and the mobile phone's functionality is contained in the role D. However, if we make the more realistic assumption that the mobile phone may behave dishonestly [32], then the mobile phone's behavior specification is contained in the role P and the cellular network is represented by $(S, P) \in E$.

6.3 Complete Characterization of Secure HISPs

In this section we classify all HISP topologies for which there exist HISPs that provide secure communication channels. As opposed to HISPs providing originating secure channels, these HISPs may restrict the communication partners to a pre-defined set of messages that can be securely exchanged, such as codewords for candidates in

an Internet voting system. Due to the weaker requirements regarding the origin of the exchanged messages, the set of HISP topologies in which a protocol providing a secure channel exists is a superset of the former set of topologies. The following example illustrates a HISP providing a secure channel but not an originating secure channel.

Example 6.2 *Suppose the human has a device which contains a keypad and a direct link to the remote server, but no display. The device is connected to the user's computer. Then there is a protocol that provides a secure, but not an originating secure, channel from the remote server to the human user. The following graph depicts this HISP topology.*



The protocol proceeds as follows. The human user enters two messages x_1, x_2 into the device. The device sends the messages securely to the remote server. The server chooses one of the two messages, say x_1, and sends it securely to the device. The device sends the other message, x_2, to the platform and the platform displays it to the human user. The human user now knows that the server communicated x_1, but the platform has no knowledge of x_1. The protocol is shown in Alice & Bob notation below. The correctness of the security claim is shown in the proof of Theorem 6.4. By Theorem 6.21, there is no protocol in this topology that provides an originating confidential channel from the server to the human user.

Protocol Example 6.2

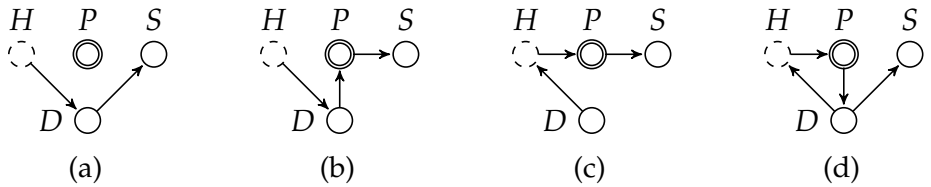
$$
\begin{aligned}
H &\bullet\!\!\rightarrow\!\!\bullet\ D : \mathrm{fresh}(x_1, x_2).\langle S, x_1, x_2\rangle \\
D &\bullet\!\!\rightarrow\!\!\bullet\ S : \langle H, x_1, x_2\rangle \\
S &\bullet\!\!\rightarrow\!\!\bullet\ D : x_1 \\
D &\circ\!\!\rightarrow\!\!\circ\ P : x_2 \\
P &\circ\!\!\rightarrow\!\!\circ\ H : x_2
\end{aligned}
$$

We now classify all HISPs with respect to protocols providing secure channels from H to S and vice versa. Our main results are stated in the following two theorems. Theorem 6.3 shows the four minimal HISP topologies in which a protocol exists that provides a secure communication channel from a human H to a server S.

Theorem 6.3 *For a HISP to provide a secure channel H to S, the underlying topology must either contain an edge from D to H and a path from H to S or contain an edge from H to D and a path from D to S. All minimal graphs satisfying these conditions are shown below.*

	$(D,H) \notin E$ $\wedge (H,D) \notin E$	$(D,H) \notin E$ $\wedge (H,D) \in E$ $\wedge (D,S) \notin E^+$	$(D,H) \notin E$ $\wedge (H,D) \in E$ $\wedge (D,S) \in E^+$	$(D,H) \in E$
A	no (Lemma 6.8)	no (Lemma 6.9)	yes (Lemma 6.11)	yes (Lemma 6.13)
C	no (Lemma 6.8)	no (Lemma 6.9)	yes (Lemma 6.11)	yes (Lemma 6.13)
S	no (Lemma 6.8)	no (Lemma 6.9)	yes (Lemma 6.11)	yes (Lemma 6.13)

Table 6.1: Classification of all HISP topologies that contain a path H to S. The cells state whether protocols providing authentic (**A**), confidential (**C**), or secure (**S**) channels from H to S exist under the conditions shown on top.



Proof We prove the theorem with Table 6.1 and the lemmas referenced therein as follows. Table 6.1 classifies all HISP topologies that contain a path from H to S. For all other topologies no protocol that provides an authentic, confidential, or secure channel from H to S can exist, because no information can be communicated from H to S. The conditions shown at the top of the table partition all of the topologies that contain a path from H to S first into two classes depending on the existence of an edge from D to H (first row). The class of topologies in which no such edge exists, is then further partitioned into two classes depending on the existence of an edge from H to D (second row). Finally, the subclass containing an edge from H to D is partitioned depending on the existence of a path from D to S (third row). Thus all possible HISP topologies are considered. For each of the four resulting classes a lemma proving the existence or non-existence of protocols is referenced in the table. □

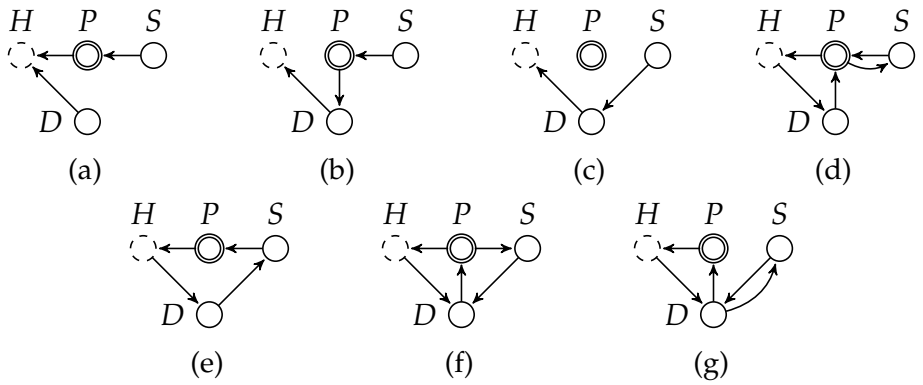
Theorem 6.4 shows the seven minimal HISP topologies in which a protocol exists that provides a secure communication channel from a server S to a human H. It's proof is analogous to the proof of Theorem 6.3 and follows from Table 6.2 and the Lemmas referenced therein.

Theorem 6.4 *For a HISP to provide a secure channel S to H, the underlying topology must either contain an edge from D to H and a path from S to H or contain an edge from H to*

	$(D,H) \notin E$ $\wedge (H,D) \notin E$	$(D,H) \notin E$ $\wedge (H,D) \in E$ $\wedge (D,H) \notin E^+$	$(D,H) \notin E$ $\wedge (H,D) \in E$ $\wedge (D,H) \in E^+$	$(D,H) \in E$
A	no (Lemma 6.8)	no (Lemma 6.10)	yes (Lemma 6.14)	yes (Lemma 6.12)
C	no (Lemma 6.8)	no (Lemma 6.10)	iff $(H,S) \in E^+$ (Lemma 6.15)	yes (Lemma 6.12)
S	no (Lemma 6.8)	no (Lemma 6.10)	iff $(H,S) \in E^+$ (Lemma 6.15)	yes (Lemma 6.12)

Table 6.2: Classification of all HISP topologies that contain a path S to H. The cells state whether protocols providing authentic (**A**), confidential (**C**), or secure (**S**) channels from S to H exist under the conditions shown on top.

D and a path from S to itself which includes D and H. All minimal graphs satisfying these conditions are shown below.



Note that Tables 6.1 and 6.2 also characterize HISP topologies with respect to authentic and confidential communication channels between H and S. The impossibility and possibility lemmas referred to in the tables are stated in Sections 6.3 and 6.3, respectively.

Impossibility Results

We first prove two folklore lemmas. The first lemma states that a confidential channel cannot be created between two agents communicating over insecure channels when one of them has an empty initial knowledge. In fact, the lemma we prove is even stronger: It states that there is no protocol providing a confidential channel from an

honest agent S to another honest agent R, even if S may send messages via authentic channels to R and R may send messages via confidential channels to S.

To express that certain types of channels do not exist from S to R we make use of the following predicate. This predicate states that in trace tr, S does not send any message over a channel of type i and R does not receive any message over a channel of type i, where $i \in \{\mathsf{I}, \mathsf{A}, \mathsf{C}, \mathsf{S}\}$.

$$\begin{aligned}&\mathrm{nochan}(i, S, R, tr) :=\\ &\forall m \in \mathcal{T}, S', R' \in \mathit{Agents}(tr) :\\ &\mathsf{Snd}_i(S, R', m) \notin tr \wedge \mathsf{Rcv}_i(S', R, m) \notin tr\end{aligned}$$

Lemma 6.5 *Let $S, R \in \mathit{Honest}(tr)$ be two distinct honest agents such that at least one of them has an empty initial knowledge and let $\mathcal{R}$ be a protocol. If the following condition is met, then $\mathcal{R}$ does not provide a confidential communication channel from S to R.*

$$\begin{aligned}&\forall tr \in TR(\mathcal{R}) :\\ &\mathrm{nochan}(\mathsf{C}, S, R, tr) \wedge \mathrm{nochan}(\mathsf{S}, S, R, tr)\\ &\mathrm{nochan}(\mathsf{A}, R, S, tr) \wedge \mathrm{nochan}(\mathsf{S}, R, S, tr)\end{aligned}$$

The proof idea for Lemma 6.5 is simple: The adversary impersonates R to S. This is possible because the messages from R to S are not authenticated. Thus, S cannot distinguish between information that R sends to S and information that the adversary sends.

Proof We start by simplifying the protocol when the number of roles specified for the protocols is greater than two. Suppose that S has an empty initial knowledge. Then we combine all roles other than the role of S into the role of R. If the initial knowledge of S is not empty, then by hypothesis, the initial knowledge of R must be empty. In this case we combine all roles other than the role of R into the role of S. This type of transformation preserves confidentiality: If the original protocol provides a confidential communication channel from S to R with role specifications for several other agents, then it provides it in particular for traces where the additional roles are instantiated with the honest agents S and R.

We may thus assume that the protocol contains only the two agents S and R. We may further assume without loss of generality that S transmits and R receives all messages over an authentic channel and that R transmits and S receives all messages over a confidential channel. That is, we may upgrade all insecure channels to channels with these stronger guarantees.

Let tr be a shortest trace satisfying the confidentiality condition (Definition 5.5) and the communication condition (Definition 5.1). Then we have $\mathsf{Secret}(S, R, m) \in tr$, $\mathsf{Comm}(S, m) \in tr$, and $\mathsf{Learn}(R, m) \in tr$ for some $m \in \mathcal{M}$. If there is no such trace, then we are done, since then the protocol does not provide a confidential communication channel. Otherwise, we have that $!\mathsf{K}(m) \notin tr$. We exhibit a trace tr' in which $\mathsf{Secret}(S, R, m) \in tr'$ and $!\mathsf{K}(m) \in tr'$. Let g be the sequence of ground instances of

rules which gives rise to the trace tr. By Equation (5.15), we can obtain this sequence from the trace tr by using the unique facts R_i appearing in the trace.

We construct a sequence of (ground) rewriting rules g' from g that give rise to a trace tr' in which the confidentiality condition is not satisfied. To this end, we will replace rules in g which contain $\mathsf{AgentState}(R, \_, \_)$ by instantiations of rules in $\mathcal{MD}$ and $\mathcal{CH}$. In order for such a transformation to produce a valid sequence of rewriting rules, we need to satisfy the following two conditions:

- Facts consumed by a rule g'_i must have been produced by a rule g'_j, for $j < i$.
- Every rule g'_i is a ground instantiation of a protocol rule in $\mathcal{R}$.

We obtain the transformation from g to g' by describing a series of deletions and insertions performed on the sequence g. For a rule g_i in g, $l(g_i)$ refers to the premises of g_i, $a(g_i)$ to the actions, $r(g_i)$ to the consequences. Thus, $g_i = [l(g_i)] \text{–}[\, a(g_i) \,]\!\rightarrow [r(g_i)]$.

1. For ease of reference, we keep track of the correspondence between the fresh terms in the knowledge of agent R and the adversary's fresh terms via the partial map $\varphi : \mathcal{C}_{\text{fresh}} \to \mathcal{C}_{\text{fresh}}$.

2. For every setup rule g_i containing an $\mathsf{AgentState}(R, c, n)$ fact for some $c, n \in \mathcal{M}$ we make the following two insertions.

   Insertion 1. For every fact $\mathsf{Fresh}(R, y) \in l(g_i)$ there are unique rules $g_k = [\,] \text{–}[\,]\!\rightarrow [\mathsf{Fr}(y)]$ and $g_j = [\mathsf{Fr}(y)] \text{–}[\,]\!\rightarrow [\mathsf{Fresh}(R, y)]$, $k < j < i$, producing $\mathsf{Fresh}(R, y)$.

   We insert an instantiation of the $\mathcal{FR}$ rule $[\,] \text{–}[\,]\!\rightarrow [\mathsf{Fr}(x)]$ immediately after g_k and an instantiation of the $\mathcal{MD}$ rule $[\mathsf{Fr}(x)] \text{–}[\,]\!\rightarrow [!\mathsf{K}(x)]$ immediately after g_j. We set $\varphi(y) := x$.

   Insertion 2. For every public constant $C : pub$ in R's knowledge n, we insert a rule $[\,] \text{–}[\,]\!\rightarrow [!\mathsf{K}(C : pub)]$ before g_i.

   After these insertions, we have a correspondence between R's initial knowledge and the adversary's knowledge. The modified sequence of rules remains a valid sequence.

3. Let g_i be the first instantiation of a role specification rule in g that contains an $\mathsf{AgentState}(R, c, n)$ fact for some $c, n \in \mathcal{M}$. By **P2** through **P4** and our hypothesis, we have only $\mathsf{Fresh}(R, \_)$, $\mathsf{Rcv}_{\mathsf{A}}(\_, R, \_)$, and $\mathsf{AgentState}(R, \_, \_)$ facts in $l(g_i)$, $\mathsf{Snd}_{\mathsf{C}}(R, \_, \_)$ and $\mathsf{AgentState}(R, \_, \_)$ facts in $r(g_i)$, and $\mathsf{Learn}(R, \_)$, $\mathsf{Comm}(R, \_)$, $\mathsf{Secret}(R, \_, \_)$, and $\mathsf{Authentic}(\_, \_, \_)$ facts in $a(g_i)$. We delete the rule g_i after having made the following changes.

   Change 1. For every $\mathsf{Fresh}(R, x)$ fact in $l(g_i)$, there exists a rule $g_j = [\mathsf{Fr}(x)] \text{–}[\,]\!\rightarrow [\mathsf{Fresh}(R, x)]$, $j < i$, producing that fact. We replace g_j by the rule $[\mathsf{Fr}(x)] \text{–}[\,]\!\rightarrow [!\mathsf{K}(x)]$. Thus every fresh term learned by R in g is learned by the adversary in g'.

Change 2. For every $\mathsf{Rcv_A}(S, R, m)$ fact we insert before g_i the rule $[\mathsf{Out}(\langle S, R, m\rangle)] \!-\![\ ]\!\!\rightarrow [!\mathsf{K}(\langle S, R, m\rangle)]$, which is an instantiation of $\mathcal{MD}$ Rule (3.3), and two instantiations of $\mathcal{MD}$ Rule (3.7) using the projecting functions in order to arrive at the facts $!\mathsf{K}(m)$, $!\mathsf{K}(S)$, $!\mathsf{K}(R)$. Note that there exists an $\mathsf{Out}(\langle S, R, m\rangle)$ fact in $r(g_j)$ for some $j < i$ due to instantiations of $\mathcal{CH}$ Rules (5.8) and (5.9) which are the source of the $\mathsf{Rcv_A}(S, R, m)$ fact.

Thus every message received by R in g is learned by the adversary in g'.

Change 3. By step 2 above (i.e., modifications of the setup rules), **P5**(c), and previous applications of the present step, all terms in $\mathsf{AgentState}(R, c, n) \in l(g_i)$ that are derivable from n, are also derivable from the adversary's knowledge up to substitution of fresh constants y in the domain of φ by $\varphi(y)$.

Change 4. For each $\mathsf{Snd_C}(R, S, m)$ fact in $r(g_i)$, we can synthesize from the adversary's knowledge a message $\tilde{m}$ that is equal to m up to substitution of fresh constants y in the domain of φ by their image $\varphi(y)$. To this end, we insert after g_i instantiations of $\mathcal{MD}$ Rule (3.7) to produce the fact $!\mathsf{K}(\langle R, S, \tilde{m}\rangle)$. We delete the corresponding rule $g_j = [\mathsf{Snd_C}(R, S, m)] \!-\![\ \mathsf{Snd_C}(R, S, m)\ ]\!\!\rightarrow [!\mathsf{Conf}(S, m)]$, $j > i$, if it exists, and replace every subsequent rule $[!\mathsf{Conf}(S, m), \mathsf{In}(R)] \!-\![\ \mathsf{Rcv_C}(R, S, m)\ ]\!\!\rightarrow [\mathsf{Rcv_C}(R, S, m), !\mathsf{Conf}(S, m)]$ with $[!\mathsf{K}(\langle R, S, \tilde{m}\rangle)] \!-\![\ !\mathsf{K}(\langle R, S, \tilde{m}\rangle)\ ]\!\!\rightarrow [\mathsf{In}(\langle R, S, m\rangle)]$ and $[\mathsf{In}(\langle R, S, m\rangle)] \!-\![\ ]\!\!\rightarrow [\mathsf{Rcv_C}(R, S, m)]$. The latter of these rules is an instantiation of $\mathcal{CH}$ Rule (5.12) and the former is an incorrect instantiation of $\mathcal{MD}$ Rule (3.4). This is due to a mismatch between the adversary's knowledge $!\mathsf{K}(\langle R, S, \tilde{m}\rangle)$ and the produced fact $\mathsf{In}(\langle R, S, m\rangle)$. This is resolved in step 4 below.

Change 5. Note that each AgentState fact in $r(g_i)$ is of the form $\mathsf{AgentState}(R, c, n)$, where the terms c and n are derivable from !K facts up to substitution of fresh constants in the domain of the φ function.

Change 6. For each $\mathsf{Learn}(R, x)$ fact in $a(g_i)$, we insert instantiations of $\mathcal{MD}$ Rule (3.7) after g_i in order to arrive at $!\mathsf{K}(x)$ (up to substitutions of fresh constants in the domain of φ). This is possible, since x is a term derivable from public constants, messages in $\mathsf{Rcv_A}(S, R, m)$ facts and knowledge in $\mathsf{AgentState}(R, c, n)$ facts.

Change 7. We may ignore the Authentic and Secret facts in $a(g_i)$: We may ignore the Authentic facts, since these indicate an authenticity claim that we are not considering here. We may ignore Secret facts occurring in an instance of a rule considered here, since these concern the confidentiality of messages sent by R, as opposed to those sent by S. We may ignore the Comm facts because they label the transmission of messages from R rather than those from S.

We repeat step 3 as long as rules g_i containing $\mathsf{AgentState}(R, \_, \_)$ facts in $l(g_i)$ are there.

4. We exchange the fresh values y in the initial knowledge of R acquired in the setup

rules with the corresponding fresh values $\varphi(y)$ in the adversary's knowledge ($!\mathsf{K}(\varphi(y))$) as follows.

For every setup rule g_i containing an $\mathsf{AgentState}(R, c, n)$ fact and a $\mathsf{Fresh}(R, y)$ fact, we replace in all terms the fresh constant y by the fresh constant $\varphi(y)$. We replace the unique rule g_j, $j < i$, producing the fact $\mathsf{Fresh}(R, y)$ by the rule $[\mathsf{Fr}(\varphi(y))] -[\ ]\!\rightarrow [\mathsf{Fresh}(R, \varphi(y))]$.

For every instantiation of a $\mathcal{MD}$ rule in g, we replace in all terms all fresh constants $\varphi(y)$ by y.

After the above replacements, we obtain a sequence of rules and consequently a trace tr' in which the adversary impersonates R. R does not perform any protocol steps other than having its initial knowledge set up.

We finally append rule $[!\mathsf{K}(m)] -[\ !\mathsf{K}(m)\ ]\!\rightarrow [\mathsf{In}(m)]$ to g' in order to have $!\mathsf{K}(m) \in tr'$. Thus we have a trace where the adversary learns m, yet $\mathsf{Secret}(S, R, m) \in tr'$. □

The following lemma states the dual of the preceding one: If an honest agent S has no access to an authentic (or secure) channel and another honest agent R has no access to a confidential (or secure) channel, then there is no protocol that provides an authentic channel from S to R.

Lemma 6.6 *Let $S, R \in Honest(tr)$ be distinct honest agents such that at least one of them has an empty initial knowledge and let $\mathcal{R}$ be a protocol. If the following condition is met, then $\mathcal{R}$ does not provide an authentic channel from S to R.*

$$\begin{aligned}&\forall tr \in TR(\mathcal{R}) :\\ &\mathrm{nochan}(\mathsf{A}, S, R, tr) \wedge \mathrm{nochan}(\mathsf{S}, S, R, tr)\\ &\mathrm{nochan}(\mathsf{C}, R, S, tr) \wedge \mathrm{nochan}(\mathsf{S}, R, S, tr)\end{aligned}$$

The proof idea for this lemma is the same as for the preceding one. The adversary impersonates S to R. This is possible, since messages from S to R are not authenticated. Thus, R cannot distinguish between information that S sends to R and information that the adversary sends.

Proof We again start by simplifying the protocol when the number of roles specified for the protocols is greater than two. Suppose that S has an empty initial knowledge. Then we combine all roles other than the role of S into the role of R. If the initial knowledge of S is not empty, then by hypothesis, the initial knowledge of R must be empty. In this case we combine all roles other than the role of R into the role of S. This type of transformation preserves authenticity: If the original protocol provides an authentic communication channel from S to R with role specifications for several other agents, then it provides it in particular for traces where the additional roles are instantiated with the honest agents S and R.

We may thus assume that the protocol contains only the two agents S and R. We may further assume without loss of generality that S transmits and R receives all

messages over a confidential channel and that R transmits and S receives all messages over an authentic channel. That is, we may upgrade all insecure channels to channels with these stronger guarantees.

Let tr be a shortest trace satisfying the authenticity condition (Definition 5.6). Then we have $\mathsf{Authentic}(S, R, m) \in tr$, $\mathsf{Comm}(S, m) \in tr$, and $\mathsf{Learn}(R, m) \in tr$ for some $m \in \mathcal{M}$. If there is no such trace, then we are done, since then the protocol does not provide an authentic communication channel. We exhibit a trace tr' in which $\mathsf{Authentic}(S, R, m) \in tr'$ and $\mathsf{Learn}(R, m) \in tr'$ but $\mathsf{Comm}(S, m) \notin tr'$. Let g be the sequence of ground instances of rules which gives rise to the trace tr. By Equation (5.15), we can obtain this sequence from the trace tr by using the unique facts R_i appearing in the trace.

We construct a sequence of (ground) rewriting rules g' from g that give rise to a trace tr' in which the authenticity condition is not satisfied. To this end, we will replace rules in g which contain $\mathsf{AgentState}(S, \_, \_)$ by instantiations of rules in $\mathcal{MD}$ and $\mathcal{CH}$. In order for such a transformation to produce a valid sequence of rewriting rules, we need to satisfy the following two conditions:

- Facts consumed by a rule g'_i must have been produced by a rule g'_j, for $j < i$.
- Every rule g'_i is a ground instantiation of a protocol rule in $\mathcal{R}$.

We obtain the transformation from g to g' by describing a series of deletions and insertions performed on the sequence g. For a rule g_i in g, $l(g_i)$ refers to the premises of g_i, $a(g_i)$ to the actions, $r(g_i)$ to the consequences. Thus, $g_i = [l(g_i)]\!-\![\, a(g_i) \,]\!\rightarrow\![r(g_i)]$.

1. For ease of reference, we keep track of the correspondence between the fresh terms in the knowledge of agent S and the adversary's fresh terms via the partial map $\varphi : \mathcal{C}_{\text{fresh}} \to \mathcal{C}_{\text{fresh}}$.

2. For every setup rule g_i containing an $\mathsf{AgentState}(S, c, n)$ fact for some $c, n \in \mathcal{M}$ we make the following two insertions.

   Insertion 1. For every fact $\mathsf{Fresh}(S, y) \in l(g_i)$ there are unique rules $g_k = [\,]\!-\![\,]\!\rightarrow\![\mathsf{Fr}(y)]$ and $g_j = [\mathsf{Fr}(y)]\!-\![\,]\!\rightarrow\![\mathsf{Fresh}(S, y)]$, $k < j < i$, producing $\mathsf{Fresh}(S, y)$.

   We insert an instantiation of the $\mathcal{FR}$ rule $[\,]\!-\![\,]\!\rightarrow\![\mathsf{Fr}(x)]$ immediately after g_k and an instantiation of the $\mathcal{MD}$ rule $[\mathsf{Fr}(x)]\!-\![\,]\!\rightarrow\![!\mathsf{K}(x)]$ immediately after g_j. We set $\varphi(y) := x$.

   Insertion 2. For every public constant $C : pub$ in S's knowledge n, we insert a rule $[\,]\!-\![\,]\!\rightarrow\![!\mathsf{K}(C : pub)]$ before g_i.

   After these insertions, we have a correspondence between S's initial knowledge and the adversary's knowledge. The modified sequence of rules remains a valid sequence.

3. Let g_i be the first instantiation of a role specification rule in g that contains an $\mathsf{AgentState}(S, c, n)$ fact for some $c, n \in \mathcal{M}$. By **P2** through **P4** and our hypothesis, we have only $\mathsf{Fresh}(S, \_)$, $\mathsf{Rcv_A}(\_, S, \_)$, and $\mathsf{AgentState}(S, \_, \_)$ facts in $l(g_i)$, $\mathsf{Snd_C}(S, \_, \_)$ and $\mathsf{AgentState}(S, \_, \_)$ facts in $r(g_i)$, and $\mathsf{Learn}(S, \_)$, $\mathsf{Comm}(S, \_)$, $\mathsf{Secret}(S, \_, \_)$, and $\mathsf{Authentic}(\_, S, \_)$ facts in $a(g_i)$. We delete the rule g_i after having made the following changes.

   Change 1. For every $\mathsf{Fresh}(S, x)$ fact in $l(g_i)$, there exists a rule $g_j = \ [\mathsf{Fr}(x)]\text{-}[\ ]\mapsto[\mathsf{Fresh}(S, x)]$, $j < i$, producing that fact. We replace g_j by the rule $[\mathsf{Fr}(x)]\text{-}[\ ]\mapsto[!\mathsf{K}(x)]$. Thus every fresh term learned by S in g is learned by the adversary in g'.

   Change 2. For every $\mathsf{Rcv_A}(R, S, m)$ fact we insert before g_i the rule $[\mathsf{Out}(\langle R, S, m\rangle)]\text{-}[\ ]\mapsto[!\mathsf{K}(\langle R, S, m\rangle)]$, which is an instantiation of $\mathcal{MD}$ Rule (3.3), and two instantiations of $\mathcal{MD}$ Rule (3.7) using the projecting functions in order to arrive at the facts $!\mathsf{K}(m)$, $!\mathsf{K}(R)$, $!\mathsf{K}(S)$. Note that there exists an $\mathsf{Out}(\langle R, S, m\rangle)$ fact in $r(g_j)$ for some $j < i$ due to instantiations of $\mathcal{CH}$ Rules (5.8) and (5.9) which are the source of the $\mathsf{Rcv_A}(R, S, m)$ fact.

   Thus every message received by S in g is learned by the adversary in g'.

   Change 3. By step 2 above (i.e., modifications of the setup rules), **P5**(c), and previous applications of the present step, all terms in $\mathsf{AgentState}(S, c, n) \in l(g_i)$ that are derivable from n, are also derivable from the adversary's knowledge up to substitution of fresh constants y in the domain of φ by $\varphi(y)$.

   Change 4. For each $\mathsf{Snd_C}(S, R, m)$ fact in $r(g_i)$, we can synthesize from the adversary's knowledge a message $\tilde{m}$ that is equal to m up to substitution of fresh constants y in the domain of φ by their image $\varphi(y)$. To this end, we insert after g_i instantiations of $\mathcal{MD}$ Rule (3.7) to produce the fact $!\mathsf{K}(\langle S, R, \tilde{m}\rangle)$. We delete the corresponding rule $g_j = \ [\mathsf{Snd_C}(S, R, m)]\text{-}[\ \mathsf{Snd_C}(S, R, m)\ ]\mapsto[!\mathsf{Conf}(R, m)]$, $j > i$, if it exists, and replace every subsequent rule $[!\mathsf{Conf}(R, m), \mathsf{In}(S)]\text{-}[\ \mathsf{Rcv_C}(S, R, m)\ ]\mapsto [\mathsf{Rcv_C}(S, R, m), !\mathsf{Conf}(R, m)]$ with $[!\mathsf{K}(\langle S, R, \tilde{m}\rangle)]\text{-}[\ !\mathsf{K}(\langle S, R, \tilde{m}\rangle)\ ]\mapsto[\mathsf{In}(\langle S, R, m\rangle)]$ and $[\mathsf{In}(\langle S, R, m\rangle)]\text{-}[\ ]\mapsto[\mathsf{Rcv_C}(S, R, m)]$. The latter of these rules is an instantiation of $\mathcal{CH}$ Rule (5.12) and the former is an incorrect instantiation of $\mathcal{MD}$ Rule (3.4). This is due to a mismatch between the adversary's knowledge $!\mathsf{K}(\langle S, R, \tilde{m}\rangle)$ and the produced fact $\mathsf{In}(\langle S, R, m\rangle)$. This is resolved in step 4 below.

   Change 5. Note that each $\mathsf{AgentState}$ fact in $r(g_i)$ is of the form $\mathsf{AgentState}(S, c, n)$, where the terms c and n are derivable from $!\mathsf{K}$ facts up to substitution of fresh constants in the domain of the φ function.

   Change 6. For each $\mathsf{Learn}(S, x)$ fact in $a(g_i)$, we insert instantiations of $\mathcal{MD}$ Rule (3.7) after g_i in order to arrive at $!\mathsf{K}(x)$ (up to substitutions of fresh constants in the domain of φ). This is possible, since x is a term derivable from public constants, messages in $\mathsf{Rcv_A}(R, S, m)$ facts and knowledge in $\mathsf{AgentState}(S, c, n)$ facts.

Change 7. We may ignore the Authentic and Secret facts in $a(g_i)$: We may ignore the Secret facts, since these indicate a confidentiality claim that we are not considering here. We may ignore Authentic facts occurring in an instance of a rule considered here, since these concern the authenticity of messages sent by R, as opposed to those sent by S. We may ignore the Learn facts because they label the transmission of messages from R rather than those from S.

We repeat step 3 as long as rules g_i containing $\mathsf{AgentState}(S, \_, \_)$ facts in $l(g_i)$ are there.

4. We exchange the fresh values y in the initial knowledge of S acquired in the setup rules with the corresponding fresh values $\varphi(y)$ in the adversary's knowledge ($!\mathsf{K}(\varphi(y))$) as follows.

   For every setup rule g_i containing an $\mathsf{AgentState}(S, c, n)$ fact and a $\mathsf{Fresh}(S, y)$ fact, we replace in all terms the fresh constant y by the fresh constant $\varphi(y)$. We replace the unique rule g_j, $j < i$, producing the fact $\mathsf{Fresh}(S, y)$ by the rule $[\mathsf{Fr}(\varphi(y))] -[\ ]\rightarrow [\mathsf{Fresh}(S, \varphi(y))]$.

   For every instantiation of a $\mathcal{MD}$ rule in g, we replace in all terms all fresh constants $\varphi(y)$ by y.

After the above replacements, we obtain a sequence of rules and consequently a trace tr' in which the adversary impersonates S. S does not perform any protocol steps other than having its initial knowledge set up. Especially, all rules where S sends a message are deleted and thus, all rules with a $\mathsf{Comm}(S, R, m)$ action are removed from tr'. Therefore, we have a trace where $\mathsf{Learn}(R, m) \in tr'$ and $\mathsf{Authentic}(S, R, m) \in tr'$, yet $\mathsf{Comm}(S, R, m) \notin tr'$. □

Using Lemmas 6.5 and 6.6 it is straightforward to prove that in a HISP topology where the trusted device cannot communicate to the user, the platform, or the server, no protocol exists that can provide a confidential or authentic channel.

Lemma 6.7 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology in which there are no outgoing edges from D. Then there exists no protocol providing a confidential channel and there exists no protocol providing an authentic channel between S and H in τ.*

Proof Since none of H, S, P receive any messages from D, a protocol that provides a confidential or authentic channel between H and S with such a role specification for D, also provides such a channel without a role specification for D. By Lemmas 6.5 and 6.6 no such protocol exists. □

We now prove our impossibility results. The first of the next three lemmas states that it is impossible to establish a secure channel between H and S if no edge exists between H and D.

Lemma 6.8 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology where no edge between H and D exists. Then there exists no protocol in τ providing a confidential channel and there exists no protocol in τ providing an authentic channel from H to S or vice-versa.*

The idea for the proof is that every trace establishing a confidential or authentic channel that involves actions of D, can be transformed into a valid trace with the same properties but not involving D. Since the channels between H and S are insecure, by Lemmas 6.5 and 6.6 neither confidential nor authentic channels can be established between H and S.

Proof Recall that the initial knowledge of H is empty. Since there is no edge between H and D, all communication channels to and from H are insecure.

Since there are no edges between H and D and all edges between D and P are labeled insecure as are the edges between S and P, we may include the D role in the S role while maintaining the property that all channels between S and P are labeled insecure. We thus obtain a protocol where all channels to and from S are insecure.

Thus the hypotheses of Lemmas 6.5 and 6.6 are satisfied and thus there is no protocol establishing a confidential or authentic channel between H and S. □

The following lemma states that it is impossible to establish a secure channel from H to S if there is neither and edge from D to H nor a path from D to S.

Lemma 6.9 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(H, D) \in E$, but $(D, H) \notin E$ and $(D, S) \notin E^+$. Then there exists no protocol providing a confidential channel and there exists no protocol providing an authentic channel from H to S in τ.*

Proof Since $(D, H) \notin E$ and $(D, S) \notin E^+$, we have $(D, S) \notin E$. We distinguish two cases, depending on whether the edge (D, P) exists.

- $(D, P) \notin E$. Then there are no outgoing edges from D and the statement follows from Lemma 6.7.
- $(D, P) \in E$. Then there is no edge from P to S, else there would be a path from D to S. It follows that there is no communication path from H to S, thus the protocol cannot provide a confidential nor an authentic channel from H to S. □

The next lemma states that it is impossible to establish a secure channel from S to H if there is no path from D to H.

Lemma 6.10 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(H, D) \in E$ and $(S, H) \in E^+$, but $(D, H) \notin E^+$. Then there exists no protocol providing a confidential and no protocol providing an authentic channel from S to H in τ.*

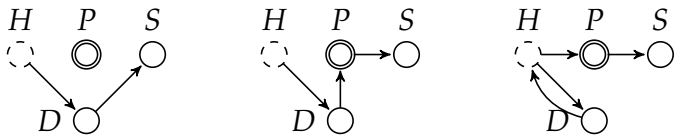
Proof Since there is no edge from D to H in τ, there are only two possible paths from S to H, namely (S, P, H) and (S, D, P, H). The second path, however, is impossible in τ, because it contains a path from D to H. It follows that there is no outgoing edge from D in τ thus by Lemma 6.7, there cannot be a protocol providing a confidential or an authentic channel from S to H in τ. □

Possibility Results

The following lemmas assert the existence of HISPs providing secure channels between H and S for the topologies not covered by the impossibility results above. Our proofs embody protocols that we have verified using Tamarin.

Lemma 6.11 *Let $\tau = (V, E, \eta, \mu)$ be any HISP topology with $(H, D) \in E$ and $(D, S) \in E^+$. Then there exists a protocol providing an originating secure channel from H to S in τ.*

Proof The following graphs consist of an acyclic path from D to S and an additional edge $(H, D) \in E$.



The following protocol communicates a message m, originating with H, authentically and confidentially from H to S using the path in case (a).

Protocol Lemma 6.11 (a)

$$H \bullet\!\!\rightarrow\!\!\bullet\ D : \mathrm{fresh}(m).\langle S, m\rangle$$
$$D \bullet\!\!\rightarrow\!\!\bullet\ S : \langle H, m\rangle$$

H first sends the fresh, secret message m together with the name of the intended recipient S to D using $H \bullet\!\!\rightarrow\!\!\bullet\ D$. Then, D passes the message and the sender's name H along to S using $D \bullet\!\!\rightarrow\!\!\bullet\ S$.

The following protocol transmits a message m, originating with H, authentically and confidentially from H to S using the path in case (b) and a secret key k shared between D and S. Recall that we specify the initial knowledge using knows$(_)$ statements.

Protocol Lemma 6.11 (b)

$$D : \mathrm{knows}(\langle S, k\rangle)$$
$$S : \mathrm{knows}(\langle D, k\rangle)$$
$$H \bullet\!\!\rightarrow\!\!\bullet\ D : \mathrm{fresh}(m).\langle S, m\rangle$$
$$D \circ\!\!\rightarrow\!\!\circ\ P : \mathrm{senc}(\langle H, m\rangle, k)/ciphertext$$
$$P \circ\!\!\rightarrow\!\!\circ\ S : ciphertext/\mathrm{senc}(\langle H, m\rangle, k)$$

The protocol runs as follows. H first sends the fresh, secret message m and the intended recipients's name S to D using $H \bullet\!\!\rightarrow\!\!\bullet\ D$. Then, D encrypts m and the sender's

name H using k and sends the cipher-text to P. P sends the cipher-text on to S where it is decrypted.

The following protocol transmits a message m, originating with H, authentically and confidentially from H to S using the path in case (c) and a secret key k shared between D and S.

Protocol Lemma 6.11 (c)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
H &\bullet\!\!\rightarrow\!\!\bullet\ D : \mathrm{fresh}(m).\langle S, m\rangle\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle m, \mathrm{senc}(m,k)\rangle / \langle m, ciphertext\rangle\\
H &\circ\!\!\rightarrow\!\!\circ\ P : ciphertext\\
P &\circ\!\!\rightarrow\!\!\circ\ S : ciphertext / \mathrm{senc}(m,k)
\end{aligned}
$$

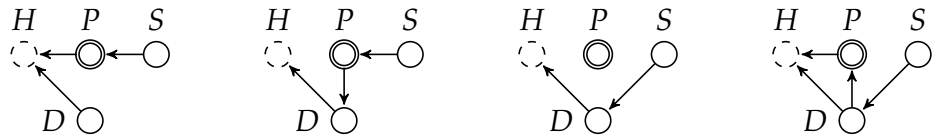
The protocol runs as follows. H first sends the fresh, secret message m and intended recipient's name S to D using $H \bullet\!\!\rightarrow\!\!\bullet\ D$. Then, D encrypts m using k and sends the cipher-text and the message back to H who inputs the cipher-text into P. P sends the cipher-text on to S where it is decrypted.

We used Tamarin to prove that these protocols provide an originating secure communication channel from H to S. □

Lemma 6.12 states that in HISP topologies that contain an edge from D to H, there exists a protocol providing a secure channel from H to S as long as there is a path from H to S.

Lemma 6.12 *Let (V, E, η, μ) be a HISP topology with $(S, H) \in E^+$. If $(D, H) \in E$ then there exists a protocol providing a secure channel from S to H.*

Proof The following are all acyclic paths from S to H together with an additional edge $(D, H) \in E$.



The protocol for case (a) is based on codebook cryptography, following [21]. It transmits a predefined message m securely from S to H.

Protocol Lemma 6.12 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, m, \mathrm{h}(m)\rangle)\\
S &: \mathrm{knows}(\langle H, D, m, \mathrm{h}(m)\rangle)\\
S &\circ\!\!\rightarrow\!\!\circ\ P : \mathrm{h}(m)/hash\\
P &\circ\!\!\rightarrow\!\!\circ\ H : hash\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle S, m, \mathrm{h}(m)\rangle/\langle S, m, hash\rangle
\end{aligned}
$$

The hash function $\mathrm{h}(m)$ represents the mapping, shared between D and S, from a clear-text message m to the code. After the protocol's execution, H compares the code supposedly received from S with the tuple $\langle m, \mathrm{h}(m)\rangle$ received from D. This represents a lookup in the codebook.

For case (b), the following protocol provides an originating secure communication channel from S to H using a secret key k shared between D and S.

Protocol Lemma 6.12 (b)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
S &\circ\!\!\rightarrow\!\!\circ\ P : \mathrm{fresh}(m).\mathrm{senc}(m, k)/ciphertext\\
P &\circ\!\!\rightarrow\!\!\circ\ D : ciphertext/\mathrm{senc}(m, k)\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle S, m\rangle
\end{aligned}
$$

S first submits the fresh, secret message m encrypted with the key k to P using $S \circ\!\!\rightarrow\!\!\circ\ P$. P passes the cipher-text on to D, who decrypts the message and sends m and its sender's name S to H using $D \bullet\!\!\rightarrow\!\!\bullet\ H$.

For case (c), the following protocol provides an originating secure communication channel from S to H using the secure links $S \bullet\!\!\rightarrow\!\!\bullet\ D$ and $D \bullet\!\!\rightarrow\!\!\bullet\ H$.

Protocol Lemma 6.12 (c)

$$
\begin{aligned}
S &\bullet\!\!\rightarrow\!\!\bullet\ D : \mathrm{fresh}(m).\langle H, m\rangle\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle S, m\rangle
\end{aligned}
$$

S first submits the fresh, secret message m together with the intended recipient's name H to D using $S \bullet\!\!\rightarrow\!\!\bullet\ D$. Then, D passes m together with the sender's name S to H using $D \bullet\!\!\rightarrow\!\!\bullet\ H$.

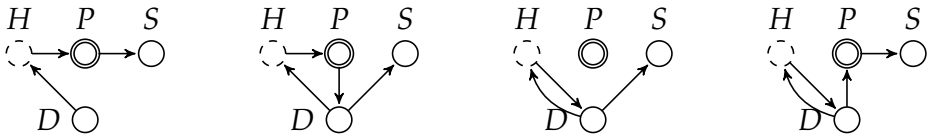
For case (d), the same protocol as for case (c) can be applied by omitting the additional edges $(D, P) \in E$ and $(P, H) \in E$.

We used Tamarin to prove that all three protocols above provide a secure communication channel from S to H. □

Lemma 6.13 states that in HISP topologies that contain an edge from D to H, there exists a protocol providing a secure channel from S to H, as long as there is a path from S to H.

Lemma 6.13 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(H, S) \in E^+$. If $(D, H) \in E$ then there exists a protocol providing a secure channel from H to S in τ.*

Proof The following are all acyclic paths from H to S together with an additional edge $(D, H) \in E$.



The following protocols each communicate a predefined message m authentically and confidentially from H to S via the paths in case (a) and (b), respectively. The hash function $\mathrm{h}(m)$ represents the mapping from a clear-text message m to the code. At the end of Protocol 6.13 (a), S compares the code supposedly received from H with the corresponding tuple $\langle m, \mathrm{h}(m)\rangle$.

Protocol Lemma 6.13 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, m, \mathrm{h}(m)\rangle)\\
S &: \mathrm{knows}(\langle H, D, m, \mathrm{h}(m)\rangle)\\
D \bullet\!\!\rightarrow\bullet\ H &: \langle S, m, \mathrm{h}(m)\rangle \ /\ \langle S, m, \mathit{hash}\rangle\\
H \circ\!\!\rightarrow\!\!\circ\ P &: \mathit{hash}\\
P \circ\!\!\rightarrow\!\!\circ\ S &: \mathit{hash} \ /\ \mathrm{h}(m)
\end{aligned}
$$

Protocol Lemma 6.13 (b)

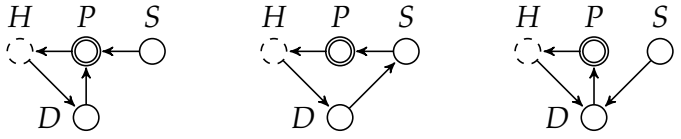
$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S\rangle)\\
D \bullet\!\!\rightarrow\bullet\ H &: \mathrm{fresh}(m).\langle m, \mathrm{h}(m)\rangle \ /\ \langle m, \mathit{hash}\rangle\\
H \circ\!\!\rightarrow\!\!\circ\ P &: \mathit{hash}\\
P \circ\!\!\rightarrow\!\!\circ\ D &: \mathit{hash} \ /\ \mathrm{h}(m)\\
D \bullet\!\!\rightarrow\bullet\ S &: \langle H, m\rangle
\end{aligned}
$$

We use Tamarin to prove that both protocols provide a secure communication channel from H to S.

Cases (c) and (d) follow from Lemma 6.11. □

Lemma 6.14 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(S, H) \in E^+$ and $(D, H) \notin E$. If $(H, D) \in E$ and $(D, H) \in E^+$, then there exists a protocol providing an originating authentic channel from S to H in τ.*

Proof The minimal graphs satisfying the lemma's hypothesis are obtained as follows. There are two acyclic paths from S to H with $(D, H) \notin E$. One satisfies $(D, H) \in E^+$ and leads to case (c). The other leads to cases (a) and (b), since there are two acyclic paths form D to H with $(D, H) \notin E$.



The following protocols provide an originating authentic channel from S to H.

Protocol Lemma 6.14 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
S \circ\!\!\rightarrow\!\!\circ\ P &: \mathrm{fresh}(m).\langle m, \mathrm{h}(\langle k, m\rangle)\rangle \ /\ \langle m, hash\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ H &: \langle m, hash\rangle\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x).\langle S, x, m, hash\rangle \ /\ \langle S, x, m, \mathrm{h}(\langle k, m\rangle)\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: x\\
P \circ\!\!\rightarrow\!\!\circ\ H &: x
\end{aligned}
$$

Protocol Lemma 6.14 (b)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
S \circ\!\!\rightarrow\!\!\circ\ P &: \mathrm{fresh}(m).\langle m, \mathrm{h}(\langle k, m\rangle)\rangle \ /\ \langle m, hash\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ H &: \langle m, hash\rangle\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x).\langle x, m, hash\rangle \ /\ \langle x, m, \mathrm{h}(\langle k, m\rangle)\rangle\\
D \bullet\!\!\rightarrow\!\!\bullet\ S &: x\\
S \circ\!\!\rightarrow\!\!\circ\ P &: x\\
P \circ\!\!\rightarrow\!\!\circ\ H &: x
\end{aligned}
$$

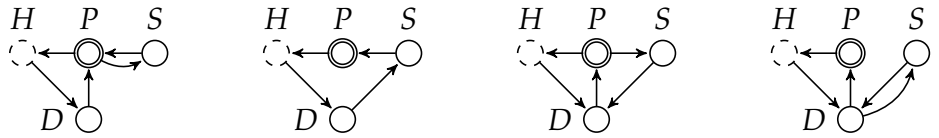
Protocol Lemma 6.14 (c)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S\rangle)\\
S &: \mathrm{knows}(\langle H, D\rangle)\\
S \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(m).m\\
D \circ\!\!\rightarrow\!\!\circ\ P &: m\\
P \circ\!\!\rightarrow\!\!\circ\ H &: m\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x).\langle S, x, m\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: x\\
P \circ\!\!\rightarrow\!\!\circ\ H &: x
\end{aligned}
$$

In each of the protocols, S generates a fresh message m, which is communicated to H. We use Tamarin to prove that these protocols provide an originating authentic communication channel from S to H. □

Lemma 6.15 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(S, H) \in E^+$, $(D, H) \notin E$, $(H, D) \in E$, and $(D, H) \in E^+$. Then there exists a protocol providing a secure channel from S to H in τ if and only if $(H, S) \in E^+$.*

Proof The minimal graphs satisfying the lemma's hypothesis are obtained from the graphs of Lemma 6.14 and the additional condition $(H, S) \in E^+$.



(a) (b) (c) (d)

For each of the minimal graphs, one of the following protocols provides a secure channel from S to H.

Protocol Lemma 6.15 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
S &: \mathrm{knows}(\langle H, D, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x_1, x_2).\langle S, x_1, x_2\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: \langle S, \mathrm{senc}(\langle x_1, x_2\rangle, \mathrm{h}(\langle k, D, S\rangle))\rangle \ /\ \langle S, ciphertext\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ S &: ciphertext \ /\ \mathrm{senc}(\langle x_1, x_2\rangle, \mathrm{h}(\langle k, D, S\rangle))\\
S \circ\!\!\rightarrow\!\!\circ\ P &: x_2\\
P \circ\!\!\rightarrow\!\!\circ\ H &: x_2
\end{aligned}
$$

Protocol Lemma 6.15 (b)

$$
\begin{aligned}
D &: \langle H, S\rangle\\
S &: \langle H, D\rangle\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x_1, x_2).\langle S, x_1, x_2\rangle\\
D \bullet\!\!\rightarrow\!\!\bullet\ S &: \langle H, x_1, x_2\rangle\\
S \circ\!\!\rightarrow\!\!\circ\ P &: \langle x_2\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ H &: \langle x_2\rangle
\end{aligned}
$$

Protocol Lemma 6.15 (c)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
S &: \mathrm{knows}(\langle H, D, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x_1, x_2).\langle S, x_1, x_2\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: \langle S, \mathrm{senc}(\langle x_1, x_2\rangle, \mathrm{h}(\langle k, D, S\rangle))\rangle \ /\ \langle S, ciphertext\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ S &: \langle ciphertext\rangle \ /\ \langle \mathrm{senc}(\langle x_1, x_2\rangle, \mathrm{h}(\langle k, D, S\rangle))\rangle\\
S \bullet\!\!\rightarrow\!\!\bullet\ D &: \langle H, x_1\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: \langle x_2\rangle\\
P \circ\!\!\rightarrow\!\!\circ\ H &: \langle x_2\rangle
\end{aligned}
$$

Protocol Lemma 6.15 (d)

$$
\begin{aligned}
D &: \langle H, S\rangle\\
S &: \langle H, D\rangle\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \mathrm{fresh}(x_1, x_2).\langle S, x_1, x_2\rangle\\
D \bullet\!\!\rightarrow\!\!\bullet\ S &: \langle H, x_1, x_2\rangle\\
S \bullet\!\!\rightarrow\!\!\bullet\ D &: \langle H, x_1\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: x_2\\
P \circ\!\!\rightarrow\!\!\circ\ H &: x_2
\end{aligned}
$$

We used Tamarin to prove that these protocols provide a secure communication channel from S to H.

To see that $(H, S) \in E^+$ is necessary, suppose that $(H, S) \notin E^+$. Recall that the initial knowledge of H is empty. Any fresh constant that H generates cannot be known to S because $(H, S) \notin E^+$. Any message that H receives is known to P because the only

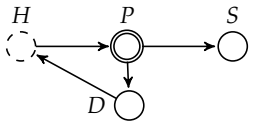


Figure 6.3: Example of a HISP topology. Edge labels omitted.

incoming edge to H is $(P, H) \in E$. Thus every message sent by S and learned by H can be learned by the adversary. It follows that every term t that can be derived from the knowledge of H using pairing and projection and that can be derived from the knowledge of S (using all functions in Σ), can also be derived using the knowledge of P. Thus there cannot be a protocol providing a confidential channel and consequently there cannot be a protocol providing a secure channel from S to H. □

6.4 Complete Characterization of HISPs Providing Originating Secure Channels

In this section we classify all HISP topologies for which protocols exist that provide an originating secure communication channel. These are protocols permitting the communication partners to securely exchange arbitrary messages. We consider the general case of HISPs that provide secure communication channels in Section 6.3. We first give four theorems that state the necessary and sufficient conditions to establish originating authentic and originating confidential channels between the human user and the remote server. The characterization of HISPs providing originating secure communication channels then follows as two corollaries.

As one might expect, an originating confidential channel from the human user to the server can only be provided if the human can input a message into the supporting device. This is because the computing platform is considered to be under the adversary's control. Similarly, in the reverse direction such a channel can only be provided if the human user can receive messages from the supporting device. Perhaps surprisingly however, the possibilities for originating authentic channels are less restricted than for originating confidential channels. This difference is due to the human user's limitations. The user's ability to generate fresh messages and compare previously sent messages with received messages suffices to guarantee originating authenticity in certain HISP topologies, but it is insufficient for originating confidentiality. The following example illustrates this difference.

Example 6.16 *Let $\tau = (V, E, \eta, \mu)$ be the HISP topology for the following scenario. A human user has a device which contains a small display and shares a symmetric key with a remote server. This is represented by $(D, H) \in E$ in τ. The device is connected to the user's computer receiving input, i.e., $(P, D) \in E$. The user sends messages to the server through the computer, i.e., $(H, P) \in E$ and $(P, S) \in E$. The HISP topology τ is shown in Figure 6.3. There is a protocol in this HISP topology that provides an originating authentic, but not confidential,*

channel from the human user to the remote server. The protocol proceeds as follows. The user inputs his message into the computer which sends it to the device. The device displays the message along with a message authentication code to the user. The user inputs the code into the computer which in turn sends the message along with the code to the remote server. The protocol is shown in the extended Alice & Bob notation below. The correctness of the originating authenticity claim is shown in Lemma 6.18. By Theorems 6.19 and 6.21 below, there is no protocol in this topology that provides a confidential channel in either direction.

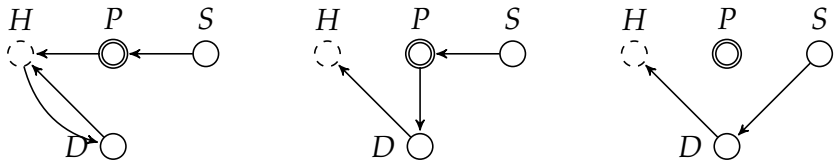
Protocol Example 6.16

$$
\begin{aligned}
D &: \text{knows}(\langle H, S, k\rangle)\\
S &: \text{knows}(\langle H, D, k\rangle)\\
H \circ\!\!\rightarrow\!\!\circ\ P &: \text{fresh}(m).m\\
P \circ\!\!\rightarrow\!\!\circ\ D &: m\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: \langle m, h(\langle k, m\rangle)\rangle \ /\ \langle m, hash\rangle\\
H \circ\!\!\rightarrow\!\!\circ\ P &: hash\\
P \circ\!\!\rightarrow\!\!\circ\ S &: \langle m, hash\rangle \ /\ \langle m, h(\langle k, m\rangle)\rangle
\end{aligned}
$$

The following two lemmas are used to prove Theorems 6.20, 6.21, and 6.22. The next lemma states that a path from S to D and an edge from D are sufficient for originating secure communication from S to H.

Lemma 6.17 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(S, D) \in E^+$ and $(D, H) \in E$. Then there exists a protocol providing an originating secure channel from S to H in τ.*

Proof Below are all acyclic paths from S to D together with an additional edge $(D, H) \in E$.



Case (c) is equal to case (c) in Lemma 6.12 where the given protocol already provides an originating secure channel from S to H. In the following we provide protocols for the remaining cases (a) and (b).

The following protocol provides an originating secure channel from S to H in case (a).

Protocol Lemma 6.17 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
S &: \mathrm{knows}(\langle H, D, \mathrm{h}(\langle k, D, S\rangle)\rangle)\\
S \circ\!\!\rightarrow\!\!\circ\ P &: \mathrm{fresh}(m).\mathrm{senc}(\langle S, D, H, m\rangle, \mathrm{h}(\langle k, D, S\rangle))\ /\\
&\quad\ ciphertext\\
P \circ\!\!\rightarrow\!\!\circ\ H &: ciphertext\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: ciphertext\ /\ \mathrm{senc}(\langle S, D, H, m\rangle, \mathrm{h}(\langle k, D, S\rangle))\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: \langle S, \mathrm{senc}(\langle S, D, H, m\rangle, \mathrm{h}(\langle k, D, S\rangle)), m\rangle\ /\\
&\quad\ \langle S, ciphertext, m\rangle
\end{aligned}
$$

For case (b), we adapt the protocol as follows.

Protocol Lemma 6.17 (b)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, \mathrm{h}(\langle k, D, H, S\rangle)\rangle)\\
S &: \mathrm{knows}(\langle H, D, \mathrm{h}(\langle k, D, H, S\rangle)\rangle)\\
S \circ\!\!\rightarrow\!\!\circ\ P &: \mathrm{fresh}(m).\mathrm{senc}(\langle S, D, H, m\rangle, \mathrm{h}(\langle k, D, H, S\rangle))\ /\\
&\quad\ ciphertext\\
P \circ\!\!\rightarrow\!\!\circ\ D &: ciphertext\ /\ \mathrm{senc}(\langle S, D, H, m\rangle, \mathrm{h}(\langle k, D, H, S\rangle))\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: \langle S, m\rangle
\end{aligned}
$$

In both cases, S first freshly generates the secret message m and sends it encrypted with the key k to P using $S \circ\!\!\rightarrow\!\!\circ\ P$. P passes the message on to H in case (a) or directly to D in case (b). The message is decrypted by D and sent to H using $D \bullet\!\!\rightarrow\!\!\bullet\ H$.

We use Tamarin to prove that these protocols provide a secure communication channel from S to H. □

The next lemma states that if an edge from D to H but not from H to D and a path from H to S exists, then an additional incoming edge to D is sufficient to establish originating authentic communication from H to S.

Lemma 6.18 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology with $(D, H) \in E$, $(H, D) \notin E$, and $(H, S) \in E^+$. If there is an incoming edge to D, then there exists a protocol providing an originating authentic channel from H to S in τ.*

Proof There must be an edge $(H, P) \in E$ because $(D, H) \in E$, $(H, D) \notin E$, and $(H, S) \in E^+$. Since D has an incoming edge, it must have either an incoming edge from P or one from S. The first of the following two protocols provides an originating authentic channel in the former case, and the second in the latter case.

Protocol Lemma 6.18 (a)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
H &\circ\!\!\rightarrow\!\!\circ\ P : \mathrm{fresh}(m).m\\
P &\circ\!\!\rightarrow\!\!\circ\ D : m\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle m, \mathrm{h}(\langle k, m, S, D, H\rangle)\rangle \ /\ \langle m, hash\rangle\\
H &\circ\!\!\rightarrow\!\!\circ\ P : hash\\
P &\circ\!\!\rightarrow\!\!\circ\ S : \langle m, hash\rangle \ /\ \langle m, \mathrm{h}(\langle k, m, S, D, H\rangle)\rangle
\end{aligned}
$$

Protocol Lemma 6.18 (b)

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, k\rangle)\\
S &: \mathrm{knows}(\langle H, D, k\rangle)\\
H &\circ\!\!\rightarrow\!\!\circ\ P : \mathrm{fresh}(m).m\\
P &\circ\!\!\rightarrow\!\!\circ\ S : m\\
S &\bullet\!\!\rightarrow\!\!\bullet\ D : \langle m, \mathrm{h}(\langle k, m\rangle)\rangle\\
D &\bullet\!\!\rightarrow\!\!\bullet\ H : \langle m, S, \mathrm{h}(\langle k, m\rangle)\rangle \ /\ \langle m, S, hash\rangle\\
H &\circ\!\!\rightarrow\!\!\circ\ P : hash\\
P &\circ\!\!\rightarrow\!\!\circ\ S : hash \ /\ \mathrm{h}(\langle k, m\rangle)
\end{aligned}
$$

The following theorem states that a human can send a confidential message to a server if and only if the human can input the message into a trusted device and there is a communication path from the trusted device to the server.

Theorem 6.19 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. There exists a protocol providing an originating confidential channel from H to S in τ if and only if $(H, D) \in E$ and $(D, S) \in E^+$.*

Proof Let τ be a HISP topology such that $(H, D) \in E$ and $(D, S) \in E^+$. By Lemma 6.11, there exists a protocol providing an originating confidential channel H to S in τ for each of the three acyclic paths from D to S.

Conversely, let $\mathcal{R}$ be a protocol providing a confidential channel H to S in τ. Then there is a trace in which a fresh constant m originating with H is transmitted to S. Thus there must be a path from H to S. Suppose $(H, D) \notin E$. Then the only outgoing edge from H is $(H, P) \in E$. Since H can only perform pairing and projection, any fresh constant m generated by H can only be paired with other terms. Thus, if H sends a message of which m is a subterm, the adversary can learn m. Thus there must be an edge from H to D. By Lemma 6.9, there must furthermore be a path from D to S. □

The following theorem shows that the conditions for a human to send an authentic message to a server are weaker than the conditions for confidential messages.

Theorem 6.20 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. Then there exists a protocol providing an originating authentic channel from H to S in τ if and only if $(H, S) \in E^+$, there exists an edge between H and D, and there exists an edge incoming to D as well as an edge outgoing from D.*

Proof It is obvious that $(H, S) \in E^+$ is a necessary condition for a protocol to provide an authentic channel from H to S. If there is no edge between D and H, then by Lemma 6.6, there is no protocol providing an authentic channel. Similarly, if all edges adjacent to D are incoming to D, then there is no protocol providing an authentic channel, as there would be one without a role specification for D, which is impossible by Lemma 6.6. Finally, if all edges adjacent to D are outgoing from D, then D never learns any fresh constant m generated by H. Thus m is never a subterm of any message sent from D to H. Thus for every message m' sent from H to P, the adversary may compute all projections of m' and substitute each m by a fresh constant $\tilde{m}$, then pair the terms up again. For all messages received by H from P, the adversary replaces in the same manner all projections to $\tilde{m}$ by m. Thus S learns $\tilde{m}$ whereas H sends m. Since m originates with H, S cannot distinguish between terms involving m and terms involving $\tilde{m}$. Since H cannot perform any functions other than pairing and projections, H cannot distinguish terms that are obtained by applying any other function to m from terms that are obtained by applying such functions to $\tilde{m}$.

Conversely, consider all the HISP topologies such that $(H, S) \in E^+$ and there exists an edge between H and D and there exists an edge incoming to D as well as an edge outgoing from D. There are two types of protocols providing an originating authentic channel from H to S, depending on the edge(s) between H and D.

- $(H, D) \in E$. Since there is a path $(H, S) \in E^+$ and an outgoing edge from D, there must be a path $(D, S) \in E^+$. It follows from Lemma 6.11 that there exists a protocol providing an originating authentic channel from H to S.
- $(D, H) \in E$ and $(H, D) \notin E$. Then there exists a protocol providing an originating authentic channel from H to S by Lemma 6.18. □

We have analogous theorems for originating confidential and originating authentic channels from a server to a human.

Theorem 6.21 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. There exists a protocol providing an originating confidential channel from S to H in τ if and only if $(D, H) \in E$ and $(S, D) \in E^+$.*

Proof Let τ be a HISP topology such that $(D, H) \in E$ and $(S, D) \in E^+$. By Lemma 6.17, there is a protocol providing an originating confidential channel S to H in τ for each of the three acyclic paths from S to D.

Conversely, let $\mathcal{R}$ be a protocol providing a confidential channel S to H in τ. Then there is a trace in which a fresh constant m originating with S is transmitted to H. Thus there must be a path from S to H. Suppose $(D, H) \notin E$. Then the only incoming edge to H is $(P, H) \in E$. Since H can only perform pairing and projection, any fresh constant m learned, but not generated by H can only be learned as a singleton or paired with other terms. Thus, if H receives a message of which m is a subterm and H learns m, then the adversary can learn m. Thus there must be an edge from D to H. Suppose now that there is no path $(S, D) \in E^+$. Then there are only outgoing edges from D, because there is a path S to H and an edge (D, H). Thus m is not in D's

knowledge, since it originates with S and there is no communication path from S to D. Thus, as above, since H can only perform pairing and projecting of terms, any fresh constant m learned by H and generated by S can be learned by the adversary. Thus there must be a path $(S, D) \in E^+$. □

Theorem 6.22 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. Then there exists a protocol providing an originating authentic channel from S to H in τ if and only if $(S, H) \in E^+$, there exists an edge between H and D, and there exists an edge incoming to D as well as an edge outgoing from D.*

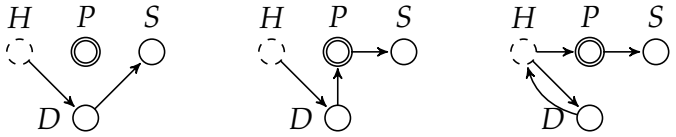
Proof It is obvious that $(S, H) \in E^+$ is a necessary condition for a protocol to provide an authentic channel from H to S. If there is no edge between D and H, then by Lemma 6.6, there is no protocol providing an authentic channel. Similarly, if all edges adjacent to D are incoming to D, then there is no protocol providing an authentic channel, otherwise there would be one without a role specification for D which is impossible by Lemma 6.6. Finally, if all edges adjacent to D are outgoing from D, then D never learns any fresh constant m generated by S. Thus m is never a subterm of any message sent from D to H. Thus for every message m' sent from S to P, the adversary may compute all projections of m' and substitute each m by a fresh constant $\tilde{m}$, then pair the terms up again. For all messages sent by H to P, the adversary replaces in the same manner all projections to $\tilde{m}$ by m. Thus H learns $\tilde{m}$ whereas S sends m. Since m originates with S, H cannot distinguish between terms involving m and terms involving $\tilde{m}$. Since H cannot perform any functions other than pairing and projections, H cannot distinguish terms that are obtained by applying any other function to m from terms that are obtained by applying such functions to $\tilde{m}$.

Conversely, consider all the HISP topologies such that $(S, H) \in E^+$ and there exists an edge between H and D and there exists an edge incoming to D as well as an edge outgoing from D. There are two types of protocols providing an originating authentic channel from S to H, depending on the edge(s) between H and D.

- $(D, H) \in E$. Since there is a path $(S, H) \in E^+$ and an incoming edge to D, there must be a path $(S, D) \in E^+$. It follows from Lemma 6.17 that there exists a protocol providing an originating authentic channel from S to H.
- $(H, D) \in E$ and $(D, H) \notin E$. Then there must be a path $(D, H) \in E^+$, since there is an outgoing edge from D. By Lemma 6.14, all such HISP topologies provide an originating authentic channel from S to H. □

By combining Theorems 6.19 and 6.20 we see that the topology of any HISP providing an originating secure channel from H to S is a supergraph of one of the graphs shown in Corollary 6.23.

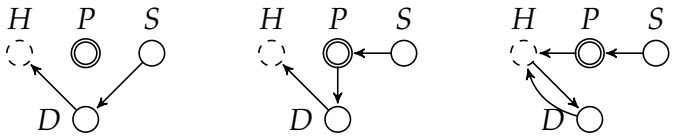
Corollary 6.23 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. There exists a protocol providing an originating secure channel from H to S in τ if and only if $(H, D) \in E$ and $(D, S) \in E^+$. The following are all minimal graphs satisfying these conditions.*



(a) (b) (c)

Theorems 6.21 and 6.22 imply Corollary 6.24. It states that the topology of any HISP providing an originating secure channel from S to H is a supergraph of one of the three graphs.

Corollary 6.24 *Let $\tau = (V, E, \eta, \mu)$ be a HISP topology. There exists a protocol providing an originating secure channel from S to H in τ if and only if $(D, H) \in E$ and $(S, D) \in E^+$. The following are all minimal graphs satisfying these conditions.*



(a) (b) (c)

7 Formal Development and Analysis of Secure HISPs

Our characterization and design methodology are not only relevant for Internet voting but for other practical applications, such as online banking, in general. To demonstrate their applicability, we provide two case studies in Section 7.2 including protocols for code voting and transaction authentication. These demonstrate how our classification guides us to appropriate protocol setups and how we can use Tamarin to verify the resulting protocols. We then apply our model and the methodology to analyze a reference Internet voting protocol suggested by the Swiss Federal Chancellery in Section 7.3.

7.1 Methodology

The characterization we provide in Chapter 6 can be used to guide the design of novel solutions for establishing secure channels between a human and a remote server. The developer uses our characterization to first analyze whether the proposed solution satisfies the necessary conditions for secure communication. This step concludes with a minimal communication topology. After a communication protocol is designed, the developer verifies that the protocol indeed employs all communication links appearing in the minimal communication topology. The protocol is then formally specified in our specification language in order to analyze the protocol's security properties with the Tamarin [84] verification tool. This methodology supports the verification of a large class of distributed algorithms running on nodes communicating over links. In particular, it supports the automatic verification of confidentiality and authenticity of information exchanged between nodes in security protocols involving humans and their insecure platforms.

7.2 Case Studies

In the following we present two case studies where we demonstrate how our characterization may guide the design of HISPs following the methodology sketched in the introduction. In the first case study, we analyze an Internet voting protocol based on

codebook cryptography. The second case study concerns transaction authentication in online banking.

Development of Secure HISPs

In codebook cryptography, a codebook is exchanged prior to communication. This codebook assigns random codes to clear-text messages. The sender chooses a code that is assigned to the clear-text message he wants to communicate and sends the code instead of the clear-text. Hence, an adversary overhearing the communication does not learn the clear-text and, without the codebook, he cannot replace the code with another valid one that will be accepted by the receiver.

To realize a code voting scheme where a voter may communicate a confidential vote authentically to the remote server, we apply Theorem 6.3. Of the four minimal communication topologies shown in the theorem, we choose the unique topology (c) in which the supporting device neither communicates with the platform nor with the remote server. We then must design a new code voting protocol or choose an existing one. We opt for the latter.

Simple code voting employs codebook cryptography. It works as follows. Prior to the election, the voter receives a code-sheet containing the candidate names and a corresponding code for each of them. Since the code-sheets are distributed by postal mail, it is assumed that the network adversary does not know them. To communicate a vote, the voter chooses a candidate and enters the corresponding code into his untrusted computer. This code is then submitted to the election authority's server. Since the election authority created the code-sheets, it can map the code back to the corresponding candidate.

HISP topology. Simple code voting is represented by the HISP topology (V, E, η, μ) shown in Figure 7.1. The voter H's dishonest computer P is used to submit a ballot, i.e., a candidate choice, to the election authority's server S. The pre-distributed code-sheet is modeled by D. The edge $(D, H) \in E$ models the voter's ability to read information

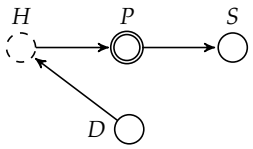


Figure 7.1: Simple code voting topology.

from the code-sheet. This communication is considered to be secure, that is, we assume that the voter reads the code-sheet in a private environment. The HISP topology is equal to the minimal HISP topology (c) in Theorem 6.3. Thus, we conclude that the simple code voting's topology is a valid option to securely communicate a vote from a human to a server. Next, we analyze the simple code voting's protocol specification.

Protocol specification. In the simple code voting protocol, the voter H possesses a personal code-sheet represented as D. The latter contains the candidate names and cor-

responding codes, bound to H and S. The election server S is initialized to know the distributed code-sheet D, the candidate names and the corresponding codes as well as H to whom the code-sheet was distributed. Voter H first reads the tuple $\langle c, \mathrm{h}(c)\rangle$ from the code-sheet, where c represents the desired candidate and $\mathrm{h}(c)$ the corresponding code. Using his dishonest computer P, H submits $\mathrm{h}(c)$ to S. The election authority maps $\mathrm{h}(c)$ back to c and counts one additional vote for the corresponding candidate. The protocol is specified as follows.

Protocol Simple code voting

$$
\begin{aligned}
D &: \mathrm{knows}(\langle H, S, c, \mathrm{h}(c)\rangle)\\
S &: \mathrm{knows}(\langle H, D, c, \mathrm{h}(c)\rangle)\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: \langle S, c, \mathrm{h}(c)\rangle \ /\ \langle S, c, code\rangle\\
H \circ\!\!\rightarrow\!\!\circ\ P &: code\\
P \circ\!\!\rightarrow\!\!\circ\ S &: code \ /\ \mathrm{h}(c)
\end{aligned}
$$

We easily verify that the protocol employs all links shown in Figure 7.1. Next, we use Tamarin to show that the protocol provides a secure channel from H to S in the above topology.

Verification results. Tamarin confirms that the simple code voting protocol as specified above satisfies confidentiality and authenticity for the term c which represents the chosen candidate.

This simple code voting case study illustrates our methodology to guide the design of a HISP. We have shown how to use our characterization to select a minimal HISP topology, choose a security protocol candidate and automatically verify the candidate protocol's security properties.

Smart-card-based Transaction Authentication

Online banking is an important application domain of HISPs. Some banks offer their customers trusted hardware to authenticate themselves to the bank's server over the Internet. A few of these banks additionally use this hardware to authenticate individual transactions. In this case study, we analyze a simple protocol for transaction authentication.

The desired security goal in this case is an originating authentic channel from the customer H to the banking server S, to communicate transaction instructions. Hence, we apply Theorem 6.20. Of the three minimal communication topologies shown in Corollary 6.23, we choose the topology (b) in which the supporting device communicates with the platform. We now need to design a transaction-authentication protocol and the corresponding supporting technology.

The transaction authentication works as follows. To secure the communication, the bank provides its customers smart cards containing private signing keys. Smart card access is protected by a personal identification number (PIN) only known to the customer and the card itself. Note that this is a stronger assumption than the one

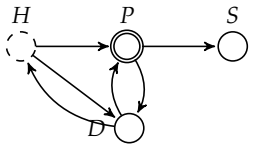


Figure 7.2: Smart-card-based transaction authentication topology.

made in Section 6.4. Furthermore, the bank provides smart card readers with small displays and key pads. The reader is connected to the customer's computer. To tell the bank to execute a transaction, the customer enters the transaction instruction into his untrusted computer. The transaction instruction is passed to the smart card reader and presented to the customer using the integrated display. After verification of its correctness, the customer confirms the transaction instruction by entering the PIN into the smart card reader. The smart card signs the transaction instruction using the contained signing key and passes this signature to the customer's computer. Finally, the computer submits the transaction information together with the digital signature to the bank's server. If the verification of the digital signature is successful, the bank performs the transaction. The HISP topology, the protocol specification, and the verification are discussed in detail below.

HISP topology. The HISP topology (V, E, η, μ) is depicted in Figure 7.2. The customer H uses his dishonest computer P to access the bank's server S. The smart card together with the reader are modeled by D. The edges $(H, D) \in E$ and $(D, H) \in E$ represent the customer's ability to enter information into the reader using the key pad and to read from the reader's display, respectively. This communication is considered to be secure, that is, we assume that the customer interacts with the reader in a private environment. The HISP topology is a supergraph of the minimal HISP topologies (b) and (c) in Corollary 6.23. By Theorem 6.20, we may conclude that this topology is a valid option to authentically communicate transaction instructions from a human to a server. Next, we analyze a possible protocol specification.

Protocol specification. The following protocol aims to provide authenticity. As we shall see, this protocol requires additional assumptions on the smart card reader's implementation to prevent race conditions. Customer H owns a personal smart card and the reader represented by D. Additionally, he knows a PIN to access the smart card. The smart card contains the private signing key and is unlocked with the PIN that the customer knows, thereby binding the signing key to the customer H. Thus D knows the PIN and the private signing key $ltkD$. For each distributed device D, the server knows the corresponding public key and the customer it was distributed to.

The customer starts the protocol with a transaction instruction m that he enters into his personal computer P. Afterwards, P sends m to the smart card (reader) D. The smart card (reader) in turn displays m to H using its integrated display. After verifying that the displayed m matches the intended instruction, H enters the PIN into D to unlock the smart card, which then signs m using its signing key and sends the

signed transaction information back to P. P finally sends m together with the digital signature to the bank's server S, which accepts m provided that it is signed with the smart card's signing key. The protocol is specified as follows.

Protocol Transauth a

$$
\begin{aligned}
D &: \text{knows}(\langle H, PIN, ltkD\rangle)\\
S &: \text{knows}(\langle H, D, \text{pk}(ltkD)\rangle)\\
H &: \text{knows}(\langle D, PIN\rangle)\\
H \circ\!\!\rightarrow\!\!\circ\ P &: \text{fresh}(m).m\\
P \circ\!\!\rightarrow\!\!\circ\ D &: m\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: m\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: PIN\\
D \circ\!\!\rightarrow\!\!\circ\ P &: \{m\}_{ltkD}\ /\ signature\\
P \circ\!\!\rightarrow\!\!\circ\ S &: \langle m, signature\rangle\ /\ \langle m, \{m\}_{ltkD}\rangle
\end{aligned}
$$

We easily verify that the protocol employs all links shown in Figure 7.2. Next, we use Tamarin to analyze whether the protocol establishes an originating authentic channel from H to S.

Verification results. Verification of originating authenticity fails and Tamarin provides a counterexample where P follows the protocol until H verifies the information on D's display. Before H enters the PIN into the smart card reader, the adversary restarts D with a new run by sending another message to the reader. When H enters the PIN, the adversary's message is signed and sent to S and hence the authenticity of the transaction information fails. Although the reader's display shows the adversary's information to H, the customer's next step in the protocol is to enter the PIN. Doing so causes D to sign an unintended transaction instruction. Such human behavior is plausible when the user simply takes the necessary steps "without thinking" because he is inattentive or because he is not familiar with online banking.

A possible solution to this problem is to extend the protocol so that H is forced to verify the message displayed by D. We slightly adapt the protocol and assume that D displays a random verification code vc together with m. H in turn enters not only the PIN but also vc into the reader's keypad. The requirement to enter vc forces the human to pay attention and to read the displayed message. The following protocol Transauth b specifies this modified protocol.

Protocol Transauth b

$$
\begin{aligned}
D &: \text{knows}(\langle H, PIN, ltkD\rangle)\\
S &: \text{knows}(\langle H, D, \text{pk}(ltkD)\rangle)\\
H &: \text{knows}(\langle D, PIN\rangle)\\
H \circ\!\!\rightarrow\!\!\circ\ P &: \text{fresh}(m).m\\
P \circ\!\!\rightarrow\!\!\circ\ D &: m\\
D \bullet\!\!\rightarrow\!\!\bullet\ H &: \text{fresh}(vc).\langle m, vc\rangle\\
H \bullet\!\!\rightarrow\!\!\bullet\ D &: \langle PIN, vc\rangle\\
D \circ\!\!\rightarrow\!\!\circ\ P &: \{m\}_{ltkD}\ /\ signature\\
P \circ\!\!\rightarrow\!\!\circ\ S &: \langle m, signature\rangle\ /\ \langle m, \{m\}_{ltkD}\rangle
\end{aligned}
$$

Tamarin's verification results for this modified protocol show that originating authenticity indeed holds. Note that originating authenticity does not guarantee protection against replay attacks. Such attacks can be foiled by pairing m with a monotone increasing counter in the penultimate protocol message.

This case study shows the subtleties of HISPs regarding the topologies in which the protocols are executed. It demonstrates the applicability of our characterization and our tool-supported protocol model to capture such subtleties and to guide the design of corresponding HISP.

7.3 Formal Analysis of Secure HISPs

As we introduced in our methodology for the development of secure HISPs, Tamarin can be used to analyze the protocols and verify their security properties. In the following, we analyze an existing protocol proposal for Vote électronique. An important security goal this protocol aims to provide is that of verifiability. With respect to the requirements introduced in Chapter 2 we give a formal definition of individual verifiability of cast as intended as a channel goal. Our modular definitions of channel goals in Chapter 5 allows us to define such additional channel goals with ease. In the following, we first introduce, and formally define, cast as intended individual verifiability. Afterwards, we analyze the Vote électronique reference protocol with respect to individual verifiability and the security goals defined in Chapter 5. We use this analysis to explain how Tamarin is used to understand the necessary conditions regarding knowledge and trust assumptions. We shall see that the protocol indeed enables verifiability and authenticity. Due to the model assumption that the voter's computer is controlled by the adversary, confidentiality does not hold. These results say that even if the voter's computer is fully compromised, vote integrity is still given, while secrecy of the vote is not.

Individual Verifiability

We extend our modular definitions of the general security properties defined in Chapter 5 with the property of verifiability. Since we focus on communication channels, we define verifiability with respect to the communication of messages, e.g., ballots in an Internet voting system.

A channel provides verifiability for an agent S with respect to a message m and an agent R, if m was communicated from S to R and S may verify this communication. This is analogous to our definition for authenticity but in contrast to authenticity, from the perspective of the sender. To identify a message m that is intended to be verifiably communicated, we annotate the protocol rule in which such a message is verified with a $\mathsf{Verify}(S, R, m)$ action.

Definition 7.1 *The* verifiability property *is defined by* (p_{verif}, q_{verif})*, where*

$$\begin{aligned} p_{verif}(tr, S, R, m) &:= \mathsf{Verify}(S, R, m) \in tr \\ q_{verif}(tr, S, R, m) &:= \mathsf{Verify}(S, R, m) \in tr \\ &\quad \rightarrow \mathrm{communicate}(tr, S, R, m). \end{aligned}$$

We use this definition in the following to prove that the reference protocol provides individual verifiability for the voter with respect to the communication of the ballot.

Formal Analysis of the Vote Électronique Reference Protocol

In the following we describe the Vote électronique reference protocol in detail. We then specify the described protocol in our extended Alice & Bob notation and present the results of the formal analysis using Tamarin.

Setup. Prior to the election, the voter receives an official envelope by mail, which is assumed to be secure. This envelope contains a voter authorization card, containing the voter's name and a voter ID. Separately, the envelope contains a code list with a code list ID, the candidates or options, a control code for every option, a confirmation code, and a finalization code. The election server knows all the contents of the code sheets and the eligible voter's IDs.

Vote Casting Phase. To cast a vote, the voter does the following. To simplify the model, we assume, that the voter has already read the voter authorization card and therefore knows the corresponding voter ID. Furthermore, we model the protocol such that the voter first reads all information of the code sheet at once and remembers all this information. Note that our results do not depend on this simplifying assumption. That is, the voter could also read just the information from the code sheet that is necessary for the corresponding step. After reading from the code sheet, the voter authenticates himself to the election server using his voter ID and then he enters the code list ID together with the chosen candidate or option into his computer. All this information is sent to the election server. The election server responds with the

corresponding control code for every submitted candidate. The voter compares this confirmation code to the one initially read from the code sheet. If they correspond, he enters the confirmation code into the computer, which in turn sends it to the election server. The server verifies that the received confirmation code corresponds to the expected one and if this is the case, it sends back the finalization code to the voter. The voter, finally, verifies that the received finalization code indeed corresponds to the expected one. The vote casting phase is specified in our extended Alice & Bob notation as depicted in Figure 7.3.

$$
\begin{array}{rcl}
H & : & \text{knows}(\langle D, voterID, voterName\rangle) \\
D & : & \text{knows}(\langle S, codeListID, m, controlCode, confirmationCode, \\
 & & \quad finalizationCode, voterName\rangle) \\
S & : & \text{knows}(\langle D, voterID, codeListID, m, controlCode, \\
 & & \quad confirmationCode, finalizationCode\rangle) \\
D \bullet\!\!\rightarrow\!\bullet H & : & \langle S, codeListID, m, controlCode, confirmationCode, \\
 & & \quad finalizationCode\rangle \\
H \circ\!\!\rightarrow\!\circ P & : & \langle voterID, codelistID, m\rangle \\
P \circ\!\!\rightarrow\!\circ S & : & \langle voterID, codelistID, m\rangle \\
S \circ\!\!\rightarrow\!\circ P & : & controlCode \\
P \circ\!\!\rightarrow\!\circ H & : & controlCode \\
H \circ\!\!\rightarrow\!\circ P & : & confirmationCode \\
P \circ\!\!\rightarrow\!\circ S & : & confirmationCode \\
S \circ\!\!\rightarrow\!\circ P & : & finalizationCode \\
P \circ\!\!\rightarrow\!\circ H & : & finalizationCode
\end{array}
$$

Figure 7.3: Vote électronique reference protocol.

The specification in our protocol specification language is straight-forward and follows the rules introduced in Chapter 5.

Verification Results. Tamarin's verification results prove authenticity as well as individual verifiability to hold. Since the vote m is submitted in clear-text to the platform, it is obvious, that the protocol does not provide confidentiality.

Note that D in the voter's initial knowledge is necessary. It models the assumption, that the voter has exclusive access to the code sheet received in the envelope during the setup phase. Otherwise, Tamarin presents a trace in which authenticity does not hold. In this trace, the adversary creates a fake code sheet and replaces the original code sheet with the fake one. Then he waits until the voter sends his voter ID together with the vote to learn the voter ID. Afterwards, he impersonates the voter with the voter ID and code sheet.

In this part, we focussed on insecure personal computers and we assumed the user to behave exactly as defined in the protocol. However, users may also deviate from

the protocol. In Part II, we examine what the user could do wrong and what the implications on the security properties are, if the user does so.

Part II

Human Error

8 Human Error in Security Protocols

Formal methods for the analysis of security protocols and the verification of their security properties usually focus on the communication between computers. The interaction between human users and their computers is often neglected. As introduced in the chapters above, for a significant number of practical applications at least one communication end is human. In this part, we examine how erroneous user behavior may compromise communication systems' security properties. In this chapter, we provide the preliminaries related to human error in security protocols. We give an overview of this part of the thesis in Section 8.1. In Section 8.2 we give a definition of the term human error and introduce existing methods for human-error analysis. In Section 8.3, we examine the role of human users as protocol agents more closely. We identify the common tasks users are usually expected to perform and we derive potential failure modes with this respect.

8.1 Human Errors

In Part I, we examined a compromised client platform's influence on secure communication between human users and remote communication partners. We assumed honest users to correctly follow the protocol, i.e., to be compliant with the protocol specification. However, humans are error prone and their behavior may unintentionally deviate from a system designer's expectations. The resulting errors may compromise even provably secure protocols and are sources of various attacks against practical protocols that are used for applications such as online banking and Internet voting. An example is that of phishing attacks where users are tricked into deviating from the specified protocol and thus, to provide confidential information to the adversary.

In this part we take human users' error-proneness into account. We focus on the second row of Table 8.1, i.e., we examine the effects caused by users who do not necessarily comply with the protocol specification. We start by identifying the negative effects of human error in settings where the user's personal computer is honest with respect to our definitions in Chapter 5. We then examine system failures where the user's computer is dishonest but not under adversarial control, that is, it may deviate from the protocol specification only in specific ways. An example is a web application, where the user may use the browser's back button and thereby, depending on the application, unexpectedly deviates from the specified protocol. We finally examine

		Platform	
		Honest	Dishonest
User	Compliant	Classical Protocol Analysis	HISP
	Non-Compliant	Human Error	Credulous Human

Table 8.1: Overview of formal modeling and analysis methods with respect to client-side security.

the setting where the user credulously interacts with his dishonest personal computer as it is the case for phishing attacks based on malware infection.

8.2 Human-Error Analysis Methods

As we introduced in Part I, we assume that humans cannot perform cryptographic operations and they cannot be certain whether or not information appearing on their computer's screen faithfully represent the messages communicated with the remote system and whether or not their computer leaks confidential information to unauthorized third parties. In addition to these limitations, we now assume that human users are also prone to various kinds of errors. In the following we explain what human errors are, how human errors can be detected, and different analysis methods proposed in scientific literature.

Definition of Human Error

The term human error is not clearly defined and researchers disagree over many aspects and even whether or not human error exists with respect to system failures. For example, Dekker [27] and Senders et al. [89] express that human error is a post hoc judgement on the outcome of human behavior. That is, that users' actions are judged as errors only after the outcome causes a system failure. In contrast, we propose that the role of the honest human user must be specified explicitly and we refer to the term error for any deviation from this specification by a honest user, even if the system's outcome remains correct. In such cases we coin the term *human-error robustness*, which describes the property that a system's security properties are independent of erroneous user behavior. In Chapter 9 we show examples for this independence and corresponding necessary and sufficient conditions for protocols to be robust against specific human errors.

Security Analysis Methods

There exist different approaches to analyze systems with respect to human errors. Kuo's classification [52] distinguishes between *user-centric analysis* and *system-centric*

analysis. Whereas user-centric analysis is centered around modeling humans, their perception, and information processing more precisely, system-centric analysis abstracts from detailed assumptions about the human user. Next, we summarize the analysis methods and techniques that are proposed to be applied to identify human errors. Afterwards, we relate our approach with these methods.

User-centric Analysis. These methods focus on the user's perception, information processing, and the resulting behavior. For example, Cranor [25] proposes a security analysis framework consisting of four steps: *task identification*, *task automation*, *failure identification*, and *failure mitigation*. During the task identification step, the system designer is supposed to "identify all of the points where the system relies on humans to perform security-critical functions". The task automation step is dedicated to finding possible ways to automate some of the human's security-critical tasks. In the failure identification step, potential failure modes are identified. Finally, in the failure mitigation step, mitigation approaches for the previously identified failure modes are identified and implemented.

Visual Analysis. A widely used visual method for more generally analyzing system failures is that of attack trees or threat trees [3, 85]. This method allows the evaluator to systematically identify a set of actions that result in a security breach or failure of a system. The root of the tree represents the attack's goal. Its children represent possible conditions to achieve the goal. Recursively, the conditions to achieve a specific condition, are represented as children of the corresponding child. The resulting tree serves as an overview of the actions that an attacker must perform to successfully attack the system, or of all combinations of failures that lead to a security-critical system failure.

Formal Analysis. Formal symbolic methods help to understand and to systematically detect logical flaws in security protocols. An example is the formal model we introduced above in Part I. Formal methods are used to verify numerous classes of security protocols. However, these protocols usually apply cryptographic primitives to secure the communication between computer systems. Communication applications, such as online banking and Internet voting, rely on a secure communication channel directly between a human and a remote communication partner. Methods, tools, and foundation results for protocols where humans are involved are largely missing. Section 4.4 gives an overview of related work with this respect.

Usability testing. This a widely adapted empirical evaluation method. For a specific system, a group of test users performs a predefined list of tasks. The test users' (mis)behavior is observed and evaluated. In [62], Nielsen et al. show that for usability tests where problems are found with probability greater than 0.3 per evaluation, more than 75% of all usability problems can be found with only up to five test users. How-

ever, with respect to security, usability testing is not sufficient since one cannot identify all possible threats. Therefore, a comprehensive systematic approach is preferable.

Cognitive walkthrough. This is an analytical evaluation method where first the assumptions about the end users, that is, their knowledge and expertise, and the tasks are defined. Based on these assumptions, the evaluator steps through each action the user has to perform to carry out the task. For each of these actions, the evaluator assesses whether or not the end user possesses the knowledge and expertise to perform the action [71]. In this way, whenever the user has the option to perform more than one action and one of these actions leads to a system failure, the evaluator may identify this.

Heuristic evaluation. Whereas the cognitive walkthrough method is specific to an application, heuristic evaluation is a more generic method. For a given application, evaluators identify usability issues based on a pre-determined list of heuristics [63].

The effectiveness of systematic techniques, such as the ones described above, depends on the expertise of the evaluator [28]. Our formal approach combines the idea of cognitive walkthroughs and heuristic evaluation in the following way. We introduce an extended formal protocol model based on the Tamarin model introduced in Chapter 3. This model allows one to specify the communication protocol including the user's interaction with his computer(s). To model human errors, we first identify common human errors in communication applications and model these errors as additional rewriting rules. The resulting error behavior model in combination with Tamarin's adversary model allow one to find potential system failures and security threats caused by human errors.

8.3 Human-Interaction in Security Protocols

Security protocols employ cryptographic operations to protect data that is being exchanged. Since most humans are too limited to perform the necessary cryptographic operations, they are usually carried out by their computers. Humans are unreliable and error prone and it is generally preferable to "remove the human from the loop" [25]. However, in some security-critical applications humans must conduct certain operations. For example, in Internet voting the voter must express his free will and cast a corresponding vote. The communication between the human user and his computer is usually managed by computer programs providing user interfaces to the user. The interaction between the user and the computer, i.e., between the user and a program's interfaces must prevent common human errors.

We follow the concept of security ceremonies [30] and consider human-computer interaction as part of an extended security protocol. We next introduce security protocols and we characterize possible pitfalls regarding human error. We apply a system-

atic approach to identify potential failure modes for the tasks the user is expected to carry out in security protocols.

Security Protocols

As a common representation of security protocols, we introduced the extended Alice & Bob notation at the end of Chapter 5. Figure 8.1 illustrates an authentication scheme for web servers in the extended Alice & Bob notation. We describe the protocol in Example 8.1.

$$
\begin{array}{rcl}
H & : & \text{knows}(\langle S, \mathit{username}, \mathit{password}\rangle) \\
S & : & \text{knows}(\langle H, \mathit{username}, \mathit{password}\rangle) \\
H \bullet\!\!\rightarrow\!\bullet\ P & : & S \\
P \circ\!\!\rightarrow\!\circ\ S & : & H \\
S \circ\!\!\rightarrow\!\circ\ P & : & \mathit{fresh}(\mathit{nonce}).\mathit{nonce} \\
P \bullet\!\!\rightarrow\!\bullet\ H & : & S \\
H \bullet\!\!\rightarrow\!\bullet\ P & : & \langle \mathit{username}, \mathit{password}\rangle \\
P \circ\!\!\rightarrow\!\circ\ S & : & \langle \mathit{username}, \mathrm{h}(\mathit{username}, \mathit{password}, \mathit{nonce})\rangle
\end{array}
$$

Figure 8.1: HTTP Digest Access Authentication.

Example 8.1 *The* HTTP Digest Access Authentication Scheme *[34] is an extension to HTTP and is a simple access authentication method for online resources that aims to "avoid the most serious security flaws of* HTTP Basic Authentication*". In contrast to Basic Authentication, the password is not sent in cleartext. The relevant parts of the protocol's specification is depicted in Figure 8.1 and described in the following.*

The user points his browser to a protected page on server S that requires authentication. S responds with the 401 "Unauthorized" *HTTP status code and provides the authentication realm and a fresh nonce. The browser shows the authentication realm to the user and requests him to enter his username and password to access the realm. After the user entered his username and password, the browser again requests the protected page and adds an authentication header including the username and the response code. This response code is a hash of the username, the password, the nonce, and further values. If the response code is correct, i.e., if the server accepts the username and password, it responds with the requested page source data that the browser finally presents to the user.*

Classical protocol security analysis methods focus on security flaws caused by an adversary. The interaction between human users and their computers is usually neglected and honest agents are expected to behave exactly as specified in the protocol. This is a strong assumption and in fact, when developing secure systems, humans must often be considered "the weakest link in the chain" [87]. The reasons are manyfold. For example, security is often not the primary goal [102] of an application, users just want to get their work done, not fiddling with annoying security settings and

not spending time in learning to handle them. Another reason is given by wrong intuitions about security concepts. Considering the message exchange illustrated in Figure 8.1 more closely we identify several difficulties with human participation. For example, a human user may mistype his password or he may confuse it with his username. In the following we identify a set of common human errors in security protocols. These errors will serve as the basis for our formal human-error analysis model and we will identify a number of necessary and sufficient conditions for protocols and their implementation to be robust against a relevant subset of these failures.

Human-Errors in Protocol Interaction

Whereas computer programs that honestly execute communication protocols are tied to their implementation and thus to strict specifications, humans are often not. In practice, they may deviate from the specification in various ways. In this thesis we focus on human error in a technical sense, that is, we identify possible misbehavior but we do not reason about the likelihood of its occurrence. This would require empirical user studies such as carried out in the research domains of *HCISec* and *Usable Security*.

To identify common human errors regarding secure communication applications, we apply a systematic process that covers the first and the third step of Cranor's process to identify human threats to system security [25].

Task identification.

As introduced in Part I, we consider humans as protocol agents with clearly specified capabilities. They may *send*, *receive*, *compare*, *concatenate* (pair) and *select* (project) terms. We do not impose any restriction on human memory and we assume that humans can *generate random* (fresh) terms. Thus, humans are expected to remember all terms received on any channel and to be able to output any term constructible from their knowledge using pairing and projection on any available channel. However, they cannot perform cryptographic operations without the help of supporting technology, such as their personal computers.

Two obvious events where a protocol involving humans relies on the end user are sending and receiving of terms. For example, Figure 8.2 depicts the relevant human interaction tasks of the above introduced HTTP Digest Access Authentication Scheme. The user is expected to enter his username and his password into his personal computer, or more precisely into a form prompted by the browser, and to receive, i.e., to read the requested information.

Failure identification.

Based on all the tasks a user can carry out in an Alice & Bob protocol specification, we identify the potential failures modes, a human may cause in security protocols. We group these failures with respect to the following tasks.

$$
\begin{array}{rcl}
H & : & \mathrm{knows}(\langle S, \mathit{username}, \mathit{password}\rangle) \\
H \bullet\!\!\rightarrow\!\bullet & : & S \\
\bullet\!\!\rightarrow\!\bullet\ H & : & S \\
H \bullet\!\!\rightarrow\!\bullet & : & \langle \mathit{username}, \mathit{password}\rangle
\end{array}
$$

Figure 8.2: Human interaction in HTTP Digest Access Authentication.

Sending a term. Whenever a human is expected to send a term, he may (a) *not send anything* or he may (b) *send a different term* from his knowledge. For example, a user requesting a web site may type in a different URL than intended. If more than one channel is available, the user may (c) *use the wrong channel*. For example, in a two-factor authentication protocol, where a human is expected to send messages from his personal computer as well as from his smartphone, he may use the wrong device to send a particular message.

Receiving a term. In the case where a user receives a term he is expected to learn this term and to correspondingly update his knowledge. In this situation, he may be inattentive and (d) *learn a different term*. This term may be a publicly known or freshly generated one. Moreover, the user may (e) *not learn a term at all*. For example, he may ignore a warning displayed on his computer's screen. He may also (f) *confuse channels*, if more than one channel exists. Altogether, users may not update their knowledge appropriately.

Comparing terms. In our specification of security protocols, terms are compared implicitly. If a received term does not correspond to the expected one known to the receiver, the protocol blocks. However, if the receiver is human he may (g) *ignore the comparison* and further execute the protocol. For example, if a user is expected to connect to a web site using SSL, he may ignore that his browser's padlock symbol does not show up correctly.

Concatenating terms. Users are expected to concatenate specific terms in a given order but they may concatenate the wrong terms or in a wrong order. All in all, they may (h) *use any arbitrary term* from their knowledge.

Selecting terms. If users are expected to select a specific term, they may just (i) *use any arbitrary term* from their knowledge.

Generating fresh terms. Whenever humans are expected to provide random terms, they may (j) *generate weak random values* that the adversary can guess. That is, the user may use a publicly known term instead. For example, if a human user is expected to register a password, he may choose a common word.

Remember terms. If humans are expected to remember terms, they may forget them, i.e., the (k) *terms are removed from their knowledge*. They may also remember them wrongly, i.e., terms in their knowledge may be exchanged with (l) *other terms*

in their knowledge. In many practical applications where humans are required to remember information, the service cannot be used without this information. For example, if the user forgets his password to access his webmail account, he cannot access this account any longer. To address this failure, the service providers usually allow the users to recover their password.

Executing a protocol in the expected order. Humans may deviate from the expected order. They may (m) *ignore non-blocking protocol steps*. They may (n) *execute an expected protocol step repeatedly* and they may (o) *confuse the expected order* of the protocol steps as long as the protocol does not block. An example are web applications, where the user is assumed to not use his browser's back-button. Consider for example a shopping application, where the user in the end confirms an order. After that, his order is processed and his credit card is charged. If the user then uses his browser's back-button he may trigger the purchase processing a second time.

The above identified failure modes serve as a heuristic for the analysis of communication applications. We are interested in the implications that such behavior has with respect to a system's security properties. In our model, we expect users to be able to deviate from the specified behavior and we analyze the resulting protocol using a standard Dolev-Yao adversary's capabilities.

Note that the identified failure modes interrelate with respect to their impact on protocols' security properties. For example, there is no difference between a human user being inattentive while receiving a term, thereby not learning the term, and learning the correct term but forgetting it before using it in a subsequent protocol step. That is, if a protocol is robust against specific human errors in our model, it is robust against the related failure modes too. Therefore, our formal model in Chapter 9 will only cover a relevant subset of the above identified failure modes.

8.4 Related Work on Human-Error Analysis

It is difficult for designers of security-critical communication applications to consider all possible misbehavior and to detect the resulting negative effects caused by human errors. To address this difficulty, a number of systematic approaches exist. For example, Cranor proposes a four-step iterative process [25] to identify and mitigate human threats for the design of security-critical systems. The process is similar to common risk management processes, as we describe in [7]. Threats are identified and corresponding countermeasures are evaluated. We partly applied this methodology above in Section 8.3 to identify possible human failure modes in communication applications.

In her thesis [52], Kuo provides a comprehensive overview of human error, its analysis, and possible countermeasures for a number of applications. She identifies, that formal methods could be of great help for the analysis of systems and protocols regarding human error. However, her work does not provide a formal model nor a

formal analysis method. We provide formal models and methods to verify a system's security properties under the assumption that users are prone to errors.

Systematic methodologies help to identify common (and obvious) flaws, but they depend on the evaluator's expertise and imagination. In classical protocol analysis, the well-known Needham-Schroeder protocol showed that flaws may remain hidden for many years [53], but are unveiled by formal reasoning. Therefore, it is natural to ask, whether similar methods can be used to reason about the negative effects of human errors in security protocols too. With this respect, the approach of Bella and Coles-Kemp to extend security ceremonies with technical and social elements such as a human agent's belief system and cultural values [10, 11] is closely related. They propose modeling security ceremonies using five layers: (1) the security of the protocol executed by the computers of the communicating partners; (2) the inter-process communication of the operating system; (3) the socio-technical protocol whereby a user interacts with a graphical user interface; (4) the user's state of mind and (5) the influence of society on individuals. In [11], they formalize layer three, which is responsible for human-computer interaction, and they give a case study. The case study demonstrates the verification of a user's confidence in the privacy assurance offered by a service provider. In contrast to Bella and Coles-Kemp we model erroneous user behavior and its impact on a protocol's security properties.

Based on our Secure Platform Problem Model in Chapter 5, Schmid provides a formal model to analyze human errors in security protocols [83]. Human error is modeled as *role substitution attacks*, where the adversary is allowed to choose any specification for the human role that the adversary can substitute for the original protocol specification. To limit an adversary's capabilities, the model allows to define countermeasures as a user's partial knowledge about the protocol. That is, countermeasures restrict the adversary's capabilities to derive human role specifications. Our modeling approach is complementary. Whereas Schmid's approach starts with a credulous human and allows to define the human user's compliance as restrictions regarding possible deviations from the human role specification, we start the other way round. We take the original human role specification and allow deviations from the role specification explicitly. This allows us to examine a protocol's robustness with respect to a specific failure mode or combinations thereof.

Curzon and Blandford introduce a formal reusable user model implemented in higher-order logic [26]. They model interactive systems and allow detecting systematic human errors with respect to generic human-system interaction. They focus on *post-completion errors*, *communication-goal errors*, and *device delay errors*. Communication goals with this respect are a "task dependent mental list of information the user knows he must communicate to the device". They model communication goals as guard-action pairs. Based on their generic formal cognitive model, various extensions and applications have been presented. Rukšėnas et al. formally specify human-computer interaction using an extended cognitive architecture [76]. The extended architecture separates the state spaces of users and devices and possible user behavior is specified as traces of actions. In [77], they further extend their modeling approach to examine

cognitive processes that affect both the information flow from the human user into the computer system and the resilience of the whole system to intruder attacks. Their analysis focusses on the confidentiality property. They provide a number of examples, where they apply their modeling approach to find possible security flaws and corresponding fixes. With this respect, our modeling and verification approach has similar goals. However, in contrast to Rukšėnas et al. we verify additional properties such as authenticity and verifiability. Moreover, our adversary model is more abstract and generic. We do not have to define the adversary's goals for different applications specifically.

In the next chapter, we use our findings to develop formal models to analyze a protocol's robustness against the above identified human errors.

9 Human-Error Analysis

In this chapter we present formal methods for the analysis of the effects of erroneous human behavior in secure communication applications. We first introduce our extended protocol model in Section 9.1. Then, in Section 9.2, we refine our protocol specification language to cover human misbehavior. In Section 9.3, we introduce our human-error modeling and analysis approach to cover the individual failure modes identified in Section 8.3. As our examples show, even if the platform itself acts honestly, the protocol must stick to specific conditions to prevent erroneous human behavior to harm the protocol's security properties. In Section 9.4, we take dishonest computers into account. Finally, in Section 9.5, we provide a systematic methodology for the formal analysis of specific human-errors and combinations thereof.

9.1 Security Protocol Model Extensions

In order to model erroneous human behavior, we extend our Secure Platform Problem model defined in Part I. We refer to Chapter 5 for specific details. In the following, we describe our extended security protocol model and introduce new facts and rules.

Model Facts and Rules

We refine the fixed set of fact symbols Σ_{Fact} and we define additional rules to model human agents in more detail. The additional fact symbols extend the set defined in Section 5.1 and we highlight them in boldface. The following equations summarize all facts used in our extended security protocol model. We describe the new facts and refer to Section 5.1 for details on the facts that are already described there.

$$\Sigma_{Fact} := \Sigma^1_{Fact} \cup \Sigma^2_{Fact} \cup \Sigma^3_{Fact} \cup \Sigma^4_{Fact} \cup \Sigma^5_{Fact} \cup \{\textbf{Failure}\}\text{, where}$$
$$\Sigma^1_{Fact} := \{\mathsf{Fr}, \mathsf{Out}, \mathsf{In}, !\mathsf{K}, \mathsf{Agent}, \mathsf{Honest}, \mathsf{Dishonest}, \mathsf{Trust}, \textbf{Human}, \textbf{Computer}\},$$
$$\Sigma^2_{Fact} := \{!\mathsf{Auth}, !\mathsf{Conf}, \mathsf{Fresh}, \mathsf{Comm}, \mathsf{Learn}, \textbf{!HK}, \textbf{Eq}, \textbf{Fail}\},$$
$$\Sigma^3_{Fact} := \{!\mathsf{Sec}, \mathsf{Secret}, \mathsf{Authentic}, \mathsf{Verify}, \mathsf{AgentState}, \textbf{Compare}, \textbf{Verified}\},$$
$$\Sigma^4_{Fact} := \{\mathsf{Snd}, \mathsf{Rcv}\},$$
$$\Sigma^5_{Fact} := \{\textbf{Sysfail}\}.$$

The set of all facts $\mathcal{F}$ is therefore

$$\mathcal{F} := \left\{ f(t_1, \ldots, t_k) \mid f \in \Sigma^k_{Facts} \wedge t_1, \ldots, t_k \in \mathcal{T} \right\}.$$

Humans and Computers

We use Human and Computer action facts to mark corresponding agents in a trace. A human agent A is marked with $\mathsf{Human}(A)$, a computer agent B with $\mathsf{Computer}(B)$. Once an agent is indicated to be human it cannot become a computer in the same trace and vice-versa, an agent indicated to be a computer cannot become a human in the same trace. We are therefore interested in the set of traces $TR(\mathcal{R})$ that satisfies the following property:

$$\forall tr \in TR(\mathcal{R}), A, B \in \mathcal{C}_{\mathsf{pub}} : \mathsf{Human}(A) \in tr \wedge \mathsf{Computer}(B) \in tr \implies A \neq B.$$

This restricts the set of traces $TR(\mathcal{R})$ for a protocol $\mathcal{R}$ to those where agents do not switch between being human and being computer.

The set of agents appearing in a trace tr, denoted by $Agents(tr)$, remains still the set of all public constants A such that $\mathsf{AgentState}(A, c, n)$ appears in a state of tr for some c and n.

Human Knowledge

We handle human knowledge using additional !HK facts. These facts represent the terms that an agent may create. That is, the initial knowledge, all terms learnt during a protocol execution, and all derivable terms a human can produce in a protocol execution. A human agent's knowledge is represented with an arbitrary number of persistent $!\mathsf{HK}(A, t)$ facts, where $A \in \mathcal{V}_{\mathsf{pub}}$ refers to the human agent's name and $t \in \mathcal{T}$ represents a term known to A. Initial knowledge and terms that a human agent learns in a protocol run are explicitly stated in the protocol specification. We will discuss this in Section 9.2 in more detail.

Additionally, we use the following set of construction rules $\mathcal{HK}$ to model human knowledge:

$$\mathcal{HK} := \{ \quad [!\mathsf{HK}(A, m), !\mathsf{HK}(A, m')] \dashv [\, \mathsf{Human}(A) \,] \vdash [!\mathsf{HK}(A, \langle m, m' \rangle)], \tag{9.1}$$

$$[\quad] \dashv [\, \mathsf{Human}(A) \,] \vdash [!\mathsf{HK}(A, m : pub)] \ \}. \tag{9.2}$$

Rule 9.1 states that whenever a human agent A knows two terms m and m', he also knows the pair of the two values. Rule 9.2 states that a human agent A knows every public term.

However, the rules in $\mathcal{HK}$ allow a human user to combine arbitrary known terms. This allows the user who confuses messages to send his entire knowledge instead of the term that was specified in the protocol. This is often not realistic as the next example shows.

Example 9.1 *Chaum's SureVote [21] Internet voting protocol, as introduced above in Section 4.2, improves simple code voting. The improved protocol uses verification codes that are sent back to the voter by the election authority. Hence, the voter can verify what candidate selection the election authority received. Figure 9.1 depicts the SureVote protocol in our extended Alice & Bob notation. Note that we describe human-interaction channels ($\hbar\!\!\rightarrow\!\!\hbar$) later in this section in full detail.*

$$
\begin{array}{rcl}
H & : & \mathrm{knows}(\langle S, c, \mathrm{h}(c), \mathrm{h}(\langle k, c\rangle), c', \mathrm{h}(c'), \mathrm{h}(\langle k, c'\rangle)\rangle) \\
S & : & \mathrm{knows}(\langle H, c, \mathrm{h}(c), \mathrm{h}(\langle k, c\rangle), c', \mathrm{h}(c'), \mathrm{h}(\langle k, c'\rangle)\rangle) \\
H \hbar\!\!\rightarrow\!\!\hbar P & : & \mathrm{h}(c) \ / \ code \\
P \circ\!\!\rightarrow\!\!\circ S & : & code \ / \ \mathrm{h}(c) \\
S \circ\!\!\rightarrow\!\!\circ P & : & \mathrm{h}(\langle k, c\rangle) \ / \ verificationcode \\
P \hbar\!\!\rightarrow\!\!\hbar H & : & verificationcode \\
H & : & \mathrm{compares}(\mathrm{h}(\langle k, c\rangle), verificationcode)
\end{array}
$$

Figure 9.1: Chaum's SureVote Internet voting protocol.

Additionally to the candidate names c and c', and the corresponding codes $\mathrm{h}(c)$ and $\mathrm{h}(c')$, the code-sheet contains verification codes $\mathrm{h}(\langle k, c\rangle)$ and $\mathrm{h}(\langle k, c'\rangle)$ for the candidates c and c', respectively. The election authority sends the verification code back to H who in turn compares the verification code to the corresponding code printed on the code-sheet. Therefore, if the human agent confuses the candidate code in the beginning, he will notice that when he verifies the verification code later on. However, using the Tamarin tool to analyze authenticity and verifiability, we find a trace where the human agent H applies Rules (9.1) *and* (9.15)*, where the human agent may send an arbitrary term from his knowledge instead of the specified one, to send the candidate code for another candidate together with the verification code of the chosen candidate. We introduce Rule 9.15 in Section 9.3. P sends the tuple to the insecure network and the adversary in turn passes the candidate code to S and the verification code back to P. Hence, the wrong candidate code is submitted to the server and P displays the verification code for the chosen candidate to the human agent, who finally verifies its correctness.*

In reality, the behavior described in Example 9.1 would require that the user interface supports the user's misbehavior. For example by displaying additional textfields to enter the additional information. In fact, Olsen et. al [64] presented a malicious user interface for the Norway Internet voting system to several hundred students. This interface was used to mount a phishing attack for collecting the private parts of the information distributed to the voters. In their study every single student was successfully tricked into entering the private information during the voting process. However, as mentioned above, in this part of the thesis, we do not focus on phishing attacks based on dishonest computers. Still, our modeling approach for insecure platforms and human error, especially our credulous-human model, may well serve for future research in this direction.

As Example 9.1 shows, due to the rules in $\mathcal{HK}$, the set of all traces $TR(\mathcal{R})$ for a protocol $\mathcal{R}$ may include unrealistic protocol executions. For example, executions

where the user enters all information into just one simple textfield. To omit these protocol executions, the rules in $\mathcal{HK}$ may be omitted. Instead, the protocol designer may model his assumptions about possible combinations of terms that the user may send along. In turn, the protocol specification then must cover these assumptions accordingly, i.e., in every protocol step the protocol designer must define the possible $!\mathsf{HK}(\_, m)$ facts. We will later show examples where we use this methodology when using the Tamarin tool for the automated analysis.

Comparison

In our Secure Platform Problem model above in Chapter 5, comparing received terms to already known terms is modeled implicitly. Receive rules are only applied if the received messages match the messages that are already known by the agent, that is, if the received term is equal to the term already present in the Agentstate fact. This models computer agents appropriately but it is not sufficient to model erroneous humans as we identified in Section 8.3, for example, in Failure (g). To later cover corresponding failures, we use the following rule in $\mathcal{CP}$ to model comparison explicitly.

$$\mathcal{CP} := \{\ [\mathsf{Compare}(A, m, m')] -[\ \mathsf{Eq}(m, m')\ ]\rightarrow [\mathsf{Verified}(A, m, m')]\ \} \tag{9.3}$$

Rule (9.3) models agent A comparing two terms m and m'. The rule is labeled with an Eq action. We additionally require that every term that a human agent receives has a different name. This requirement enforces explicit comparison using Compare and Verified facts as we will point out in Section 9.2 below. Finally, in addition to the trace restriction regarding humans and computers, we restrict the set of traces $TR(\mathcal{R})$ to the traces that satisfy the following property:

$$\forall tr \in TR(\mathcal{R}), m, m' \in \mathcal{M} : \mathsf{Eq}(m, m') \in tr \implies m = m'.$$

With this additional trace restriction, whenever two ground terms m and m' are marked to be equal in a trace, these terms are indeed equal.

Redefined Channel Assumptions

In this part we reduce the number of facts used to model channels to the two facts Snd and Rcv. To cover insecure, confidential, authentic, and secure channels, the facts are of the form $\mathsf{Snd}(p, A, B, m)$ and $\mathsf{Rcv}(p, A, B, m)$ respectively, where $p \in \{\mathsf{I}, \mathsf{A}, \mathsf{C}, \mathsf{S}\}$ indicates the channel property for an insecure, authentic, confidential, or secure channel. Figure 9.2 summarizes our redefinition of the channel rules.

Rules (9.6) and (9.7) model authentic channels. In Rule (9.6), the adversary learns the message (Out). The auxiliary !Auth fact ensures that in Rule (9.7) the adversary can neither alter the message nor its sender. The !Auth fact is persistent, which reflects the adversary's capability to replay authentically transmitted messages. The rules are annotated with the corresponding Snd and Rcv actions.

Confidential channels are modeled using Rules (9.8)–(9.10). Rule (9.8) creates an auxiliary !Conf fact and the adversary does not learn the message. Rule (9.9) represents the case where the adversary passes the (unknown) confidential message m to the intended recipient, possibly pretending that it stems from another sender (In). The !Conf fact is persistent, which reflects the adversary's capability to replay confidentially transmitted messages. Rule (9.10) represents the adversary's capability to access the confidential channel to deliver any message from his knowledge.

Rules (9.11) and (9.12) model secure channels. In Rule (5.13), the adversary learns nothing and an auxiliary !Sec fact is generated, which models that the adversary can neither alter the message nor its sender. Rule (9.12) models receiving a message from a secure channel. The !Sec fact is persistent, allowing the adversary to replay securely transmitted messages.

Human Interaction

In addition to the channel rules for insecure, confidential, authentic, and secure channels, we extend the channel abstraction with a *human-interaction channel*. This channel represents the direct interaction between a human user and his computer. Such a channel has different properties than the previous channels since the adversary cannot replay messages. Consider for example a user entering a message into his computer using a keyboard. It is reasonable to assume, that the adversary cannot replay this message and the same applies for a user reading information from a computer screen. Rules (9.13) and (9.14) extend the existing channel rules.

The rules model, that term m sent by a human agent A or received by a human agent B using a channel with property HI is directly delivered and therefore, the adversary does not learn m nor is he able to modify or replaying it. Note that these rules alone still allow blocking which is not realistic in practice because whenever the user enters some information into his personal computer, for example using a keyboard, it is immediately received by the personal computer. However, this model is sufficient for our purposes.

We modify and extend the model specification rules $\mathcal{R}_{Model}$ from Section 5.1 with the above extensions and we omit dishonest agents. Hence we have

$$\mathcal{R}_{HEModel} := \mathcal{FR} \cup \mathcal{MD} \cup \mathcal{CHE} \cup \mathcal{HK} \cup \mathcal{CP}.$$

We use this extended model to analyze a protocol's robustness against human errors and to derive necessary and sufficient robustness conditions.

Failure Measures

Usually, honest human users do not fail repeatedly. That is, users may confuse terms when reading them from a sheet of paper, for example by shifting to a line lower than intended, but they do not repeat this failure over and over again. The number of times the human user is assumed to fail is an assumption and may be captured by empirical

$$\mathcal{CHE} := \{\ [\mathsf{Snd}(\mathsf{I}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{I}, A, B, m)\ ] \mapsto [\mathsf{Out}(\langle A, B, m\rangle)], \tag{9.4}$$

$$[\mathsf{In}(\langle A, B, m\rangle)] \dashv [\ \mathsf{Rcv}(\mathsf{I}, A, B, m)\ ] \mapsto [\mathsf{Rcv}(\mathsf{I}, A, B, m)], \tag{9.5}$$

$$[\mathsf{Snd}(\mathsf{A}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{A}, A, B, m)\ ] \mapsto [!\mathsf{Auth}(A, m), \mathsf{Out}(\langle A, B, m\rangle)], \tag{9.6}$$

$$[!\mathsf{Auth}(A, m), \mathsf{In}(B)] \dashv [\ \mathsf{Rcv}(\mathsf{A}, A, B, m)\ ] \mapsto [\mathsf{Rcv}(\mathsf{A}, A, B, m)], \tag{9.7}$$

$$[\mathsf{Snd}(\mathsf{C}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{C}, A, B, m)\ ] \mapsto [!\mathsf{Conf}(B, m)], \tag{9.8}$$

$$[!\mathsf{Conf}(B, m), \mathsf{In}(A)] \dashv [\ \mathsf{Rcv}(\mathsf{C}, A, B, m)\ ] \mapsto [\mathsf{Rcv}(\mathsf{C}, A, B, m)], \tag{9.9}$$

$$[\mathsf{In}(\langle A, B, m\rangle)] \dashv [\ \ ] \mapsto [\mathsf{Rcv}(\mathsf{C}, A, B, m)], \tag{9.10}$$

$$[\mathsf{Snd}(\mathsf{S}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{S}, A, B, m)\ ] \mapsto [!\mathsf{Sec}(A, B, m)], \tag{9.11}$$

$$[!\mathsf{Sec}(A, B, m)] \dashv [\ \mathsf{Rcv}(\mathsf{S}, A, B, m)\ ] \mapsto [\mathsf{Rcv}(\mathsf{S}, A, B, m)], \tag{9.12}$$

$$[\mathsf{Snd}(\mathsf{HI}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{HI}, A, B, m), \mathsf{Human}(A)\ ] \mapsto [\mathsf{Rcv}(\mathsf{HI}, A, B, m)], \tag{9.13}$$

$$[\mathsf{Snd}(\mathsf{HI}, A, B, m)] \dashv [\ \mathsf{Snd}(\mathsf{HI}, A, B, m), \mathsf{Human}(B)\ ] \mapsto [\mathsf{Rcv}(\mathsf{HI}, A, B, m)]\ \} \tag{9.14}$$

Figure 9.2: Redefined and extended channel rules.

user studies for a given application. However, finding reasonable numbers with this respect is beyond the scope of this thesis.

We use Fail facts to model the assumption that a human user may fail in a specific way. We introduce the Fail facts and their usage in Section 9.2. Fail facts are of the form $\mathsf{Fail}(A, failureID)$, where $A \in \mathcal{V}_{\mathsf{pub}}$ is a human agent's name and $failureID \in \mathcal{C}_{\mathsf{pub}}$ is the failure identifier of the set of rewriting rules representing the corresponding failure mode. We summarize the failure identifiers and the corresponding failure modes and rewriting rules in Tables 9.1 and 9.2.

Whenever a rewriting rule modeling a failure is applied in a protocol run, a corresponding Fail fact is consumed. The number of these facts therefore models the number of failure repetitions an agent is assumed to do in a protocol run. It also allows to model specific misbehavior. For example, it allows to model a protocol, where the human user is allowed to only fail in specific ways.

As we will show later in this chapter, certain human failures require the computer systems to support them. For example, the user cannot send a wrong term, if the user interface does not show a corresponding textfield to enter it. Another example concerns protocol step confusion. The user may only repeat protocol steps if the application running on his computer allows him to. To model this, we use Sysfail facts. We introduce the Sysfail facts and their usage in Section 9.2. Sysfail facts are of the form $\mathsf{Sysfail}(A, c, n, c', n')$, where $A \in \mathcal{V}_{\mathsf{pub}}$ is a computer agent's name, $c \in \mathcal{C}_{\mathsf{pub}}$ identifies a state from which A may jump to state $c' \in \mathcal{C}_{\mathsf{pub}}$. n and n' specify A's knowledge in the states identified with c and c', respectively. These facts allow a protocol designer to specify possible but undesirable state transitions. Note that in

contrast to human agents and the adversary, the computer agent does not construct terms, therefore, all possible state transitions for a computer agent are derivable from the protocol specification.

Channel Goals Revisited

In our human-error model, we examine the same channel goals as in our Secure Platform Problem model defined in Chapters 5 and 7. That is, confidentiality (Definition 5.5), authenticity (Definition 5.6), and individual verifiability (Definition 7.1). However, since the human communication end point now is the user's brain, with respect to human error, authenticity has a different meaning. Recall that in Chapter 5, we defined communication of knowledge as a trace property where for a sender S, a receiver R, and a message m a corresponding $\mathsf{Comm}(S, m)$ action occurs before a $\mathsf{Learn}(R, m)$ action in a trace. With respect to human errors, a $\mathsf{Comm}(S, m)$ action specifies the expected communication that the honest user is intended to perform. That is, we analyze the channel from the user's brain to a remote communication partner. But since we allow the human user to deviate from the specified protocol according to the failure modes identified in Chapter 8, he may send a different message m'. In this case, even if the communication channel itself provides authenticity with respect to our Secure Platform Problem model above, the message received ($\mathsf{Learn}(R, m')$) deviates from the message that was intended to be sent and therefore, authenticity does not hold with respect to our human error model. Example 9.2 illustrates this.

Example 9.2 *Consider a simple one-step protocol, where a sender S sends a message m to a receiver R using an authentic channel. If S is allowed to confuse m with another message m′, R would receive m′ instead of m and authenticity would not hold.*

In the next section, we define our protocol specification language for analyzing the impact of human errors on a protocol's security properties.

9.2 Protocol Specification

We extend the protocol specification requirements in Section 5.3 in the following way. The setup rule containing an $\mathsf{AgentState}(A, c, n)$ fact must contain one of the actions $\mathsf{Human}(A)$, or $\mathsf{Computer}(A)$. $\mathsf{Human}(A)$ indicates that an agent A is human, $\mathsf{Computer}(A)$ indicates that the agent is a computer. Recall that agents cannot be both human and computer and that agents cannot change between being human and computer. In the following, we highlight the extensions in bold face. Formally, a setup rule $l -[\, a \,]\!\to r$ is a rule where:

S1 Only Fresh and Fr facts occur in l.

S2 For every $\mathsf{AgentState}(A, \_, \_)$ fact in r, there is an $\mathsf{Agent}(A)$, $\mathsf{Honest}(A)$, or $\mathsf{Dishonest}(A)$ action in a.

S3 **For every $\mathsf{AgentState}(A, \_, \_)$ fact in r, there is a $\mathsf{Human}(A)$ or $\mathsf{Computer}(A)$ action in a.**

S4 **For every subterm n' of n in an $\mathsf{AgentState}(A, \_, n)$ fact where A is marked with a $\mathsf{Human}(A)$ action in a, there is a $\mathsf{!HK}(A, n')$ fact in r.**

S5 **For all $\mathsf{AgentState}(A, c, \_)$ facts in the protocol specification $\mathcal{R}_{Spec}$, there is a $\mathsf{!HK}(A, c)$ fact in r.**

A protocol specification may contain exactly one failure rule. In this failure rule, we use an arbitrary number of **Fail** facts to model erroneous human behavior. A failure rule $l -[\ a\ ]\!\rightarrow r$ is a rule where:

F1 l is empty.

F2 One $\mathsf{Failure}()$ action occurs in a.

F3 Only **Fail** facts occur in r.

Additionally to the trace restrictions defined above in Section 9.1, we restrict the set of traces $TR(\mathcal{R})$ to the traces that satisfy the following property:

$$\forall tr \in TR(\mathcal{R}), tr', tr'' \in \mathcal{P}(\mathcal{G})^* : tr = tr' \cdot tr'' \wedge \mathsf{Failure}() \in tr' \implies \mathsf{Failure}() \notin tr''$$

With this additional trace restriction, the failure rule appears only once in a trace. This limits the number of failures in a trace to exactly those specified in the failure rule.

Still, we only allow protocols where after the setup phase all information is exchanged using the channels defined in our channel abstraction model. That is, information may not flow from one agent to another in any way other than by one of the channels defined in $\mathcal{CHE}$.

Whenever a human agent is expected to compare two terms m and m', this must be explicitly specified. In the protocol rule where the term is assumed to be compared, a $\mathsf{Compare}(m, m')$ fact is added to the conclusion. As a premise for the subsequent protocol rule, a $\mathsf{Verified}(m, m')$ fact must be added.

A protocol rule $l -[\ a\ ]\!\rightarrow r$ is a rule such that the following 6 conditions are satisfied.

P1 The facts in l, a, and r do not contain elements of $\mathcal{C}_{\text{fresh}}$ as subterms.

P2 Only Rcv, Fresh, **Verified**, and $\mathsf{AgentState}$ facts occur in l.

P3 Only Snd, **Compare**, **!HK**, and $\mathsf{AgentState}$ facts occur in r.

P4 Exactly one $\mathsf{AgentState}$ fact occurs in l, zero or more $\mathsf{AgentState}$ facts occur in r.

P5 If $\mathsf{AgentState}(A, c, n)$ occurs in l, then

a) every Rcv and Fresh fact is of the form $\mathsf{Rcv}(p, B, A, x)$ and $\mathsf{Fresh}(A, x)$ where $p \in \{\mathsf{I}, \mathsf{A}, \mathsf{C}, \mathsf{S}\}$ and $B, x \in \mathcal{T}$,

b) **for every $\mathsf{Rcv}(p, B, A, m) \in l$ where A is a human agent's name, all subterms of m are different from all subterms in n.**

c) **for every subterm m' of m in a $\mathsf{Rcv}(p, B, A, m)$ fact where A is a human agent's name, there is a $\mathsf{!HK}(A, m')$ fact in r.**

d) every Learn, Comm, Trust, Secret, Authentic, Verify, and Snd fact is of the corresponding form $\mathsf{Learn}(A, x)$, $\mathsf{Comm}(A, x)$, $\mathsf{Trust}(B)$, $\mathsf{Secret}(A, B, x)$, $\mathsf{Authentic}(B, A, x)$, $\mathsf{Verify}(A, B, x)$, and $\mathsf{Snd}(p, A, B, x)$, where $p \in \{\mathsf{I}, \mathsf{A}, \mathsf{C}, \mathsf{S}\}$, $B \in \mathcal{V}_{\mathsf{pub}}$, $x \in \mathcal{T}$ and x is derivable from terms in $\mathcal{C}_{\mathsf{pub}}$, terms in Fresh and Rcv facts occurring in l, and terms in n.

e) every AgentState fact in r is of the form $\mathsf{AgentState}(A, c', n')$, where $c' \in \mathcal{C}_{\mathsf{pub}}$ and n' is derivable from terms in $\mathcal{C}_{\mathsf{pub}}$, terms in Fresh, and Rcv facts occurring in l, and terms in n.

f) **every !HK fact is of the form $\mathsf{!HK}(A, x)$, where x is derivable from terms in Fresh, and Rcv facts occurring in l.**

g) **for A representing a computer agent's name, an arbitrary number of $\mathsf{Sysfail}(A, c, n, c', n')$ facts may occur in r. Where, c' and n' refer to another $\mathsf{AgentState}(A, c', n')$ fact occurring in the premise of a rule in $\mathcal{R}_{Spec}$.**

P6 **For every $\mathsf{Snd}(p, B, A, x)$ and $\mathsf{Rcv}(p, A, B, x)$ fact, where A denotes a human agent, x is of the form $\langle n, m \rangle$, with $n \in \mathcal{C}_{\mathsf{pub}}$ is an identifier and $m \in \mathcal{V}_{\mathsf{pub}}$ is the actual message.**

P7 $vars(r) \subseteq vars(l) \cup \mathcal{V}_{\mathsf{pub}}$.

The above extensions allow one to examine systems where humans, computers, and servers communicate in more detail regarding erroneous human user behavior. We model this behavior with additional rewrite rules.

For ease of reading, in our extended Alice & Bob notation introduced in Section 5.3, we write $A : \mathsf{compares}(m, m')$ to express that an agent executing role A compares the two terms m and m'. Moreover, we write $\overset{\mathsf{h}}{\rightarrow\!\!\!\!\!\rightarrow}$ for human-interaction channels. For example, $A \overset{\mathsf{h}}{\rightarrow\!\!\!\!\!\rightarrow} B : m$ expresses, that a message m is to be sent from an agent executing role A to an agent executing role B and that either A or B describes a human role.

Protocols specified as introduced in this section may now be formally analyzed using the methods and tools we describe in the next section.

9.3 Human Error Analysis

In this section we introduce a formal analysis method and tools to examine the security properties of secure communication applications in the presence of erroneous human users. The analysis method allows one to examine a protocol's robustness against

individual human errors and combinations thereof. For example, the human user may be assumed to confuse channels only but to perform as specified with respect to the remaining failure modes. With this respect, we focus on individual human errors, i.e., we assume that the user accidentally deviates from the protocol specification but that he does not fail repeatedly.

Individual Human-Error Robustness

Formally, Failures (a) – (o) are represented by the sets $\mathcal{E}_{\text{failureID}}$ of *human-error rules* as described in Table 9.2. These human-error rules are multiset rewriting rules that model a human agent's capability to deviate from the protocol specification accordingly. We define the human-error rules below in this section. Table 9.2 summarizes the failure modes identified in Section 8.3, and shows how they are covered by the human-error rules in our human-error model. We define $\mathcal{E}_{all}$ as the set of all human-error rules and F_{ID} as the set of all failure identifiers. The failure identifiers in F_{ID} are described in Table 9.1.

$failureID$	Description
msc	Message sending confusion.
csc	Channel sending confusion.
mrc	Message receiving confusion.
nlc	No learning confusion.
crc	Channel receiving confusion.
icc	Ignoring comparison confusion
wrc	Weak randomness confusion.
soc	Step order confusion.
src	Step repetition confusion.

Table 9.1: Overview of the failure identifiers in F_{ID}.

For all rules $l -[\, a \,]\!\rightarrow r \ \in \mathcal{E}_{all}$, l includes a Fail fact. As described above in Section 9.2, the failure rule may include an arbitrary number of Fail facts in its conclusion. Hence, to analyze a protocol's strength with respect to a specific set of human errors, the protocol specification must include the corresponding Fail facts in the failure rule.

Formally, with respect to the channel properties defined in Section 5.2, we define a protocol's robustness against a specific single human-error as follows.

Definition 9.3 *A protocol $\mathcal{R}$ is $(A, failureID)$-robust with respect to property (p, q), if all channels provided by R having the (p, q) property, these channels have the (p, q) property in the protocol $\mathcal{R}' \cup \mathcal{E}_{all}$. $\mathcal{R}'$ represents the original protocol $\mathcal{R}$ extended with a* $\mathsf{Fail}(A, failureID)$ *fact in the corresponding failure rule for human user A in $\mathcal{R}_{Spec}$, as defined in Section 9.2.*

The definition states that a protocol is robust against a single human error if its security properties are independent of the erroneous human behavior specified by

the corresponding human-error rules. We introduce combinations and repetitions of human errors later at the end of this section.

Next, we first define the human-error rules corresponding to the failure modes introduced in Chapter 8. We define a human-error ruleset for individual failure modes and we provide corresponding examples. Moreover, we derive necessary or sufficient conditions for protocols to be robust against individual human errors. Later in Section 9.5 we provide a systematic human-error analysis process to verify a given protocol's robustness against individual human errors and specific combinations thereof. Although our definition above allows one to analyze protocols with more than one human agent involved, in the remainder of this part of the thesis, we focus exclusively on protocols with just one human agent involved. Therefore, whenever we state that a protocol is $(A, failureID)$-robust with respect to a certain channel property, we assume that A is executed by the human agent.

Failure	Description	Rules
(a)	Not applying a rule at all. Modeled implicitly.	-
(b)	Sending an arbitrary term from knowledge.	$\mathcal{E}_{\text{msc}}$
(c)	Sending a term on the wrong channel.	$\mathcal{E}_{\text{csc}}$
(d)	Learning an arbitrary public or fresh term instead of the received term.	$\mathcal{E}_{\text{mrc}}$
(e)	Learning no term at all.	$\mathcal{E}_{\text{nlc}}$
(f)	Learning a term from the wrong channel.	$\mathcal{E}_{\text{crc}}$
(g)	Ignoring comparison of terms.	$\mathcal{E}_{\text{icc}}$
(h)	Modeled implicitly with the rules in $\mathcal{HK}$.	-
(i)	Modeled implicitly with the rules in $\mathcal{HK}$.	-
(j)	Generating weak fresh terms.	$\mathcal{E}_{\text{wrc}}$
(k)	Forgetting a term is equals to not using it for sending or comparing. Covered by the rules in $\mathcal{E}_{\text{msc}}$ and $\mathcal{E}_{\text{icc}}$.	-
(l)	Confusing knowledge affects sending and comparing terms. Covered by the rules in $\mathcal{E}_{\text{msc}}$ and $\mathcal{E}_{\text{icc}}$.	-
(m)	Ignoring a non-blocking protocol step.	$\mathcal{E}_{\text{soc}}$
(n)	Executing a protocol step repeatedly.	$\mathcal{E}_{\text{src}}$
(o)	Confusing the expected order of the protocol steps is modeled by repeated application of the rule in $\mathcal{E}_{\text{soc}}$	-

Table 9.2: Summary of the failure modes and their formal representation as human-error rules.

In the following, we define the human-error rules corresponding to the failure modes identified in Section 8.3. Table 9.2 summarizes these failure modes and their representation in our model. We provide practical examples for the human-error rules and we show the rules' interrelations regarding their impacts on a communication application's security properties. For example, there is no difference if a user is inat-

tentive while receiving a term, thereby not learning the term, or if he learns the correct term but forgets it before he is supposed to use it. We derive necessary and sufficient conditions for protocols to be robust against individual human errors.

We group the human errors into six areas: *sending confusion*, *learning confusion*, *comparing confusion*, *knowledge confusion*, *weak randomness*, and *protocol-step confusion*. We introduce the first five areas in the remainder of this section. Protocol-step confusion will be covered in Section 9.4 since these human errors require other protocol agents to misbehave too.

Sending confusion.

We first examine how human agents input information into communication applications. That is, how they send individual messages to a remote communication partner. With this respect, humans may send wrong terms. For example, they may mistype terms or they may confuse them with other terms from their knowledge. Moreover, they may confuse channels and thereby sending terms using the wrong channel. That is, whenever they are supposed to send a specific term, they may not send the term as in Failure (a), causing the protocol to block, they may use any known term instead of the specified one, which corresponds to Failure (b), or they may send the expected term using another channel than specified, which corresponds to Failure (c). Note that the chosen term may be constructed using concatenation and selection.

Agents who do not send a term when supposed to do so are modeled implicitly. All traces of a protocol $\mathcal{R}$ where the corresponding rule is not applied are included in the set of all traces of the protocol $TR(\mathcal{R})$.

To model users who send arbitrary terms instead of the specified ones, we define the following human-error rule in $\mathcal{E}_{\text{msc}}$.

$$\mathcal{E}_{\text{msc}} := \{\ [\mathsf{Fail}(A, msc), !\mathsf{HK}(A, m'), \mathsf{Snd}(p, A, B, m)] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Snd}(p, A, B, m')]\ \} \tag{9.15}$$

In Rule (9.15), agent A may send a message m' from his knowledge to a receiver B using a channel from A to B with property p, if he is supposed to send a message m using this channel.

The following two practical examples show the application of our analysis approach. The first example is concerned with the HTTP Digest Access Authentication Scheme introduced in Section 8.3.

Example 9.4 *The HTTP Digest Access Authentication Scheme in Example 8.1 aims to avoid that the password is sent over the network in cleartext. Obviously, if the human agent may send the password instead of the username, i.e., if he enters the password into the textfield, where the username would be expected, the password will be sent in cleartext as we show in Proposition 9.5.*

Proposition 9.5 *For human user A, server B, and A's password p the HTTP Digest Authentication Scheme is not (A, msc)-robust with respect to the confidentiality property (p_{conf}, q_{conf}) regarding p.*

Proof We prove this using our formal human error model in the Tamarin tool. The Tamarin tool proves confidentiality for the protocol but finds a trace where confidentiality does not hold if the human agent H is allowed to confuse terms, i.e., if a $\mathsf{Fail}(H, msc)$ fact is given in the failure rule.

In the protocol step where human agent H sends the username, Rule (9.15) is being applied and a Snd fact for the password is produced. Hence, the human agent's browser sends the password in cleartext as part of the authentication header using the insecure channel to the server and the adversary learns the password. □

A fix to make the HTTP Digest Access Authentication Scheme robust against sending confusion is to use hashes of usernames. If the human agent erroneously enters the password instead of the username, the hash of the password would be sent.

The next two examples show that while simple code voting is not robust against human agents who confuse messages, SureVote is.

Example 9.6 Simple code voting *is a concept based on codebook cryptography and applied for Internet voting as described above in Chapter 7. Since we focus on human error, we omit modeling the code-sheet as a trusted device, rather we assume that the voter knows the code-sheet. Figure 9.3 provides the protocol in our extended Alice & Bob notation.*

$$\begin{array}{rcl} H & : & \mathrm{knows}(\langle S, c, \mathrm{h}(c), c', \mathrm{h}(c')\rangle) \\ S & : & \mathrm{knows}(\langle H, c, \mathrm{h}(c), c', \mathrm{h}(c')\rangle) \\ H \; \mathrm{h}\!\!\rightarrow\!\!\!\!\!\rightarrow\mathrm{h} \; P & : & \mathrm{h}(c) \;/\; \mathit{code} \\ P \circ\!\!\rightarrow\!\!\!\!\!\rightarrow\circ \; S & : & \mathit{code} \;/\; \mathrm{h}(c) \end{array}$$

Figure 9.3: Simple code voting.

Prior to the election, voter H receives a code-sheet containing the candidate names c and c', and the corresponding codes $\mathrm{h}(c)$ and $\mathrm{h}(c')$, respectively. Since the code-sheets are distributed out-of-band, it is assumed that the network adversary does not learn them. To communicate a vote for candidate c, H enters the corresponding code $\mathrm{h}(c)$ into his computer. This code is then submitted to the election authority S. Since the election authority created the code-sheets, it can map the code back to the corresponding candidate.

In the following Proposition 9.7, we show that simple code voting is not robust against users who confuse message.

Proposition 9.7 *The simple code voting protocol is not (A, msc)-robust with respect to the authenticity property (p_{auth}, q_{auth}).*

Proof We use the Tamarin tool to prove the proposition. Tamarin identifies a trace where authenticity does not hold. The simple code voting protocol's specification includes a rule with a $\mathsf{Snd}(\mathsf{HI}, H, P, c)$ fact, where HI indicates human agent H's access to his computer P. c is the code corresponding to the candidate's name H intends to vote for. H knows c ($!\mathsf{HK}(H, c)$ as well as c' ($!\mathsf{HK}(H, c')$), where c' represents the code

for another candidate. H can apply Rule (9.15) to rewrite the initial $\mathsf{Snd}(\mathsf{HI}, H, P, c)$ fact to $\mathsf{Snd}(\mathsf{HI}, H, P, c')$ and authenticity does not hold, since H communicated c, i.e., $\mathsf{Comm}(H, c) \in tr$, but the voting authority receives c', hence, $\mathsf{Learn}(S, c') \in tr$. □

With respect to confidentiality, simple code voting is not (A, msc)-robust, either. We show this in the next proposition 9.8.

Proposition 9.8 *The simple code voting protocol is not (A, msc)-robust with respect to the confidentiality property (p_{conf}, q_{conf}).*

Proof We use the Tamarin tool to prove the proposition. Tamarin identifies a trace where confidentiality does not hold. Analogously to the trace found in Proposition 9.7, H again applies Rule (9.15). Instead of the chosen candidate's code $\mathrm{h}(c)$, H enters the candidate's name c, which in turn is sent by P using the insecure channel to S and thus, the adversary learns the voter's choice. □

SureVote, as introduced in Example 9.1 above, is not robust against humans confusing messages either. We prove this in the next propositions.

The voter may leak the chosen candidate instead of his code. Therefore, SureVote is not robust against sending confusion with respect to confidentiality.

Proposition 9.9 *The SureVote Internet voting protocol is not (A, msc)-robust with respect to the confidentiality property (p_{conf}, q_{conf}).*

Proof We use the Tamarin tool to prove that the confidentiality property does not hold. Tamarin finds a trace where instead of sending the chosen candidate's voting code, Rule 9.15 is applied and the candidate's name itself is sent. That is, instead of the candidate code, the user enters the candidate's name and thus, he leaks his choice. □

Although the voter receives a verification code to verify the vote that was recorded by the election authority, if the voter confuses the candidate codes the election authority receives a vote for another candidate than intended by the voter. Hence, authenticity does not hold.

Proposition 9.10 *The SureVote Internet voting protocol is not (A, msc)-robust with respect to the authenticity property (p_{auth}, q_{auth}).*

Proof We use the Tamarin tool to prove that the authenticity property does not hold. Tamarin finds a trace where instead of sending the chosen candidate's voting code, Rule 9.15 is applied and another candidate's code is sent. The election authority in turn records a vote for the other candidate. □

In the next proposition we prove that verifiability does not hold, if the voter confuses messages.

Proposition 9.11 *The SureVote Internet voting protocol is not* (A, msc)*-robust with respect to the verifiability property* (p_{verif}, q_{verif}).

Proof Using the Tamarin tool, we find a trace where instead of sending the chosen candidate's voting code, Rule 9.15 is applied and the verification code is sent. The voter's computer sends the verification code along using the insecure channel to the election authority's server. Hence, the adversary learns the verification code and sends it back to the voter's computer and the voter's computer displays it to the voter who verifies the verification code to be valid although no message was received by the election authority. □

With respect to authenticity, verifiability can be used to protect against humans confusing messages. Even if the user accidentally sends the wrong message, due to verification information that the server sends back to the user, he notices that the wrong term was sent. However, the user must additionally be given the option to confirm or invalidate the previously sent message based on the verification's outcome. An example to illustrate this is Helbach and Schwenk's modification [46] of SureVote. Additionally to the voting codes and verification codes, they introduce finalization codes. After successful verification of the verification code, the voter sends the finalization code back to the election authority to confirm the correctness. In Figure 9.4 we provide the protocol in our extended Alice & Bob notation.

$$
\begin{array}{rcl}
H & : & \text{knows}(\langle S, c, \text{h}(c), \text{h}(\langle k, c\rangle), \text{h}(\langle k', c\rangle), c', \text{h}(c'), \text{h}(\langle k, c'\rangle), \text{h}(\langle k', c'\rangle)\rangle) \\
S & : & \text{knows}(\langle H, c, \text{h}(c), \text{h}(\langle k, c\rangle), \text{h}(\langle k', c\rangle), c', \text{h}(c'), \text{h}(\langle k, c'\rangle), \text{h}(\langle k', c'\rangle)\rangle) \\
H \overset{\text{h}}{\mapsto} P & : & \text{h}(c) \ / \ code \\
P \overset{\circ}{\to} S & : & code \ / \ \text{h}(c) \\
S \overset{\circ}{\to} P & : & \text{h}(\langle k, c\rangle) \ / \ verificationcode \\
P \overset{\text{h}}{\mapsto} H & : & verificationcode \\
H & : & \text{compares}(\text{h}(\langle k, c\rangle), verificationcode) \\
H \overset{\text{h}}{\mapsto} P & : & \text{h}(\langle k', c\rangle) \ / \ finalizationcode \\
P \overset{\circ}{\to} S & : & finalizationcode \ / \ \text{h}(\langle k', c\rangle)
\end{array}
$$

Figure 9.4: Helbach code voting variant.

The additional finalization step makes the Helbach code voting variant robust against message sending confusion with respect to authenticity. We prove this in the next proposition.

Proposition 9.12 *The Helbach code voting variant is* (A, msc)*-robust with respect to the authenticity property* (p_{auth}, q_{auth}).

Proof We use the Tamarin tool to prove that the authenticity property holds. That is, whenever the election authority receives a valid finalization code, the previously received candidate code corresponds to the candidate the voter indeed intended to vote for. □

In general, to be robust against humans confusing messages with respect to confidentiality, it is sufficient if the protocol does only use secure channels. In that case, even if the human user accidentally sends a confidential message, the adversary will not learn the confidential message. Contrary, if a non-confidential channel exists, the confused confidential message may be sent using this channel and the adversary learns it or, in the case of non-authentic channels, the adversary may inject a message which may later be confused by the user with a confidential message. The next theorem proves this sufficiency condition in our model.

Theorem 9.13 *For a protocol $\mathcal{R}$ to be (A, msc)-robust with respect to (p_{conf}, q_{conf}), it is sufficient that for every $\mathsf{Snd}(p, B, C, m)$ and $\mathsf{Rcv}(p, B, C, m)$ facts specified in the rules of $\mathcal{R}_{Spec}$, $p \in \{HI, S\}$.*

Proof The proof is straightforward since in our human-error model the only way for the adversary to learn a fresh term m' sent by human agent A is using Out facts produced by Rules 9.4 and 9.6. In order to be executed, these rules need corresponding $\mathsf{Snd}(p, S, R, m)$ facts to be specified in the rules in $\mathcal{R}_{Spec}$, with $p = I$ or $p = A$, respectively. Hence, if for all rules $r \in \mathcal{R}_{Spec}$ no such Snd fact is specified, then for all traces $tr \in TR(\mathcal{R})$ we have $\mathsf{Out}(m') \notin tr$. Therefore, the adversary may never learn the term m'. The only way the adversary may inject a known term m'', is to use In facts consumed by Rules 9.5 and 9.10. In order to be executed, these rules need corresponding $\mathsf{Rcv}(p, S, R, m)$ facts to be specified in the rules in $\mathcal{R}_{Spec}$, with $p = I$ or $p = C$, respectively. Hence, if for all rules $r \in \mathcal{R}_{Spec}$ no such Rcv fact is specified, then for all traces $tr \in TR(\mathcal{R})$ we have $\mathsf{In}(m'') \notin tr$. Therefore, if $\mathcal{R}$ satisfies $(p_{\text{conf}}, q_{\text{conf}})$ and only $\mathsf{Snd}(p, S, R, m)$ and $\mathsf{Rcv}(p, S, R, m)$ facts with $p \in \{HI, S\}$ are specified in $\mathcal{R}_{Spec}$, $(p_{\text{conf}}, q_{\text{conf}})$ must hold in $\mathcal{R}' \cup \mathcal{E}_{all}$, where $\mathcal{R}'$ represents $\mathcal{R} = \mathcal{R}_{Model} \cup \mathcal{R}_{Spec}$ extended with a $\mathsf{Fail}(A, msc)$ fact in the corresponding failure rule for human agent A. □

Next, we define the human-error rule in $\mathcal{E}_{\text{csc}}$ to model human agents who confuse channels, as it is the case in Failure (c).

$$\mathcal{E}_{\text{csc}} := \{\ [\mathsf{Fail}(A, csc), \mathsf{Snd}(p, A, B, m), \mathsf{Snd}(p', A, C, m')] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Snd}(p, A, B, m), \mathsf{Snd}(p', A, C, m)]\ \} \tag{9.16}$$

In Rule (9.16), whenever human agent A is supposed to send message m to receiver B using a channel with property p and a message m' to receiver C using a channel with property p', A may confuse the channels and send message m to receiver C using the channel with property p'.

A protocol that is robust against human agents who may send arbitrary terms from their knowledge instead of the specified ones is also robust against human agents who may confuse channels. With respect to our model, human agents who may send arbitrary terms from their knowledge may send the message expected to be sent on the one channel on another channel. We show this in Theorem 9.14.

Theorem 9.14 *Every protocol $\mathcal{R}$ that is (A, msc)-robust is (A, csc)-robust.*

Proof We show that confusing channels is a special case of confusing messages. That is, all protocol runs where channels are confused can be simulated by confusing messages. Let protocol $\mathcal{R}$ specify a human user A accessing two different channels to send messages m and m', i.e., protocol rules that include $\mathsf{Snd}(p, A, B, m)$ and $\mathsf{Snd}(p', A, C, m')$ facts in their conclusion. Now let $\mathcal{R}''$ be the protocol $\mathcal{R}$, extended with a $\mathsf{Fail}(A, csc)$ fact in the failure rule and $\mathcal{R}'$ be the protocol $\mathcal{R}$, extended with a $\mathsf{Fail}(A, msc)$ fact in the failure rule. In every protocol run of $\mathcal{R}'' \cup \mathcal{E}_{all}$ where Rule (9.16) is applied, the $\mathsf{Snd}(p', A, C, m')$ fact is replaced with a $\mathsf{Snd}(p', A, C, m)$ fact. Since m and m' must be in A's knowledge in order to be sent to B and C, respectively, for each of these protocol runs, there exists a corresponding protocol run of $\mathcal{R}' \cup \mathcal{E}_{all}$ where Rule (9.15) is applied to replace the $\mathsf{Snd}(p', A, C, m')$ fact with a $\mathsf{Snd}(p', A, C, m)$ fact. Therefore, if $\mathcal{R}$ is (A, msc)-robust, $\mathcal{R}$ is also (A, csc)-robust. □

Due to the unexpected message received, the computer agent may block and the communicated message will not be learned by the specified receiver. However, sending a secret using a different channel than specified could allow the adversary to learn the confidential message anyway. The following example illustrates this.

Example 9.15 *Consider the case where a user accesses web applications using his computer's browser with several tabs opened. In one browser tab, the user accesses an online banking application secured with an encrypted connection, in another tab the user accesses a simple web application which is not secured. In both applications, the human agent is supposed to enter his credentials. The user stores his credentials for both applications in a password safe from which he receives his password for the online banking application. Next, he pastes the password into the corresponding password entry field of the simple web application instead of the banking application's. Unless the credentials for both applications are the same, the user will not be able to successfully log into the insecure application. However, the adversary may intercept the password on the insecure channel and thus confidentiality does not hold.*

It is obvious that for a protocol providing different channels to a human agent, providing just one channel at a time is a sufficient condition for robustness against human agents that confuse channels. To do so in practice, protocol agents need to be synchronized, i.e, state information must be exchanged between the protocol agents. In our example above, the browser must not allow two browser tabs to be opened with text input fields. However, such restrictions are unrealistic and human users are required to comply with the protocol specification with respect to channel confusion.

So far we analyzed human error based on message sending. Next, we examine human errors with respect to learning messages.

Learning confusion.

Learning confusion concerns a human agent's perception of information. The human agent may confuse messages he receives during a protocol run. In contrast to sending confusion, the specified term is never known to the human agent and thus, it can

never be used by the human agent during the protocol run. Learning confusion may be caused by learning a different term, as it is the case in Failure (d), or by omitting learning the term at all, like in Failure (e). Human agents may also confuse incoming channels, i.e., they may learn messages from a different channel than specified, as in Failure (f).

We model human agents who learn arbitrary different terms instead of the specified ones with the two rules in $\mathcal{E}_{\text{mrc}}$.

$$\mathcal{E}_{\text{mrc}} := \{ \quad [\mathsf{Fail}(A, mrc), \mathsf{Rcv}(p, B, A, m)] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Rcv}(p, B, A, m' : pub)], \tag{9.17}$$

$$[\mathsf{Fail}(A, mrc), \mathsf{Rcv}(p, B, A, m), \mathsf{Fr}(m')] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Rcv}(p, B, A, m')] \ \} \tag{9.18}$$

In Rule (9.17), whenever the human agent receives a term, this term may be replaced with any public term. Considering the case, where the human agent learns a fresh term incorrectly, the specified term can also be replaced with an new fresh term. We model this with Rule (9.18).

Rule (9.18) also covers the case where the human agent does not learn any term at all. If the human agent is supposed to use the received term later in the protocol run, he can just use the fresh term instead of the specified one or he may block and not execute execute the protocol step. Therefore, we model this failure using the same human error rule in $\mathcal{E}_{\text{nlc}}$.

$$\mathcal{E}_{\text{nlc}} := \{\ [\mathsf{Fail}(A, nlc), \mathsf{Rcv}(p, B, A, m), \mathsf{Fr}(m')] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Rcv}(p, B, A, m')]\ \} \tag{9.19}$$

It is obvious that a protocol that is robust against human agents who learn a different term instead of the specified one are also robust against human agents who do not learn any term at all. We state this with the following Theorem 9.16.

Theorem 9.16 *Every protocol $\mathcal{R}$ that is (A, mrc)-robust with respect to a property (p, q) is (A, nlc)-robust with respect to the same property (p, q).*

Proof We show that not learning a message is a special case of learning an arbitrary different message. Let protocol $\mathcal{R}$ specify a human user A receiving a message m, i.e., a protocol rule including a $\mathsf{Rcv}(p, B, A, m)$ fact in its premise. Now let $\mathcal{R}''$ be the protocol $\mathcal{R}$, extended with a $\mathsf{Fail}(A, nlc)$ fact in the failure rule and $\mathcal{R}'$ be the protocol $\mathcal{R}$, extended with a $\mathsf{Fail}(A, mrc)$ fact in the failure rule. In every protocol run of $\mathcal{R}'' \cup \mathcal{E}_{all}$ where Rule (9.19) is applied, the $\mathsf{Rcv}(p, B, A, m)$ fact is replaced with a $\mathsf{Rcv}(p, B, A, m')$ fact, where m' is fresh. For each of these protocol runs, there exists a corresponding protocol run of $\mathcal{R}' \cup \mathcal{E}_{all}$ where Rule (9.18) is applied to replace the $\mathsf{Rcv}(p, B, A, m)$ fact with a $\mathsf{Rcv}(p, B, A, m')$ fact, where m' is fresh. Therefore, if $\mathcal{R}$ is (A, msc)-robust with respect to property (p, q), then $\mathcal{R}$ is also (A, csc)-robust with respect to property (p, q). □

Human agents may confuse terms provided on different channels from different senders or from the same sender. Differently than our modeling approach above regarding sending confusion of messages, the act of reading different codes written on a paper sheet as it is the case in code voting, could be modeled as different communication channels. To do so, all terms sent and received to and from the human agent would be modeled separately. The next example shows why we need an additional specification requirement with respect to examine a protocol's robustness against Failure (f).

Example 9.17 *Consider a code-sheet for simple code voting. The human agent reads the codes for each of the candidates from the code-sheet. In the protocol specifications above in Part I we used to specify reading the candidate name and the corresponding code all at once. With respect to learning confusion, this makes it impossible to model the case where the human agent confuses the codes when reading them, e.g., due to accidentally shifting lines.*

Remember protocol specification requirement 9.2. That is, whenever a human user receives a message m, the corresponding Snd and Rcv facts are marked with an identifier describing the context of m.

We model human agents who confuse incoming channels with the following human-error rule in $\mathcal{E}_{\text{crc}}$.

$$\begin{aligned}\mathcal{E}_{\text{crc}} := \{\ [\mathsf{Fail}(A, crc), \mathsf{Rcv}(p, B, A, \langle n, m\rangle), \mathsf{Rcv}(p', C, A, \langle n', m'\rangle)] \\ -[\ \mathsf{Human}(A)\ ]\mapsto \\ [\mathsf{Rcv}(p, B, A, \langle n, m'\rangle), \mathsf{Rcv}(p', C, A, \langle n', m\rangle)]\ \}\end{aligned} \tag{9.20}$$

In Rule (9.20), we model that whenever human agent A has the option to receive messages m and m', he may confuse the channels, that is, he may remember m' in the context where he is supposed to remember m and vice versa.

Note that learning confusion usually leads to blocking conditions since the human agent is not able to provide the proper input expected by the receiver. That is, learning confusion is often concerned with availability issues. Consider for example a service, where the human agent is supposed to register for, and thereby, receiving a new password. He may fail to remember his password correctly. Hence, he will not have access to the service, which concerns availability.

We do not cover availability in this thesis in more detail. However, examining necessary or sufficient conditions to ensure availability with respect to human errors would be an interesting future research question.

In the following, we examine the cases where humans fail to correctly compare messages.

Comparing confusion.

In this class, the human agent mainly follows the protocol specification but does not perform comparisons, as it is the case in Failure (g). We model this erroneous behavior with the human-error rule in $\mathcal{E}_{\text{icc}}$. It allows the human agent to omit comparing terms

when he is supposed to do so. For example, in transaction authentication protocols, a lazy user may omit checking a transaction's correctness.

$$\mathcal{E}_{\text{icc}} := \{\ [\mathsf{Fail}(A, icc), \mathsf{Compare}(A, m, m')] \dashv [\ ] \mapsto [\mathsf{Verified}(A, m, m')]\ \} \tag{9.21}$$

Rule (9.21) allows traces where correct comparison of terms is omitted, i.e., where human agent A does not compare a received term with another term in his knowledge. Remember that we restrict the set of traces to those satisfying, that whenever an $\mathsf{Eq}(m, m')$ action occurs in the trace, then $m = m'$.

In the next examples we show what impact this erroneous behavior may have.

Example 9.18 *Consider an online banking solution where the bank does not trust their customers' computers. Hence, they distribute trusted devices to them to provide transaction authentication, i.e., every transaction performed by the customer must be explicitly verified and confirmed by the customer. The trusted device has a display and a keypad. The customer uses the bank's web application to enter the intended transaction details. The web application sends the transaction instructions (e.g., recipient and amount) to the trusted device which in turn displays the details to the customer. The customer is now supposed to verify the transaction's correctness as presented by the trusted device and to confirm his consent by pressing a button on the device. If the customer does not verify the comparison correctly, a customer's malicious computer could have modified the transaction instruction. Hence, authenticity is not given.*

In the next example, we introduce a common device-pairing protocol that is not robust against comparing confusion.

Example 9.19 *For Bluetooth communication,* Secure Simple Pairing (SSP) *is intended to simplify the wireless device pairing procedures for users and is specified in the Bluetooth specification [15]. Its security goals is to protect against passive eavesdropping and against man-in-the-middle attacks. SSP uses four association models depending on a device's capabilities. We introduced different devices and their capabilities above in Section 4.2. For example a device may have a display, or a keypad, or both, or none, etc. One of the association models is based on* Numeric Comparison*. Figure 9.5 gives a simplified overview of the Numeric Comparison SSP in our extended Alice & Bob notation. The original protocol includes a commitment phase. For our purposes the simplification is sufficient.*

First, the two devices P and S exchange their public keys $\mathrm{pk}(ltk_P)$ *and* $\mathrm{pk}(ltk_S)$*, then they exchange fresh nonces* N_P *and* N_S*. Each device computes a six digit number* V_P *and* V_S *using public function* $\mathrm{g}(\mathrm{pk}(ltk_P), \mathrm{pk}(ltk_S), N_P, N_S)$*. On both devices P and S the six digit number is displayed. The human user S in turn is supposed to compare these numbers and to confirm their equality. It is obvious that if H omits the comparison the adversary could successfully mount a man-in-the-middle attack. We show next, that the this association model is not robust against comparing confusion.*

Proposition 9.20 *The Numeric Comparison association model of Bluetooth Secure Simple Pairing is not* (A, icc)*-robust with respect to the authenticity property* (p_{auth}, q_{auth})*.*

$$
\begin{array}{rcl}
P & : & \mathrm{knows}(ltk_P) \\
S & : & \mathrm{knows}(ltk_S) \\
P \circ\!\!\rightarrow\!\!\circ\, S & : & \mathrm{pk}(ltk_P) \\
S \circ\!\!\rightarrow\!\!\circ\, P & : & \mathrm{pk}(ltk_S) \\
P \circ\!\!\rightarrow\!\!\circ\, S & : & \mathrm{fresh}(N_P).Np \\
S \circ\!\!\rightarrow\!\!\circ\, P & : & \mathrm{fresh}(N_S).Ns \\
P \hookrightarrow\!\!\!\rightarrow H & : & \mathrm{g}(\mathrm{pk}(ltk_P), \mathrm{pk}(ltk_S), N_P, N_S) \ / \ V_P \\
S \hookrightarrow\!\!\!\rightarrow H & : & \mathrm{g}(\mathrm{pk}(ltk_P), \mathrm{pk}(ltk_S), N_P, N_S) \ / \ V_S \\
H & : & \mathrm{compares}(V_P, V_S)
\end{array}
$$

Figure 9.5: Bluetooth SSP Numeric Comparison association model.

Proof In the protocol step where the human agent is supposed to compare the numeric numbers V_P and V_S, Rule (9.21) can be applied and hence the exchanged nonces' authenticity does not hold. We provide a proof using the Tamarin tool. □

The next proposition shows that the Helbach code voting variant is not robust against comparing confusion with respect to verifiability.

Proposition 9.21 *The Helbach code voting variant is not (A, icc)-robust with respect to the verifiability property (p_{verif}, q_{verif}).*

Proof The Tamarin tool finds the trivial trace where the adversary intercepts the voting code and simply returns a fresh or public term instead of a valid verification code. Since the voter does not properly compare the received verification code to the codesheet, he accepts the adversary's term and sends the corresponding finalization code, hence, verifiability does not hold. That is, the election authority does not learn the communicated candidate but the voter assumes his vote to be received correctly. □

Differently to verifiability, with respect to authenticity and confidentiality, the Helbach code voting variant is robust against comparing confusion.

Proposition 9.22 *The Helbach code voting variant is (A, icc)-robust with respect to the authenticity property (p_{auth}, q_{auth}) and with respect to the confidentiality property (p_{conf}, q_{conf}).*

Proof We prove this with the Tamarin tool. Even if verifiability does not hold, the adversary may still not provide another candidates code and is not able to derive the corresponding candidate. Hence, if the voting authority receives a valid candidate code, this code must have been sent by the legitimate voter. Therefore, authenticity and confidentiality is still given. □

To improve a protocol's robustness with respect to comparing confusion, the human agent must be forced to provide specific verification information and verification is then done on the server side.

Example 9.23 *To enforce transaction authentication, the protocol could require the user to confirm the transaction by sending specific parts of the verification information back to the server. Thus, the user is at least forced to notice the verification information. However, to ensure verifiability, the server is then again required to send again verification information for this. Otherwise, the adversary could simply drop the verification information parts that the user sends back to the server without the user noticing it.*

Next, we examine humans who confuse terms in their knowledge. We show that confusing known terms is already covered with the above human-error rules.

Knowledge Confusion.

Knowledge confusion concerns erroneous human agent behavior with respect to the human agent's knowledge. Although human agents learn the received terms correctly, they may fail with handling this knowledge. For example, human agents may fail to concatenate or to select terms correctly, as it is the case in Failures (h) and (i), respectively. These two failures are implicitly covered with our model rules in $\mathcal{H}K$ or in our requirement to explicitly model, how the human agent may handle knowledge as described in Section 9.1. Moreover, human agents may confuse knowledge, as it is the case in Failure (l). For example a human agent who knows two passwords for two different services may fail to remember the correct password for the corresponding service. All these failures affect sending and comparing of terms later in the protocol run. We cover the case of sending with Rule (9.15) in $\mathcal{E}_{\text{msc}}$, which allows the human agent to send an arbitrary message from his knowledge. Regarding comparing of terms, the adversary may successfully change the order of terms, which would go unnoticed by the human agent who confuses knowledge. For example, if the human agent knows two terms x and y and is supposed to compare two terms $\langle x, y\rangle$ and m', he may concatenate x and y in the wrong order, resulting in a comparison of $\langle y, x\rangle$ with m'. But this case is covered in Failure (g), where the human agent ignores the comparison. Hence, we model the case with Rule (9.21) in $\mathcal{E}_{\text{icc}}$.

Human agents may also forget terms, as in Failure (k), which has the same effect as not to send a term from a certain point in time on and is covered in Failure (a).

Regarding our model, which emphasizes communication, knowledge confusion is a special case of sending confusion and learning confusion. We now examine failures with respect to generating weak random terms.

Weak randomness.

Whenever a human agent is supposed to provide a fresh random term, it may be not random. Hence, the adversary may guess the term, as it is the case in Failure (j). We model this erroneous behavior with the human-error rule in $\mathcal{E}_{\text{wrc}}$.

$$\mathcal{E}_{\text{wrc}} := \{\ [\mathsf{Fail}(A, wrc)] \dashv [\ \mathsf{Human}(A)\ ] \mapsto [\mathsf{Fresh}(A, m : pub)]\ \} \tag{9.22}$$

Using Rule (9.22), human agent A may choose to provide a public term m instead of a fresh term as specified in Rule (5.2).

The following example shows a common application, where weak randomness causes security issues.

Example 9.24 *For various kinds of web services, users are requested to set up a password to protect access. Users often decide to choose simple passwords that the adversary may easily guess. To prevent this, the service should enforce strong passwords based on a policy with predefined characteristics, such as minimal length, usage of special characters and numbers. Alternatively, additionally to the password, the user could be required to provide a proof of possessing a hardware token, as it is the case for two-factor authentication.*

In this section we introduced our human-error rules and examples of individual human errors. Whereas we focussed on robustness with respect to single failures, in the next subsection, we define robustness against humans who fail with respect to multiple human errors. We give an example of combined human errors and their impact on a protocol's security properties.

Combination and Repetition of Errors

Although we assume that humans at least partly comply with the protocol specification, they may be prone to more than just one human error. To analyze protocols with respect to combinations and repetitions of human errors we extend our robustness definition above to cover arbitrary multisets of human errors.

Definition 9.25 *Let a combination of human errors be represented by the multiset of failure identifiers $F \in F_{ID}^{b}$. A protocol $\mathcal{R}$ is (A, F)-robust with respect to property (p, q), if all channels provided by R having the (p, q) property, these channels have the (p, q) property in the protocol $\mathcal{R}' \cup \mathcal{E}_{all}$. $\mathcal{R}'$ represents the original protocol $\mathcal{R}$ extended with a $\mathsf{Fail}(A, f)$ fact for all $f \in F$ in the corresponding failure rule in $\mathcal{R}_{Spec}$.*

The next example shows a specific application where a combination of human errors causes authenticity to not hold.

Example 9.26 *With respect to the authenticity property, Proposition 9.12 above shows that the Helbach code voting variant is robust against message sending confusion, if the voter is assumed not to leak both, another candidate's voting code and the chosen candidate's verification code. Moreover, in Proposition 9.22, we show that the Helbach code voting variant is robust against comparing confusion with respect to authenticity. However, if the voter is assumed to omit comparison and to confuse the voting codes as well as the finalization codes, the message sending confusions may go unnoticed and authenticity does not hold. We show this in Proposition 9.27.*

Proposition 9.27 *The Helbach code voting variant is not $(A, \{msc, msc, icc\}^{b})$-robust with respect to the authenticity property (p_{auth}, q_{auth}).*

Proof We prove this with the Tamarin tool. Consider the protocol $\mathcal{R}' \cup \mathcal{E}_{all}$, where $\mathcal{R}'$ specifies the Helbach code voting variant extended with two $\mathsf{Fail}(H, msc)$ facts and a $\mathsf{Fail}(H, icc)$ fact in the corresponding failure rule for human user H. There exists a protocol run of $\mathcal{R}' \cup \mathcal{E}_{all}$ where Rule (9.15) is applied to send the candidate code for another candidate than intended. Instead of verifying the verification code, Rule (9.21) is applied. Finally, in the same protocol run Rule (9.15) is applied to confuse the finalization codes. It is obvious, that for this protocol run, authenticity does not hold, since the election authority learns another candidate than the one chosen by the voter. □

With the above error rules we modeled human misbehavior in communication applications where we expected the user to follow the specified protocol itself. In the next section, we introduce humans who do not follow the protocol but confuse the specified protocol steps.

9.4 Dishonest Client Platform

In this section, we focus on humans who deviate from the protocol with respect to its steps. In order for the user to confuse the specified protocol steps, i.e., to ignore, repeat, or reorder them, other protocol agents must support this. Otherwise, the user's options to deviate with this respect is limited. For example, a human user may only enter a message into his computer, if the user interface, representing the computer agent's role, offers him an option to do so by displaying a corresponding textfield. This support may be due to a system failure where the corresponding protocol agent does not collaborate with the adversary or due to the adversary controlling the agent as we introduced in Chapter 4.

We first analyze protocol-step confusion based on system failures before we extend our human-error model with the secure platform problem model's adversary controlling the user's computer platform.

Protocol-step Confusion and Platform Failures

Regarding the failure modes identified in Section 8.3, protocol-step confusion includes Failure (m), where the human agent may ignore a protocol step, Failure (n), where the human agent may repeat specific protocol steps a number of times, and Failure (o), where the human agent may execute the expected protocol steps in a different order than specified.

The following example shows a typical example of step confusion based on an error-prone application implementation.

Example 9.28 *In web shops, the user is often warned to not use the browser's back button or to refresh the page while placing an order or processing payments. Otherwise the order or payment processing may fail such that the same order is placed multiple times, which corresponds to Failure* (n)*.*

To model computer agents who support erroneous human behavior in the context of protocol-step confusion we define the following dishonest-agent rule in $\mathcal{DAE}$.

$$\begin{aligned}\mathcal{DAE} := \{\ [\mathsf{Sysfail}(A, c, n, c', n'), \mathsf{AgentState}(A, c, n)] \\ -[\ \mathsf{Dishonest}(A), \mathsf{Computer}(A)\ ]\!\!\rightarrowtail \\ [\mathsf{AgentState}(A, c', n')]\ \}\end{aligned} \tag{9.23}$$

Rule (9.23) models a dishonest computer agent's behavior. Note that with respect to step-confusion, dishonesty has a different meaning than with respect to insecure platforms as described in Part I. In contrast to the previous definition above in Section 5.1, dishonesty in this context defines erroneous computers. That is, computers that deviate from the specified protocol with respect to the protocol order but the computer is not under adversarial control and it does not intentionally leak information to the adversary.

To analyze protocols with respect to erroneous computers and application implementations, we extend $\mathcal{R}_{HEModel}$ with Rule (9.23) as follows:

$$\mathcal{R}_{SCModel} := \mathcal{FR} \cup \mathcal{MD} \cup \mathcal{CHE} \cup \mathcal{HK} \cup \mathcal{CP} \cup \mathcal{DAE}.$$

In many cases, ignoring a protocol step, as it is the case in Failure (m), may block the protocol. However, consider for example a protocol where the human agent receives verification information and is supposed to verify the correctness. The human agent may simply ignore the verification step.

We model humans who confuse the order of protocol steps with the following human-error rule in $\mathcal{E}_{\text{soc}}$.

$$\mathcal{E}_{\text{soc}} := \{\ [\mathsf{Fail}(A, scf), \mathsf{AgentState}(A, c, n), !\mathsf{HK}(A, c'), !\mathsf{HK}(A, n')] -[\ \mathsf{Human}(A)\ ]\!\!\rightarrowtail [\mathsf{AgentState}(A, c', n')]\ \} \tag{9.24}$$

In Rule 9.24, whenever human agent A is in a state identified with c and corresponding knowledge n, he may change the state to another state with identifier c' and corresponding knowledge n'. Thereby the user is allowed to skip one or several protocol steps. However, A may not switch roles.

Repeated application of Rule 9.24 models confusing the order of the protocol steps as it is the case in Failure (o).

Executing a protocol step repeatedly, as covered by Failure (n) allows a human agent to remain in the same state. We model this misbehavior with the following human-error rule in $\mathcal{E}_{\text{src}}$.

$$\mathcal{E}_{\text{src}} := \{\ [\mathsf{Fail}(A, src), \mathsf{AgentState}(A, c, n)] -[\ \mathsf{Human}(A)\ ]\!\!\rightarrowtail [\mathsf{AgentState}(A, c, n), \mathsf{AgentState}(A, c, n)]\ \} \tag{9.25}$$

Rule (9.25) duplicates a human-agent A's AgentState fact. Hence, it allows the human agent to be two times in the same state and thus, to repeat all subsequent protocol steps.

So far, human users were assumed to at least partly comply with the protocol specification. Human errors where explicitly modeled by Fail facts specified in the corresponding failure rule. Moreover, computer agents were assumed not to be under adversarial control and not leaking any information to the adversary. In the next subsection, we model credulous humans, i.e., humans who may deviate from the specified behavior according to any of the above introduced human error rules. Additionally we allow the adversary to control dishonest computer agents.

Credulous Humans and Insecure Client Computers

As introduced in Section 8.3, humans must not be assumed to behave as supposed by the secure system designer. Hence, similarly to the adversary, humans may interact with the protocol arbitrarily with respect to their capabilities. However, (honest but confused) humans are tied to the security protocol with respect to the available channels provided by the application's user interface. If the adversary controls the user's computer, as introduced in Part I, the adversary may trick the user into leaking confidential information and he is able to impersonate the credulous human.

In our model, credulous humans are modeled using the following rule in $\mathcal{CRH}$.

$$\mathcal{CRH} := \{\, [\ \ ] \dashv [\, \mathsf{Human}(A) \,] \mapsto [\mathsf{Fail}(A, i : pub)] \,\} \tag{9.26}$$

Rule (9.26) allows to produce arbitrary $\mathsf{Fail}(A, i)$ facts, where A is the public name of a human agent and i is a public constant referring to any failure identifier. With this rule, all of the human-error rules may be applied at any time during a protocol execution. In addition, the rules in $\mathcal{DA}$, specified in Section 5.1, may be applied, i.e., the adversary may control the honest human user's platform.

We define the credulous-human model $\mathcal{R}_{CHModel}$ as follows:

$$\mathcal{R}_{CHModel} := \mathcal{FR} \cup \mathcal{MD} \cup \mathcal{CHE} \cup \mathcal{HK} \cup \mathcal{CP} \cup \mathcal{CRH} \cup \mathcal{DA}.$$

Whereas in Part I the honest human agent was limited to execute the specified role, the credulous human is not. He may execute all possible modified protocols. The next example shows an application of this.

Example 9.29 *In phishing attacks, the adversary aims to get confidential information from the user. To do so, he tricks the user to reveal this information. For example, the adversary may provide a web page to the user that looks like the user's online bank's web page. If the user does not verify its origin, he may enter his login credentials and in turn the adversary may impersonate the user.*

It is obvious, that robustness against human agents who arbitrarily deviate from the protocol is difficult to achieve. Due to the increased complexity in these cases and the infinite protocol modifications, our modeling approach does not provide tool support in general. However, we do not focus on credulous humans and phishing

attacks in this thesis but assume honest humans to only deviate in specific ways from the protocol.

In the next section, we show a systematic analysis process to examine protocols with respect to human-error robustness and we introduce how our extended protocol model allows tool-supported verification.

9.5 Human-Error Analysis Process and Tool Support

Since our formal model is based on the Tamarin model, using the Tamarin tool to analyze a protocol is straightforward. However, with the number of additional rules, complexity is increased. To successfully analyze protocols using the Tamarin tool, we propose a systematic analysis process, where the protocol in each step is analyzed with respect to a specific subset of human-error rules. Our analysis process is depicted in Figure 9.6. We first define the human errors a given protocol shall be analyzed against. Then we generate verification tasks representing individual human errors or arbitrary combinations thereof. Finally, for each verification task, we analyze the protocol's robustness.

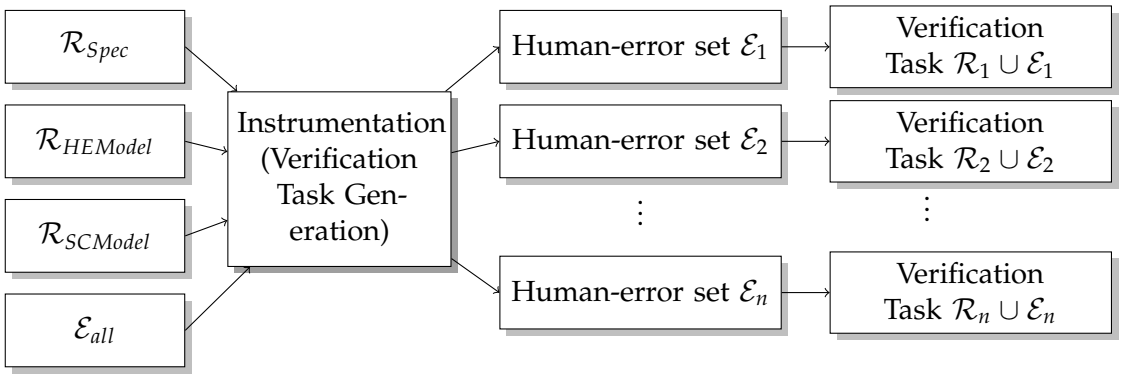


Figure 9.6: Human error analysis process.

In more detail, the process works as follows.

1. Start with the protocol specification $\mathcal{R}_{Spec}$ in our extended security protocol model (recall that for $model \in \{HEModel, SCModel\}$, $\mathcal{R} = \mathcal{R}_{model} \cup \mathcal{R}_{Spec}$ and $\mathcal{R}_{model} \cap \mathcal{R}_{Spec} = \emptyset$).

2. From the human errors represented by their failure identifiers, choose the errors the protocol shall be tested against, identify the corresponding human-error rulesets from $\mathcal{E}_{\text{msc}}$ – $\mathcal{E}_{\text{rsc}}$, and define human-error sets $\mathcal{E}_1$ to $\mathcal{E}_n$ as test cases.

3. Choose the corresponding $model \in \{HEModel, SCModel\}$ and instrument protocol specification $\mathcal{R}_{Spec}$ with the Fail facts corresponding to the verification tasks. This defines protocols $\mathcal{R}_1$ to $\mathcal{R}_n$.

4. Analyze the protocol's security properties using the corresponding model $\mathcal{R}_i \cup \mathcal{E}_i$, for all $1 \leq i \leq n$ and compare the security properties to $\mathcal{R}$'s.

Systematically applied, this process allows one to start analyzing a protocol with an error-free user and then extending the user's error-proneness by introducing more and more failures. The analysis using the Tamarin tool provides proofs for a protocol's robustness against different individual human errors and any combination thereof, or it reveals possible attacks and flaws regarding specific security properties caused by human (mis)behavior.

The classification into verification tasks allows one to define hierarchies of protocols regarding their relative strength with respect to arbitrary sets of human errors. However, we leave such applications for future research.

10 Conclusion

In this thesis, we have examined two different problem areas of secure end-to-end communication in remote Internet voting: *insecure platforms* and *human error*. For both problem areas we have focused on formal methods for analyzing communication systems and protocols. In the following we describe and summarize the conclusions in more detail.

10.1 Insecure Client Platform

We have introduced a formal model for security protocols running in an environment with humans, computers, and devices as actors. The salient feature of our model is the communication topology, which is a labeled graph whose vertices and edges represent all the protocol's actors (called nodes) and their available communication means (called links). The vertex labeling represents the nodes' knowledge and computational capabilities, and indicates whether the nodes are assumed to be compromised. The edge labeling assigns channel assumptions (such as confidential, authentic, insecure) to links.

We have used our topology model to completely characterize necessary and sufficient conditions for the existence of *human-interaction security protocols* (HISP). This is the class of security protocols where a human securely communicates with a remote server while using a compromised computer platform. This class of protocols includes Internet voting protocols. The characterization of HISPs can be used as a starting point or a blueprint to characterize other classes of protocols with respect to various security properties and at different levels of abstraction than considered in this work.

Our model is supported by Tamarin [84], a security protocol verification tool. We have provided two case studies that show concrete applications of our modeling approach and its tool support. Moreover, we extended our formal model with a formal definition of individual verifiability and analyzed the Vote électronique reference protocol's vote casting phase.

10.2 Human Errors

Humans are error-prone and unreliable. Therefore, even if a system is provably secure, system faults may well be caused by the users. To analyze the effects of erroneous user

behavior on communication applications' security properties, we have first examined the common failure modes with respect to human-server communication. Then, we have extended our formal protocol model for HISPs with a set of rewriting rules to model these failure modes.

Based on the extended formal model we have introduced a methodology to analyze security protocols with respect to erroneous user behavior. This methodology allows one to systematically test a given protocol regarding its robustness against specific failure modes or combinations thereof and it helps to identify corresponding weaknesses and improvements. We have used our model to characterize robustness against a number of the identified failure modes. These results serve as a foundation for application developers who aim to design robust and secure communication applications where one endpoint is human.

We have demonstrated our methodology's applicability with various protocol examples where we have shown that the protocols are not robust against certain human errors. Similar to our model in Part I, the extended model is supported by Tamarin. This allows the automated verification of a protocol's robustness against common human errors.

10.3 Future Research

Our models may be used to verify Internet voting systems' security properties on different levels of abstraction. For example, the nodes and links in our communication topology model can be refined for more specific applications and settings. We believe that our methodologies can serve as a starting point to model and verify more complex and more specific channel properties. An example of how additional channel properties can be modeled is that of individual verifiability in Chapter 7 above. Furthermore, temporal aspects could be considered too, for example, our communication topology model could be extended for modeling the temporal order of communication requirements. Putting together the above points, an interesting future research direction would be a refinement methodology from abstract topology models to concrete implementation specifications for secure voting systems or even, taking into account temporal aspect, to communication protocols.

Similarly to our complete characterization of secure human-server communication using insecure client platforms, a complete characterization of secure human-error-robust communication could be developed. This characterization could be specific to certain failure modes, as we describe in Part II above or to combinations of human errors. As a result all necessary and sufficient requirements for human-error robustness could be developed like we introduced for some specific classes of human-error. Specific to certain applications additional failure modes could be identified and modeled as an extension to our model. Another related research direction would be to examine the relations between the different failure modes.

We have shown, that our methodologies and tools can be applied to not only Internet voting systems, but to other security-critical communication systems too. There-

fore, it would be interesting to apply our results to other domains such as online banking. A systematic analysis of existing systems could yield weaknesses in these systems with respect to insecure client platforms and human error.

Finally, our modeling approach could be extended to environmental influences too. This would again narrow the gap between the basic idea of security ceremonies, where nothing is out-of-bound, and existing formal methods for the analysis of security protocols.

Bibliography

[1] B. Adida and C. Neff. Ballot casting assurance. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, page 7. USENIX Association, 2006.

[2] A. Alkassar, A. Sadeghi, S. Schulz, and M. Volkamer. Towards Trustworthy Online Voting. In *Proceedings of the 1st Benelux Workshop on Information and System Security–WISSec*, 2006.

[3] E. G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[4] M. Baentsch, P. Buhler, R. Hermann, F. Höring, T. Kramp, and T. Weigold. A Banking Server's Display on Your Key Chain. *ERCIM News*, pages 44–45, 2008.

[5] D. Basin and C. Cremers. Degrees of Security: Protocol Guarantees in the Face of Compromising Adversaries. In A. Dawar and H. Veith, editors, *19th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 6247 of *LNCS*, pages 1–18, Brno, Czech Republic, 08 2010. Springer-Verlag.

[6] D. Basin, S. Radomirović, and M. Schläpfer. A complete characterization of secure human-server communication. In *Computer Security Foundations Symposium (CSF), 2015 28th IEEE*. IEEE, 2015.

[7] D. Basin, P. Schaller, and M. Schläpfer. *Applied Information Security: A Hands-on Approach*. Springer, 2011.

[8] D. A. Basin and C. J. Cremers. From dolev-yao to strong adaptive corruption: Analyzing security in the presence of compromising adversaries. *IACR Cryptology ePrint Archive*, 2009:79, 2009.

[9] P. Beaucamps, D. Reynaud-Plantey, J. Marion, and E. Filiol. On the use of Internet Voting on Compromised Computers. 2008.

[10] G. Bella and L. Coles-Kemp. Seeing the full picture: the case for extending security ceremony analysis. In *Proceedings of 9th Australian Information Security Management Conference*, pages 49–55, Security Research Centre, Edith Cowan University, Perth, Western Australia, 2011.

[11] G. Bella and L. Coles-Kemp. Layered analysis of security ceremonies. In D. Gritzalis, S. Furnell, and M. Theoharidou, editors, *Information Security and Privacy Research*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 273–286. Springer Berlin Heidelberg, 2012.

[12] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553. ACM New York, NY, USA, 1994.

[13] I. Berta. *Mitigating the attacks of malicious terminals*. PhD thesis, Budapest University of Technology and Economics, 2005.

[14] I. Berta and I. Vajda. Limitations of humans when using malicious terminals. *Tatra Mountains Mathematical Publications*, 29:1–16, 2004.

[15] S. Bluetooth. Bluetooth specification version 4.0. *Bluetooth SIG*, 2010.

[16] N. Braun and D. Brändli. Swiss e-voting pilot projects: Evaluation, situation analysis and how to proceed. *Electronic Voting*, 86:27–36, 2006.

[17] N. Braun, S. F. Chancellery, and B. West. E-voting: Switzerland's projects and their legal framework–in a european context. *Electronic Voting in Europe: Technology, Law, Politics and Society. Gesellschaft für Informatik, Bonn*, pages 43–52, 2004.

[18] C. Caleiro, L. Viganò, and D. A. Basin. On the semantics of Alice & Bob specifications of security protocols. *Theor. Comput. Sci.*, 367(1-2):88–122, 2006.

[19] M. C. Carlos, J. E. Martina, G. Price, and R. F. Custódio. A proposed framework for analysing security ceremonies. In *SECRYPT*, pages 440–445, 2012.

[20] M. C. Carlos, J. E. Martina, G. Price, and R. F. Custódio. An updated threat model for security ceremonies. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1836–1843. ACM, 2013.

[21] D. Chaum. Surevote: technical overview. In *Proceedings of the workshop on trustworthy elections (WOTE'01)*, 2001.

[22] D. Chaum, R. T. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity ii: end-to-end verifiability by voters of optical scan elections through confirmation codes. *Information Forensics and Security, IEEE Transactions on*, 4(4):611–627, 2009.

[23] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *Security & Privacy, IEEE*, 6(3):40–46, 2008.

[24] Council of Europe. Recommendation on legal, operational and technical standards for e-voting. Rec(2004)11, 2004. `http://www.coe.int/t/dgap/democracy/Activities/Key-Texts/Recommendations/00Rec(2004)11_rec_adopted_en.asp`.

[25] L. F. Cranor. A framework for reasoning about the human in the loop. *UPSEC*, 8:1–15, 2008.

[26] P. Curzon and A. Blandford. Detecting multiple classes of user errors. In M. Little and L. Nigay, editors, *Engineering for Human-Computer Interaction*, volume 2254 of *Lecture Notes in Computer Science*, pages 57–71. Springer Berlin Heidelberg, 2001.

[27] S. Dekker. The field guide to human error. *Bedford, UK*, 2006.

[28] H. Desurvire, J. Kondziela, and M. E. Atwood. What is gained and lost when using methods other than empirical testing. In *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 125–126, New York, NY, USA, 1992. ACM.

[29] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[30] C. M. Ellison. Ceremony design and analysis. *IACR Cryptology ePrint Archive*, 2007:399, 2007.

[31] Statistics of Internet Elections in Estonia. `http://www.vvk.ee/voting-methods-in-estonia/engindex/statistics`.

[32] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In X. Jiang, A. Bhattacharya, P. Dasgupta, and W. Enck, editors, *SPSM'11, Proceedings of the 1st ACM Workshop Security and Privacy in Smartphones and Mobile Devices, Co-located with CCS 2011, October 17, 2011, Chicago, IL, USA*, pages 3–14. ACM, 2011.

[33] A. Filyanov, J. M. McCuney, A.-R. Sadeghiz, and M. Winandy. Uni-directional trusted path: Transaction confirmation on just one device. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 1–12. IEEE, 2011.

[34] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. Rfc 2617: Http authentication: Basic and digest access authentication. *Internet RFCs*, 1999.

[35] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ASIACRYPT '92, pages 244–251, London, UK, 1993. Springer-Verlag.

[36] S. Gajek. A universally composable framework for the analysis of browser-based security protocols. In J. Baek, F. Bao, K. Chen, and X. Lai, editors, *Provable Security*, volume 5324 of *LNCS*, pages 283–297. Springer Berlin Heidelberg, 2008.

[37] German Federal Office for Information Security (BSI). Common Criteria Protection Profile for Basic set of security requirements for Online Voting Products (Common Criteria Schutzprofil für Basissatz von Sicherheitsanforderungen an Online-Wahlprodukte), 2008. `https://www.bsi.bund.de/cae/servlet/contentblob/480284/publicationFile/29304/pp0037b_pdf.pdf`.

[38] Gesellschaft für Informatik. Information für GI-Mitglieder zu möglichen Sicherheitsproblemen auf Clientseite bei Vorstands- und Präsidiumswahlen mit dem Onlinewahlverfahren. Technical report, Gesellschaft für Informatik, `https://www.gi-ev.de/fileadmin/redaktion/Wahlen/handreichungen_gi_onlinewahlen.pdf`, 2007.

[39] R. Grimm, R. Krimmer, N. Meißner, K. Reinhard, M. Volkamer, M. Weinand, J. Helbach, et al. Security requirements for non-political internet voting. *Electronic Voting*, 86:203–212, 2006.

[40] R. Grimm and M. Volkamer. Multiple Cast in Online Voting - Analyzing Chances. In R. Krimmer, editor, *Electronic Voting 2006 - 2nd International Conference*, volume 86 of *LNI*, pages 97–106, Bonn, 2006. Gesellschaft für Informatik.

[41] R. Grimm and M. Volkamer. Development of a formal it security model for remote electronic voting systems. In *3nd International Workshop on Electronic Voting, Lecture Notes in Informatics*, pages 185–196, 2008.

[42] D. Gritzalis. Principles and requirements for a secure e-voting system. *Computers & Security*, 21(6):539–556, 2002.

[43] T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser model for security analysis of browser-based protocols. In S. Vimercati, P. Syverson, and D. Gollmann, editors, *Computer Security – ESORICS 2005*, volume 3679 of *LNCS*, pages 489–508. Springer Berlin Heidelberg, 2005.

[44] J. Heather, P. Y. A. Ryan, and V. Teague. Pretty good democracy for more expressive voting schemes. In *ESORICS*, pages 405–423, 2010.

[45] J. Helbach and J. Schwenk. Secure Internet Voting with Code Sheets. In *VOTE-ID*, pages 166–177. Springer, 2007.

[46] J. Helbach and J. Schwenk. Secure internet voting with code sheets. In A. Alkassar and M. Volkamer, editors, *E-Voting and Identity*, volume 4896 of *Lecture Notes in Computer Science*, pages 166–177. Springer Berlin Heidelberg, 2007.

[47] A. Herzberg and R. Margulies. Forcing Johnny to login safely. In V. Atluri and C. Diaz, editors, *Computer Security – ESORICS 2011*, volume 6879 of *LNCS*, pages 452–471. Springer Berlin Heidelberg, 2011.

[48] A. Hiltgen, T. Kramp, and T. Weigold. Secure internet banking authentication. *Security & Privacy, IEEE*, 4(2):21–29, 2006.

[49] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, page 70. ACM, 2005.

[50] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In D. Chaum, M. Jakobsson, R. Rivest, P. Ryan, J. Benaloh, M. Kutylowski, and B. Adida, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, LNCS 6000, pages 37–63. Springer, 2010.

[51] C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security Symposium*, volume 12, page 39, 2005.

[52] C. Kuo. *Reduction of End User Errors in the Design of Scalable, Secure Communication*. PhD thesis, Pittsburgh, PA, USA, 2008. AAI3311778.

[53] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Berlin Heidelberg, 1996.

[54] T. Matsumoto. Human–computer cryptography: An attempt. *Journal of Computer Security*, 6(3):129–149, 1998.

[55] U. Maurer and P. Schmid. A calculus for secure channel establishment in open networks. *Computer Security – ESORICS 94*, pages 173–192, 1994.

[56] M. McGaley and J. Gibson. A critical analysis of the council of Europe recommendations on e-voting. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, page 9. USENIX Association, 2006.

[57] C. Meadows and D. Pavlovic. Formalizing physical security procedures. In A. Jøsang, P. Samarati, and M. Petrocchi, editors, *Security and Trust Management*, volume 7783 of *LNCS*, pages 193–208. Springer Berlin Heidelberg, 2013.

[58] C. Meyer and J. Schwenk. Lessons learned from previous ssl/tls attacks-a brief chronology of attacks and weaknesses. *IACR Cryptology ePrint Archive*, 2013:49, 2013.

[59] mIDentity from KOBIL. `http://www.kobil.com/index.php?id=49&L=0`.

[60] L. Mitrou, D. Gritzalis, and S. Katsikas. Revisiting legal and regulatory requirements for secure e-voting. In *Security in the Information Society*, pages 469–480. Springer, 2002.

[61] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In M. Backes and P. Ning, editors, *Computer Security – ESORICS 2009*, volume 5789 of *LNCS*, pages 337–354. Springer Berlin Heidelberg, 2009.

[62] J. Nielsen and T. K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 206–213. ACM, 1993.

[63] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 249–256. ACM, 1990.

[64] K. A. Olsen and H. F. Nordhaug. Internet elections: Unsafe in any home? *Commun. ACM*, 55(8):36–38, Aug. 2012.

[65] R. Oppliger. Addressing the Secure Platform Problem for Remote Internet Voting in Geneva. *e-SECURITY Technologies Rolf Oppliger Beethovenstrasse*, 10, 2002.

[66] R. Oppliger. How to address the secure platform problem for remote internet voting. *SIS*, 2:153–173, 2002.

[67] R. Oppliger, J. Schwenk, and J. Helbach. Protecting Code Voting Against Vote Selling. In *Sicherheit*, pages 193–204, 2008.

[68] R. Oppliger, J. Schwenk, and C. Löhr. Captcha-based code voting. In R. Krimmer and R. Grimm, editors, *Electronic Voting*, volume 131 of *LNI*, pages 223–222. GI, 2008.

[69] D. Otten. Mehr Demokratie durch Internetwahlen? Presentation at Nixdorf Forum in Paderborn, 2005.

[70] D. Pavlovic and C. Meadows. Actor-network procedures. In R. Ramanujam and S. Ramaswamy, editors, *Distributed Computing and Internet Technology*, volume 7154 of *LNCS*, pages 7–26. Springer Berlin Heidelberg, 2012.

[71] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of man-machine studies*, 36(5):741–773, 1992.

[72] M. Polychronakis, P. Mavrommatis, and N. Provos. Ghost turns zombie: Exploring the life cycle of web-based malware. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 11:1–11:8, Berkeley, CA, USA, 2008. USENIX Association.

[73] S. Popoveniuc and P. L. Vora. Remote ballot casting with captchas. `http://www.seas.gwu.edu/~poorvi/RemoteBallotCasting.pdf`, 2008.

[74] L. Qiu, Y. Zhang, F. Wang, M. Kyung, and H. R. Mahajan. Trusted computer system evaluation criteria. In *National Computer Security Center*. Citeseer, 1985.

[75] R. Rivest and W. Smith. Three voting protocols: ThreeBallot, VAV, and Twin. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, page 16. USENIX Association, 2007.

[76] R. Rukšėnas, P. Curzon, J. Back, and A. Blandford. Formal modelling of cognitive interpretation. In *Interactive systems. Design, specification, and verification*, pages 123–136. Springer, 2007.

[77] R. Rukšėnas, P. Curzon, and A. Blandford. Modelling and analysing cognitive causes of security breaches. *Innovations in Systems and Software Engineering*, 4(2):143–160, 2008.

[78] P. Ryan and S. Schneider. Prêt à voter with re-encryption mixes. *Computer Security–ESORICS 2006*, pages 313–326, 2006.

[79] P. Ryan and V. Teague. Pretty good democracy. In *Proceedings of the 17th International Workshop on Security Protocols, Cambridge, UK*, 2009.

[80] N. Sastry, T. Kohno, and D. Wagner. Designing voting machines for verification. In *USENIX Security Symposium*, 2006.

[81] M. Schläpfer, R. Haenni, R. Koenig, and O. Spycher. Efficient vote authorization in coercion-resistant internet voting. In A. Kiayias and H. Lipmaa, editors, *E-Voting and Identity*, volume 7187 of *Lecture Notes in Computer Science*, pages 71–88. Springer Berlin Heidelberg, 2012.

[82] M. Schläpfer and M. Volkamer. The secure platform problem: Taxonomy and analysis of existing proposals to address this problem. In *Proceedings of the 6th International Conference on Theory and Practice of Electronic Governance*, pages 410–418. ACM, 2012.

[83] L. Schmid. Human errors in secure communication protocols. Master's thesis, Institute of Information Security, ETH Zurich, 2015.

[84] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie–Hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 78–94. IEEE, 2012.

[85] B. Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[86] B. Schneier. The solitaire encryption algorithm, 1999. `http://www.schneier.com/solitaire.html`.

[87] B. Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2011.

[88] G. Schryen. How security problems can compromise remote internet voting systems. *Electronic Voting in Europe Technology, Law, Politics and Society*, page 9, 2004.

[89] J. W. Senders and N. Moray. *Human error: Cause, prediction, and reduction*. L. Erlbaum Associates Hillsdale, NJ, 1991.

[90] G. Skagestein, A. V. Haug, E. Nødtvedt, and J. Rossebø. How to create trust in electronic voting over an untrusted platform. *GI-Edition*, 2006.

[91] O. Spycher, R. Koenig, R. Haenni, and M. Schläpfer. A new approach towards coercion-resistant remote e-voting in linear time. In G. Danezis, editor, *Financial Cryptography and Data Security*, volume 7035 of *Lecture Notes in Computer Science*, pages 182–189. Springer Berlin Heidelberg, 2012.

[92] O. Spycher, R. E. Koenig, R. Haenni, and M. Schläpfer. Achieving meaningful efficiency in coercion-resistant, verifiable internet voting. In *Electronic Voting*, pages 113–125, 2012.

[93] Trusted Computing Group. `http://www.trustedcomputinggroup.org/`.

[94] Team of oeh-wahl.gv.at. Sicherheitsempfehlung für Endbenutzer. Technical report, `http://www.oeh-wahl.gv.at/Content.Node/E-Voting_Sicherheitsempfehlung.pdf`, 2009.

[95] Trojanersichere Online Accounts. `http://www-ti.informatik.uni-tuebingen.de/~borchert/Troja/`.

[96] UPnP Security Working Group. UPnP™ security ceremonies design document, October 2003.

[97] L. van Doorn. Trusted computing challenges. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, STC '07, pages 1–1, New York, NY, USA, 2007. ACM.

[98] M. Volkamer, A. Alkassar, A. Sadeghi, and S. Schulz. Enabling the Application of Open Systems Like PCs for Online Voting. In *Proc. of Workshop on Frontiers in Electronic Elections*. Citeseer, 2006.

[99] M. Volkamer and D. Hutter. From legal principles to an internet voting system. *Electronic Voting in Europe Technology, Law, Politics and Society*, 2004.

[100] T. Weigold and A. Hiltgen. Secure confirmation of sensitive transaction data in modern internet banking services. In *Internet Security (WorldCIS), 2011 World Congress on*, pages 125–132. IEEE, 2011.

[101] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch. The Zurich Trusted Information Channel – An Efficient Defence Against Man-in-the-middle and Malicious Software Attacks. *Trusted Computing-Challenges and Applications*, pages 75–91, 2008.

[102] A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium*, volume 99, page 16. McGraw-Hill, 1999.

[103] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune. Building verifiable trusted path on commodity x86 computers. In *Security and Privacy (S&P), 2012 IEEE Symposium on*, pages 616–630. IEEE, 2012.